MINING AND MODELING THE OPEN SOURCE SOFTWARE COMMUNITY

A Dissertation

Submitted to the Graduate School
of the University of Notre Dame
in Partial Fulfillment of the Requirements
for the Degree of

Doctor of Philosophy
in
Computer Science

by

Jin Xu, M.S.

Gregory Madey, Director

Graduate Program in Computer Science and Engineering
Notre Dame, Indiana
April 2007

UMI Number: 3299241

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

MINING AND MODELING THE OPEN SOURCE SOFTWARE COMMUNITY

Abstract

by

Jin Xu

The success of Open Source Software (OSS) has attracted increased interest in many research areas. Unlike proprietary closed software, OSS projects are developed in a distributed and decentralized way. The OSS community is largely composed of part-time developers. These developers have developed a substantial number of outstanding technical achievements. A research study on how OSS developers interact with each other and how projects are developed will help researchers understand the success and failure of OSS projects. OSS developers can also benefit from this research, by being able to make more informed decisions for participating on OSS projects.

In this dissertation, we address the challenge of efficiently mining data from OSS web repositories and building models to study OSS community features. Data collection for OSS study is nontrivial since most OSS projects are developed by distributed developers using web tools. Most previous studies focus on manually creating a web crawler to collect data from OSS web sites. This method is usually implemented by creating a web crawler based on specific research goals. We design a mining process which combines web mining and database mining together to identify, extract, filter and analyze data. We address and analyze the difficulty

of mining OSS data. Our work provides a general solution for researchers to implement advanced techniques, such as web mining, data mining, statistics, and algorithms to collect and analyze web repository data.

Based on our mining results, we model the OSS community as a social network, one which can be further modeled as a project network and a developer network, and study properties of these networks. Our goal is to find intrinsic mechanisms that lie in OSS networks to explain some OSS specific features such as roles of developers, communication, and reliability of the OSS community. We construct four social networks for the OSS development community at SourceForge [59]. Each social network is created by expanding the number of people with different roles in the network, moving from the core project leaders, to the core developers, to the co-developers, and finally out to active users. Social network properties such as degree distribution, diameter, cluster size, and clustering coefficient are calculated and compared for each of the expanding social networks. We elaborate on how the changing topological characteristics of the social networks may signify important capabilities for the diffusion of information, the ability to find collaborations, and the overall robustness of the OSS development community. We further find that all the social networks have scale-free properties, and the inclusion of the co-developers and active users triggers the emergence of the small-world phenomenon for the social network. We examine how these topological network properties may potentially explain the success and efficiency of OSS development practices.

To study the organization and backbones of the OSS community, we conduct the identification of the community structure on the SourceForge project network. We find that groups exist in the SourceForge project network. Furthermore, we

explore possible reasons for the formation of those groups by examining assortative mixing coefficients for projects categories. Among them, we find projects with same programming languages, operating systems and topics are more likely to be grouped together. Our research provides useful information to study the interaction between projects and the communication and information flow in OSS virtual organizations.

We simulate the OSS community based on four social network models: random graphs, preferential attachment, preferential attachment with constant fitness, and preferential attachment with dynamic fitness, using two tools – Repast and Swarm. Our simulation models are fit to data from year two in the history of SourceForge. To prove the correctness of our simulations, docking experiments are performed on the Repast simulation and the Java/Swarm simulation. Our models simulate developers' actions and the growth of the OSS community. We compare properties of social networks such as degree distribution, diameter and clustering coefficient to dock Repast and Swarm simulations of four social network models. Our practice demonstrates the importance of verifications in scientific simulations. The simulation models we build can be used to forecast future development of OSS community.

To my dear parents, Lixia Yang and Peiguo Xu

CONTENTS

FIGURES

TABLES

ACKNOWLEDGMENTS

I would like to thank my parents, who continually supported and encouraged me throughout my Ph.D study, especially during my major writing time. This dissertation would not have been possible without their help and sacrifice. This dissertation is therefore dedicated to my mother, Lixia Yang, and my father, Peiguo Xu.

I would like to acknowledge a very special thank to my advisor, Dr. Madey, for his encouragement throughout my whole graduate study. I've benefited enormously from his support, professional guidance, advice, and counsel in these years.

Many thanks are given to Professor Renee Tynan and my colleagues, especially Scott Christley, Yongqin Gao, and Yingping Huang, for their invaluable and numerous constructive suggestions, discussions and cooperation. Their contributions are very much appreciated.

I would like to take this opportunity to thank all Open Source Software developers and researchers, who contribute to the development of OSS, from all around the world. Without their brilliant contributions and hard work, the OSS success would not be as noteworthy as they are today. Many of the participants unselfishly share their time, knowledge and expertise.

I would like to thank my husband, Junbo Feng, who continually stood behind me throughout the challenging life of a Ph.D student. Finally, a special thanks is given to my son, Ronald Feng, for bringing me happiness and motivation to finish

this dissertation.

The material presented in this dissertation is based in part upon work supported by the National Science Foundation, CISE/IIS-Digital Society & Technology, under Grant No. 0222829.

CHAPTER 1

INTRODUCTION

In recent years, Open Source Software (OSS) has achieved fast growth and great success [29, 51, 94]. Open Source Software is computer software whose licenses allow users to run, study, modify and redistribute the original or modified source code. OSS projects are typically developed by a large number of individuals from different organizations around the world, working together for a common goal. Among them, most are volunteers contributing to OSS development due to their interests but without direct economic reward. These distributed contributors collaborate through Internet infrastructures.

Open Source Software projects are developed in a different way from proprietary closed source software projects. According to Raymond [94], the way proprietary software projects are developed follows a cathedral model. In this model, the software development process has central management and control. Developers have their clearly defined roles in designing, managing, and implementing software. However, OSS projects are developed under a bazaar model. There is often no centralized control of OSS development. Many OSS users contribute to improve software quality by suggesting new features, submitting patches, reporting and fixing bugs. Such users are often considered as developers of OSS projects. They shorten the time between software flaw discovery and the bug fix.

The features of the bazaar model often cause OSS projects to have high quality with lower cost of development and maintenance.

Many OSS projects have gained significant market share against closed source software. For example, Apache is the No.1 web server [102]; Sendmail is the leading email server [100]; Linux is a popular operating system; Perl and PHP are widely used scripting languages [93]. Furthermore, OSS has begun to draw attention from major corporations. Many large proprietary software companies such as IBM, SUN Microsystems, and HP have invested in OSS development. These companies pay their employees to participate in OSS projects [97].

The success of Open Source Software has increased the interest in studying its nature in many research areas i.e., computer software engineering, economics, psychology, and organization science, etc. Among them, some studies focus on analyzing motives of individuals or organizations to engage in the OSS development. Hars and Ou [47] study the motivations of individuals who actively participate in Open Source projects. Bitzer et al [11] build models to explain the intrinsic motives of Open Source Software developers. Rossi et al [97] provide empirical evidence of the motivations of the software companies which enter the OSS market aiming at making profit. Feller et al [29] create a framework of organizations' motivations to engage in OSS. Some researchers conduct studies of OSS on the software engineering field. Crowston et al [23, 24] identify the success factors involved in OSS projects. Scacchi et al [65, 98] study the development process, work practice, and community dynamics in OSS development. Impact of OSS phenomenon on other fields is also addressed in some research. Based on the OSS model, Von Hippel [106] argues that manufacturers should follow user innovations to redesign their innovation processes. Dalle et al [26] discuss the modes of organi-

zation, governance and performance of OSS to assess the potentialities of utilizing the OSS model for other knowledge and information-goods production.

A large number of these OSS research projects collect data using surveys. Researchers design questions according to their research objectives and send out questionnaires to a specific group of samples. Analysis can be conducted based on the responses from those samples. This method is widely used in the study of OSS [49, 50, 67, 89, 97]. Data collected in this way is incomplete and biased, because it may be affected by researchers' and samplers' subjective behaviors.

The internet has become a major communication tool used by OSS developers. Online public archives contain data and records about the history of OSS projects, which is more thorough and thus a good resource for OSS study. Most of these public archives are freely accessible. There is a trend to use these archives in OSS studies [23, 25, 65, 98]. Although their goals are different, these researchers extract information on projects and developers from public archives on OSS hosting web sites.

Collecting data either way is a non-trivial task. Due to the difficulty of data collection, most OSS studies are qualitative in nature. Moreover, even for quantitative studies, the number of sample projects of current OSS research is in the hundreds, which is not big enough to reflect the OSS phenomenon. Relationships between different OSS projects and developers have not been fully explored. We still lack simulation models of OSS development. As a result, the state of the art needs to be improved to give a more thorough explanation on the motivations and working processes under OSS development.

1.1 Social Network Study on OSS

Prior research suggests that the OSS network can be considered as a complex and self-organizing system [16, 17, 38, 39, 72, 73, 75, 76]. The OSS community is largely composed of part time developers, who have developed a large number of outstanding technical achievements. A research study on how the OSS developers interact with each other and how projects are developed will help researchers understand the success and failure of OSS projects. OSS developers can also benefite from this research to make more informed decisions for participating on OSS projects.

In this dissertation, we model the OSS community as a social network, one which can be further modeled as a project network and a developer network, and study properties of these networks. Our goal is to find intrinsic mechanisms that lie in OSS networks to explain some OSS specific features such as roles of developers, communications, and reliability of the OSS community.

As we discussed above, OSS projects are typically developed collaboratively over the Internet. OSS hosting web sites provide archives to record the history of OSS projects development. We can mine information about OSS, i.e. projects statistics, developers data, and their interactions, through these public online archives. All our research data is from SourceForge.net, the largest OSS hosting web site, sponsored by VA Software [61]. Our initial data collection is through web mining on SourceForge web pages. Then, with an agreement with VA Software to provide a copy of their internal database, we conduct data mining on the extensive data which contains more complete information about projects and developers. We sum up our data collection methods into a web and data mining research framework which implements statistical, mathematical, algorithmic, web mining,

and data mining techniques, etc.

Based on the data collected, we model the OSS community as a social network. We conduct social network analysis at the community, project, and developer levels. Social network properties are explored to study the OSS project and developer network structure and evolvement. Our OSS study investigates behaviors, interactions, and mechanisms across multiple projects. We simulate the life cycle of projects, the behaviors of developers and their relationships using two agent-based simulation toolkits – Swarm and Repast. A docking process is performed to validate our models.

1.2 Contributions

Our study of Open Source Software has made significant contributions to the understanding of OSS development processes. First of all, this dissertation addresses the challenge of efficiently mining data from OSS web repositories. Most previous studies focus on manually creating a web crawler to collect data from OSS web sites based on specific research needs. We design a mining process which combines web mining and database mining together to identify, extract, filter and analyze data. Our work provides a general solution for researchers to implement advanced techniques, such as web mining, data mining, statistics, and algorithms, to collect and analyze web repository data.

Our research provides a comprehensive social network analysis of the Open Source Software development community. We make a classification of OSS development community according to developer roles in OSS projects development. Statistical and social network properties are explored to find the effect of different roles in the OSS development community. We identify that peripheral develop-

ers and users are important in making the OSS network more efficient. Based on these social network topological properties, we discover that the OSS development community is a small world and self-organizing network. Our work provides quantitative analysis on OSS community structure, formation, and growth information which is crucial to explain the information flow and robustness of OSS community. Such features are closely related to the success of OSS projects.

We build agent-based simulations to model the growth of OSS network. Our simulation models describe developers' behavior in OSS development and fit the growth of OSS network. Such a model can be used to predict future evolvement and calculate missing historical data. Our simulation models are validated by a docking process. We validate simulation results from two agent-based simulations using two toolkits – Swarm and Repast. Our validation demonstrates the transformation between simulation tools in scientific modeling processes.

Our work has further impact on understanding virtual communities. The OSS community is a typical example of virtual communities. The study of the OSS community can help researchers understand other forms of virtual communities. The benefits and lessons we get from OSS can also be applied to other virtual communities. Research in OSS from other areas such as economics, psychology, and management can also benefit from our models and explanations of OSS network.

1.3 Organization

The remainder of this dissertation is organized as follows. The next chapter describes how we collect data for our research. Our research data is from the largest OSS hosting website - SourceForge.net. We discuss advantages and difficulties in data collection from OSS web sites. A data collection framework is

given in this section. Examples of web mining and database mining are provided through the description of several small research investigations on the OSS phenomenon. Chapter 3 presents OSS community analysis based on social network perspectives. We discuss social network topological properties such as diameter, clustering coefficient, clusters and their sizes, and apply these properties on the OSS network. Furthermore, we study the effect of peripheral developers and users on OSS development. Chapter 4 continues our social network analysis on OSS network. We compute community structures on the OSS network and give explanations of the existence of community structures. Chapter 5 describes the validation of OSS simulations. Two simulation models using Swarm and Repast are compared on four different network models. Chapter 6 summarizes the entire dissertation and presents future research directions.

CHAPTER 2

OSS DATA COLLECTION METHODOLOGIES

2.1 Introduction

In this chapter, we describe the data collection methodologies we use in this dissertation. All our research studies are based on the data retrieved by these methodologies. This chapter gives a general data collection framework on OSS studies.

Most OSS projects are developed in a distributed and decentralized environment[1]. The number of developers of OSS projects ranges from several to thousands of people, who are usually not physically located in the same place. They rarely meet face to face. For example, the Linux development team consists of more than $3,000$ developers from over 90 countries [82]. Although some companies pay their employees to contribute to OSS projects, few developers belong to the same organization. Often, many developers work for their individual interests [47].

Such a distributed and decentralized environment poses challenges to the management, cooperation, and communication of OSS project development. Developers may work for their own goals, rather than focus on the project goals. Furthermore, if some developers decide to leave a project, it may put the project

[1]This chapter is based on paper "A Research Support System Framework for Web Data mining Research", Workshop on Applications, Products and Services of Web-based Support Systems at the Joint International Conference on Web Intelligence (2003 IEEE/WIC) and Intelligent Agent Technology, Halifax, Canada, October 2003, pp. 37-41 [119].

at risk. Due to the geographical separation, synchronization is also a problem both for software version control and communication. Because developers have their own preference for platforms and systems, tools must be provided to support their development needs. Well-written documentation is also necessary to assist developers to understand the OSS projects they contribute to.

Some public archives are already developed to address the above needs. OSS project developers use Source Configuration Management (SCM) tools such as version control systems (CVS), Subversion, and SubMaster to control, coordinate, and record changes of source code and documentation. With SCM, geographically-separated developers can work on their own and merge their work later. Also, project leaders can control the team goal and manage the project development by controlling commits to SCM. Moreover, many OSS projects maintain trackers to manage bug reports, patch submissions, support and feature requests, and many other issues. Trackers keep detail information such as problem description, dates, submitters, updates, comments, etc. For asynchronous communication, mailing lists and discussion boards are prevalent tools. Developers can broadcast and discuss their development issues by posting messages through these tools.

There are many web sites which host OSS projects. We can classify those web sites into two groups: individual OSS web sites and group OSS web sites. Individual OSS web sites only host one OSS project or closely related projects. Some famous OSS projects usually have their own hosting web sites, such as Linux [55], Perl [57], and Apache [52], etc. On the other hand, group OSS web sites host multiple unrelated projects together. An OSS project team can register a new project on or port an existing project to these web sites. Examples of these sites include SourceForge [59], Savannah [58], and Freshmeat [54], etc. Both individual

OSS web sites and group OSS web sites use all or some of public archives we discussed in the above paragraph. Those web sites contain abundant information about projects and their developers.

Although these OSS web sites can be treated as attractive data sources for researchers, difficulties exist in collecting data. They may not have access to the backend databases, in which case, tools are needed to extract, download, parse, and port web data. The researchers must eliminate irrelevant or unnecessary items in the extracted data. Furthermore, overlapping data or incomplete data also exists. Some users may use multiple IDs or an ID may be used by multiple users. We also need to deal with missing and dirty data. For example, dumped and defunct projects must be distinguished from normal projects [63]. It is imperative for researchers to improve data quality and accuracy.

There are two ways to collect data from OSS web repositories. In most studies, the collecting process is based on retrieving web pages or files (usually by a web crawler), parsing the downloaded pages, and analyzing data for different research goals. Robles et al. [96] design a data collecting system, GlueTheos, which includes modules to download raw data from CVS repositories, analyze the source code and store data as XML files or in a SQL database. Some researchers focus on gathering, sharing, and storing OSS development data for academic research [62, 110]. These systems use a crawler to collect OSS project and developer information from OSS websites and provide ready-to-use databases for researchers. Jensen et al. [65] explore mining techniques for discovering OSS development processes by analyzing text, links, usage, and update patterns from OSS web repositories. Crob et al. [44] discuss selecting and extracting OSS web server log files and use data mining techniques to analyze those log files. Collecting data by web

crawling has several disadvantages. First, a web crawler can hardly be used for different OSS web sites because those sites have different web structures or a site may change its structure. Second, because crawling is always time- and resource-consuming, downloading data by a web crawler may affect the performance of the downloaded site and may result in a denial of service for regular users. Third, historic data are often unavailable to web crawlers because a web site usually only contains current information. The other way to collect data is to get a copy of a back-end database dump directly from the OSS web sites. Although direct access to the data dump can avoid those disadvantages of web crawling or spidering, this approach still poses several problems. The lack of documentation about data dumps may require a huge effort to understand schemas and tables. Unlike data listed directly on web pages, names and meaning of table fields may be confusing and obscured and the interpretation may not be easy. Moreover, data processing and cleaning are complicated due to the fact that there are noisier and duplicated data stored in databases.

This chapter addresses the challenge of efficiently mining data from OSS web repositories. Most previous studies focus on manually creating a web crawler to collect data from OSS web sites based on specific research needs. We design a mining process which combines web mining and database mining to identify, extract, filter and analyze data. Furthermore, SourceForge database schemas and tables are described to help researchers understand data dump structures. Our work provides a general solution for researchers to implement advanced techniques, such as web mining, data mining, statistics, and algorithms to collect and analyze web repository data.

The rest of this chapter is organized as follows: Section 2 describes web data

sources and data features. Section 3 describes the limitations of our data sources. Section 4 presents an overview of our mining process. Section 5 gives an example of using a web crawler to collect data. Section 6 focuses on design and implementation of data collection from data dumps. Conclusions and future work are given in Section 7.

2.2 Data Sources

We collect our data from SourceForge – the world's largest Open Source software development and collaboration web site. With over 140,000 projects and over 1.5 million registered users as of March 2007, SourceForge.net, sponsored by VA Software, offers a centralized place for OSS developers to control and manage OSS development by providing project web servers, trackers, mailing lists, discussion boards, and software releases, etc. Users can download and use those projects for free under an Open Source license. This site provides highly detailed information about projects and developers, including project characteristics, developers' activities, and "top ranked" developers. By studying this web site, we can explore developers' behaviors and projects' growth.

2.2.1 Project Information

SourceForge provides a summary for each project, including its name, registered data, classification, list of developers, and activity data. According to the features and properties of a project, SourceForge groups each project into different categories, called a software map. Those categories have multi-level subcategories. A project can be classified into different subcategories in the same top category. The top level category includes database environment, development

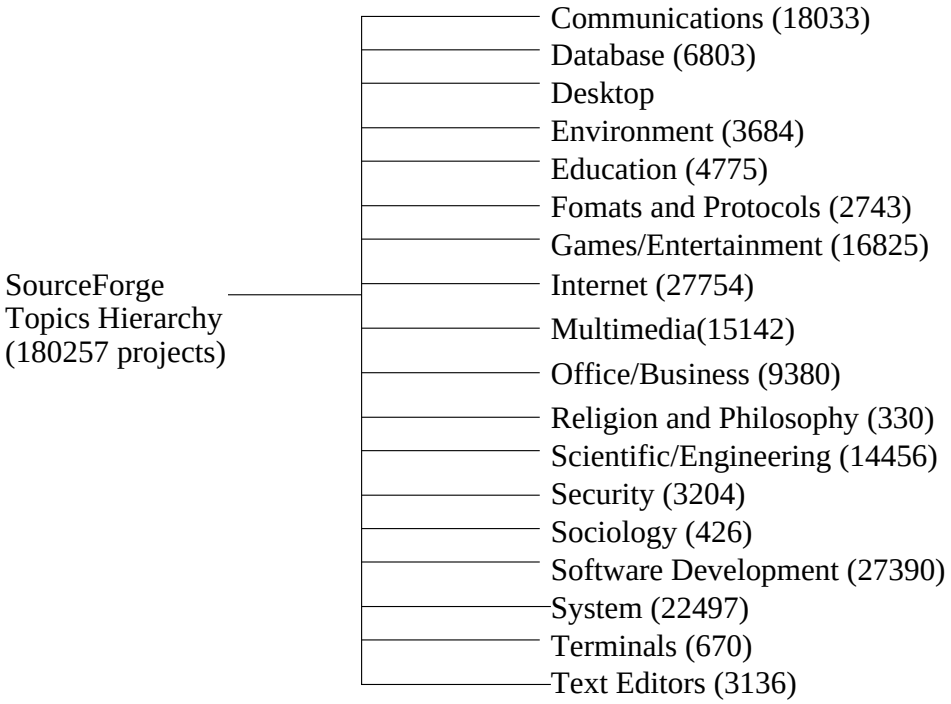


Figure 2.1. SourceForge topics hierarchy. This graph only shows the first level of the topics hierarchy tree.

status (planning, pre-alpha, alpha, beta, production/stable, mature, inactive), intended audience, license, operating system, programming Language, topic, translations (natural language the project is based on), and user interface. Figure 2.1 shows the first leve subcategories of SourceForge projects topics.

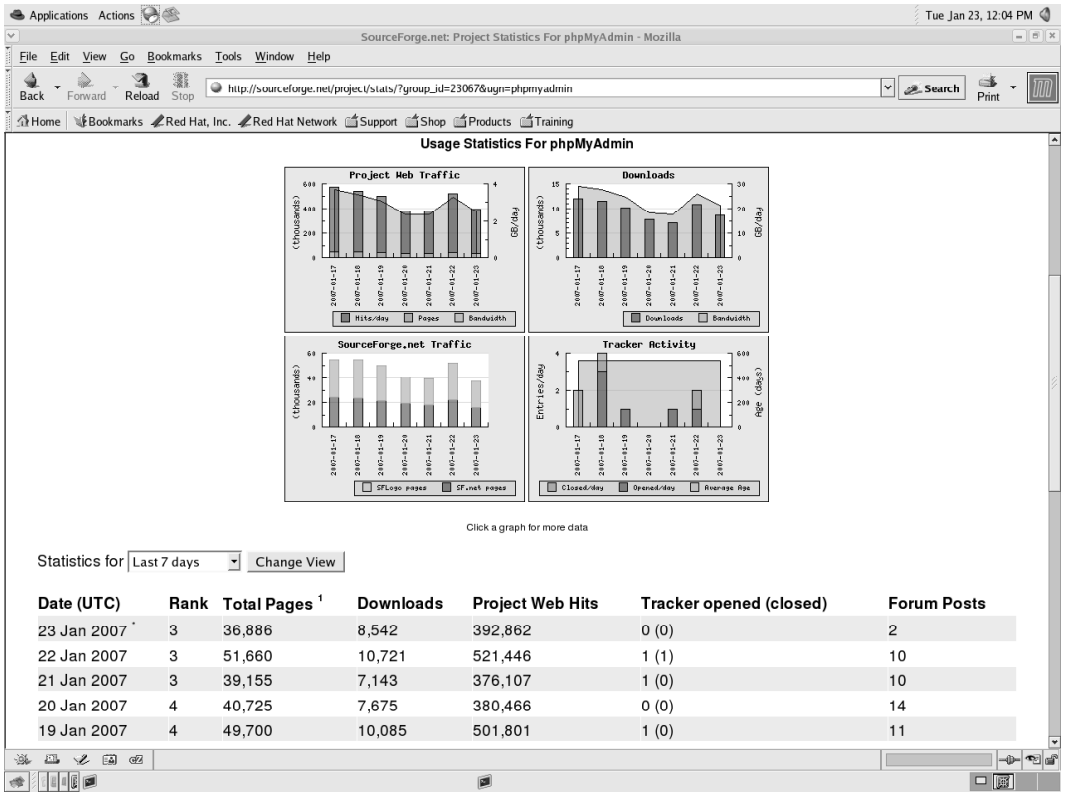
The tracker system is a tool provided by SourceForge.net that allows developers to develop, collaborate, and support software. By using trackers, developers and users can report and manage bug reports, patch submissions, support requests, feature requests. Each project may have multiple trackers. Each tracker keeps total and open counts, summary of a request, open date, update date, current status (open, close), submitter ID, and assignee ID.

Some projects may maintain forums for different types of purposes. Forums provide a convenient communication for developers and users to raise, discuss, and answer questions. SourceForge records each message, as well as its topic starter, total replies, and follow-ups.

For each project, you can find historical data of its activities, such as its rank, total page views, total downloads, and posts on forums (shown in Figure 2.2). The history of file releases is also available, which provides information about the file release time, version, size, and package names.

2.2.2 Member and User Information

A project has a list of administrators and developers. They are registered members of SourceForge. Each project has one or multiple project administrators, who are often the initiators of that project. These project administrators usually control the source code change and maintain project membership. Besides project administrators, each project may have developers who work on that project but



Date (UTC)	Rank	Total Pages [1]	Downloads	Project Web Hits	Tracker opened (closed)	Forum Posts
23 Jan 2007 *	3	36,886	8,542	392,862	0 (0)	2
22 Jan 2007	3	51,660	10,721	521,446	1 (1)	10
21 Jan 2007	3	39,155	7,143	376,107	1 (0)	10
20 Jan 2007	4	40,725	7,675	380,466	0 (0)	14
19 Jan 2007	4	49,700	10,085	501,801	1 (0)	11

Figure 2.2. SourceForge project statistics. Each day, SourceForge records rank, total number of a project's web pages, daily number of downloads, daily number of web hits, and tracker requests for each project.

have no control privileges. These developers must be registered users on SourceForge and granted membership by project administrators. Because both project administrators and developers are registered members, SourceForge keeps records of their user IDs, user names, real names, contact information, and projects they participate on.

There are a large number of registered users on SourceForge who do not belong to any projects. These users may or may not have activities on some projects. We can track their information by checking a project's trackers and forums.

However, SourceForge is also visited by a large number of other unregistered end-users, who may download software, report bugs, and post messages on forums. Since they are not registered members of SourceForge, we have no way to know their quantities and identifications although they may be very important for the development of a project.

2.3 Data Limitation

Although we can get abundant data from OSS hosting web sites, the actual data collection process has been proven to be challenging. An OSS web site can be accessed by millions of developers and users. Each person has her own natural language, input format, and other preferences. Moreover, some developers decide to move their project to an OSS web site after the project is developed for some time, while some developers may decide to stop the development of their project or move it out of an OSS web site and build their own project site. All these actions taken by developers or users can lead to difficulties in the data collection process. In summary, data collected from OSS web sites may include the following dirty or noisy data:

1. Missing data: Data collected may be incomplete. For example, since SourceForge started in November 1999, for those projects which started earlier, we could not get their history data earlier than that date.

2. Outliers: Abnormal values may be contained in the collected data. Some projects are dumped into SourceForge. These projects should be identified because they will cause inaccuracy in research studies if we treat them the same way as other projects.

3. Anonymous data: Some data may have no identification which poses difficulty in analyzing them. For example, users are allowed to use a user ID called "nobody" to post messages on SourceForge. We have no way to know how many different users are behind those "nobody" activities. When we include this data in data analysis, the way we treat it (delete all "nobody", treat each one as a different individual, or treat all as one individual) will lead to huge difference in results.

2.4 Mining Process

In order to explore the OSS data, we need to identify, extract, filter and analyze data resources. Our OSS mining process combines web retrieval, statistics, and data mining techniques together to provide information for our research. The mining process, as shown in Fig 2.3, consists of four phases: identification and extraction, cleansing and preprocessing, analysis, and report.

The first phase is to select appropriate content from the data sources. The OSS data sources may be web sites or data dumps. Usually, a web site contains many web pages, such as documentation, newsgroups, forums, mailing lists, etc. A data dump stores a large number of tables. Either web pages or database tables include

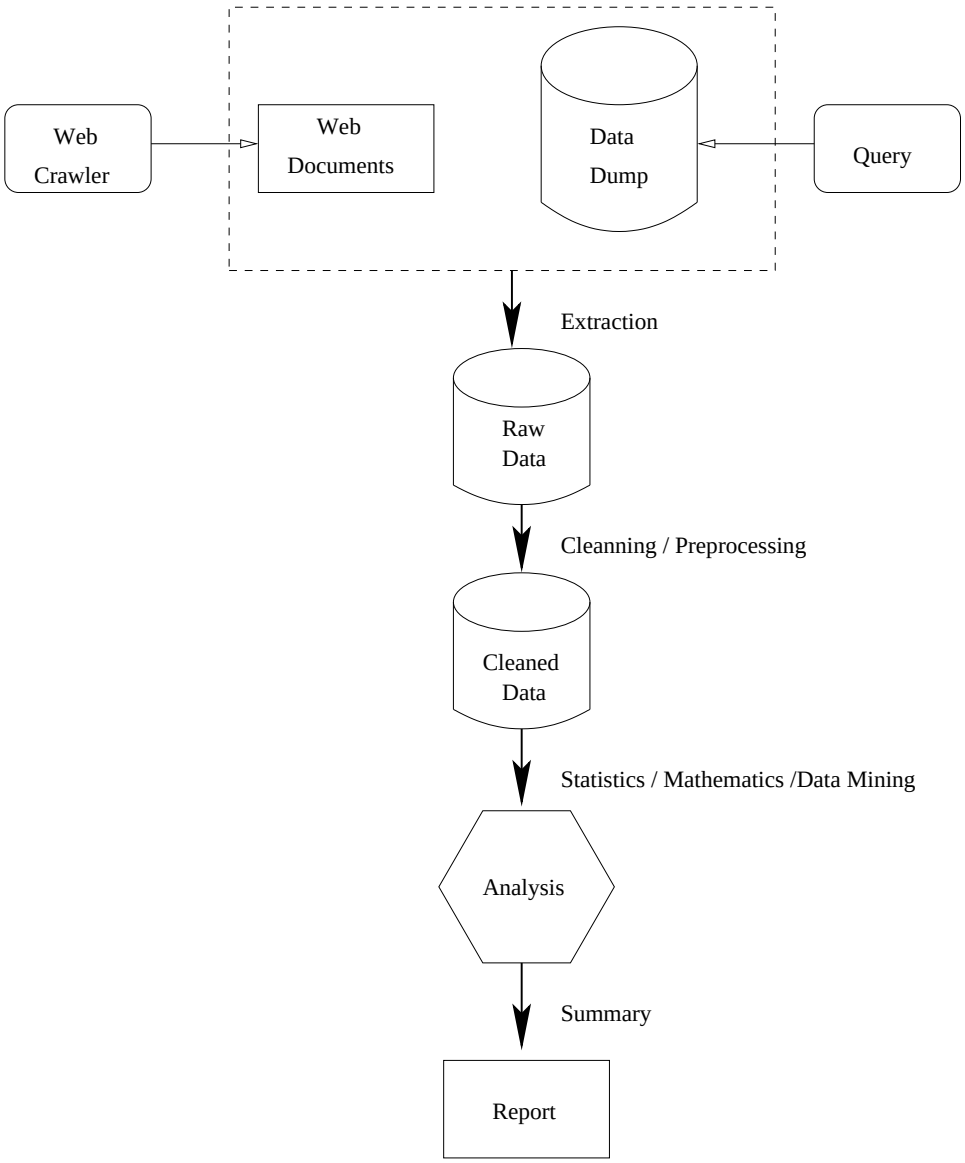


Figure 2.3. Mining process. Data are extracted from web pages or data dumps and stored into back-end database; Raw data in the database are cleaned and preprocessed for later statistics, mathematics or data mining analysis; Scientific report is based on data analysis.

relevant and irrelevant information to our research purpose. This phase decides which web pages or database tables should be extracted. Different tools and techniques will be used depending on the data source. If data is extracted from a web site, a crawler/spider will be used to automatically extract specific fragments of a web document. Extracted data from a data dump requires interpretation and exploration of database tables. Database management and connection tools, i.e. SQL and JDBC, are used in our data collection from data dumps.

The second phase converts the raw data into cleaned data abstractions necessary for analysis. The purpose of data cleansing and preprocessing is to improve data quality and increase mining accuracy. The main tasks of data cleansing consist of handling missing values, identifying outliers, filtering out noisy data and correcting inconsistent data. *Data cleansing* eliminates irrelevant or unnecessary items in the analyzed data. A web site can be accessed by millions of users. Different users may use different format when creating data. Furthermore, overlapping data and incomplete data also exist. By *Data cleansing*, errors and inconsistencies will be detected and removed to improve the quality of data. *Data preprocessing* converts data to a format suitable for later analysis, which may include integrating data from multiple sources into a coherent store; scaling values of data to a range; reducing a huge data set to a smaller representative subset.

In the analysis phase, advanced techniques are utilized to discover patterns. Statistics can be applied to calculate measures such as totals and means, etc. Data mining functions can be performed to find dependencies and relationships. Simulation models can be built to predict future development. This phase gives analytical data which needs further interpretation.

The report phase includes interpreting and validating the potential information

from patterns offered by analysis. The objective of this task is to gain knowledge from the information provided by former tasks. According to their knowledge and experience, different researchers may get different or even opposite explanations for the same analytical data.

2.5 OSS Web Mining

In the first stage of our OSS study, web mining is our main method used to collect data. Because web data are semi-structured or even unstructured, which can not be manipulated by traditional database techniques, it is imperative to extract web data to port them into databases for further handling. The purpose of our OSS web mining is to extract a specific portion of web documents useful for a research objective. Our web mining process takes web documents as input, identifies a core fragment, and transforms that fragment into a structured and unambiguous format [28].

2.5.1 Web Crawler Implementation

In our Open Source Software study, a web crawler is developed to help extract useful information from OSS web resources [119] A web crawler is a program which automatically traverses web sites, downloads documents and follows links to other pages [69]. It keeps a copy of all visited pages for later use. Most web search engines use web crawlers to create entries for indexing. They can also be used in other possible applications such as page validation, structural analysis and visualization, update notification, mirroring and personal web assistants/agents etc. [22].

We designed a web crawler shown in Figure 2.4, which is implemented using

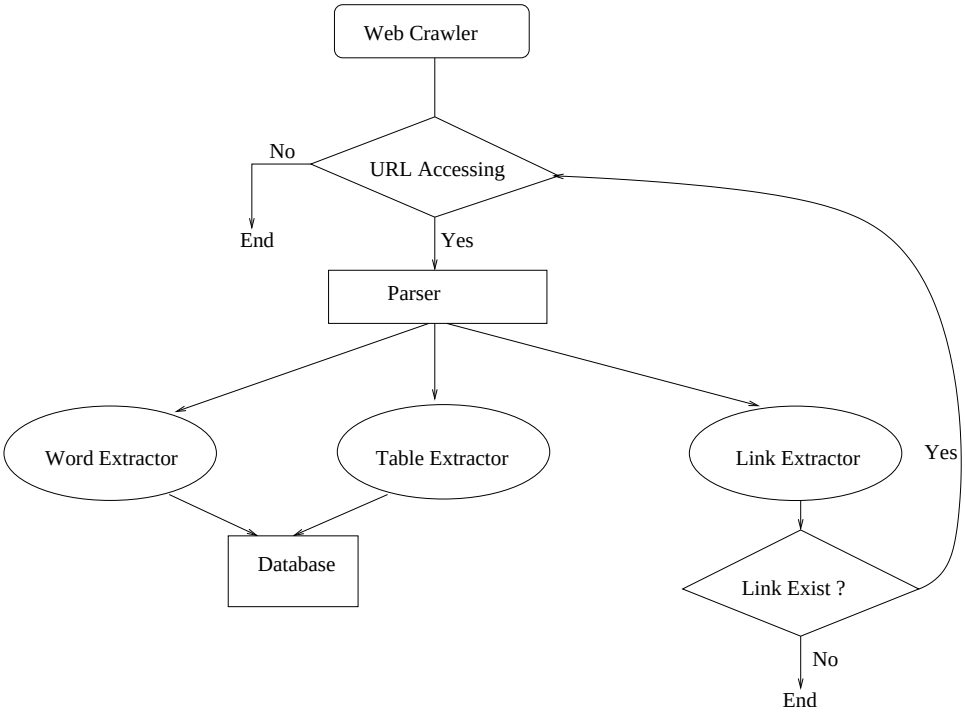


Figure 2.4. A web crawler. The crawler starts with a URL, identifies other links on the HTML page, visits those existing links, extracts and parses web pages. The parser includes a word extractor, a table extractor and a link extractor.

Perl because Perl provides useful modules to help web retrieval such as HTTP communication, HTML parsing, etc. A crawler can also be implemented in Java, Ruby, and Python, etc.

Our crawler starts with a URL, identifies other links on the HTML page, visits those links, extracts information from web pages and stores information into databases. Thus, the crawler consists of a URL access method, a web page parser with some extractors, and a back-end database. The access function of a crawler should prevent repeatedly accessing the same web address and should identify dead links. The parser recognizes start tags, end tags, text and comments. The database provides storage for extracted web information.

The key component of our web crawler is the parser, which includes a word extractor, a table extractor and a link extractor. The word extractor is used to extract word information. It should provide a string checking function. Tables are used commonly in web pages to visually align information. A table extractor identifies the location of the data in a table. A link extractor retrieves links contained in a web page. There are two types of links – absolute links and relative links. An absolute link gives the full address of a web page, while a relative link needs to be converted to a full address by adding a prefix.

2.5.2 Web Raw Data Collection

After informing SourceForge of our plans and receiving permission, we gathered data monthly at SourceForge. Data is collected at the community, project, and developer level, characterizing the entire OSS phenomenon, across multiple numbers of projects, investigating behaviors and mechanisms at work at the project and developer levels. The primary data required for this research are stored in two tables – project statistics and developers. The project statistics table, shown in Table 2.2, consists of records with 9 fields: project ID, lifespan, rank, page views, downloads, bugs, support, patches and CVS. The developers table, shown in Table 2.1, has 2 fields: project ID and developer ID. Because projects can have many developers and developers can be on many projects, neither field is a unique primary key. Thus the composite key composed of both attributes serves as a primary key. Each project in SourceForge is given a unique ID when registered with SourceForge.

Our web crawler, implemented using Perl and CPAN (Comprehensive Perl Archive - the repository of Perl module/libraries) modules, traversed the Source-

Forge web server to collect the necessary data. Appendix A gives an implementation of our web crawler. All project home pages in SourceForge have a similar top-level design. Many of these pages are dynamically generated from a database. The web crawler uses LWP, the libwww-Perl library, to fetch each project's homepage. CPAN has a generic HTML parser to recognize start tags, end tags, text and comments, etc. Because both statistical and member information are stored in tables, the web crawler uses an existing Perl Module called *HTML::TableExtract* and string comparisons provided by Perl to extract information. Link extractors are used if there are more than one page of members. We use several data mining algorithms to study data extracted by the we crawler. Analysis of OSS web retrieved data using standard data mining tools and techniques is described in Appendix B.

TABLE 2.1

MEMBERSHIP FOR EACH PROJECT

project ID	Login ID
1	agarza
1	alurkar
1	burley
1	chorizo
2	atmosphere
2	burley
2	bdsabian

TABLE 2.2

PROJECT STATISTICS.

project ID	lifespan	rank	page views	downloads	bugs	support	patches	all trackers	tasks	cvs
1	1355 days	31	12,163,712	71,478	4,160	46,811	277	52,732	44	0
2	1355 days	226	4,399,238	662,961	0	53	0	62	0	14,979
3	1355 days	301	1,656,020	1,064,236	364	35	15	421	0	12,263
7	1355 days	3322	50,257	20,091	0	0	0	12	0	0
8	1355 days	2849	6,541,480	0	17	1	1	26	0	13,896

Table note: Each row in this table reflects activities of a project.

2.6 Example of Data Dump Mining

Web mining of OSS is our initial stage in this project. Most of our subsequent OSS study uses SourceForge data dumps. Appendix F gives all tables we use in this dissertation. The SourceForge website is supported by a "back-tier" relational database built on PostgreSQL. Starting from 2003, we receive monthly data dumps from SourceForge [27, 35, 88]. Each data dump contains approximately 100 tables which record all activities and changes on SourceForge hosting projects and registered users. Compared to web retrieval, extraction from data dumps is more convenient and complete. However, SourceForge does not provide us definitions of these database tables, as well as their relationships. In this section, we demonstrate how to investigate tables and extract data from data dumps by a real research project.

2.6.1 Research Background

Although there are many successful OSS projects, a large number of OSS projects fail in their development or even stop in their early stages. On SourceForge, there are more inactive projects than active ones. Thus, one goal for OSS research is to identify factors related to effective OSS projects.

It's interesting to notice that the initial condition of OSS project development is almost the same. For example, on SourceForge, each project has the same potential pool of developers and users; SourceForge provides the same software development environment and management tools for the hosting projects. The difference lies in the reputation of core developers, the goal of the project, task characteristics, motivation, and communication. We propose a set of hypotheses based on Hackman's model of team effectiveness [46] to analyze the relationship

between developer interaction, task characteristics, and outcome in OSS projects (Hypotheses and results are reported in [104]) and use data from SourceForge data dumps to support our hypotheses. In this dissertation, we only present the data mining process.

Because it is impossible to include the whole population of OSS projects in our research, a sample of projects is created across which statistical analysis can be applied. The sample projects are selected from the SourceForge data dump generated in February 2005. An appropriate set of samples should contain both effective and ineffective projects with the same initial development conditions.

2.6.2 Sample Selection Algorithm

In order to sufficiently test our hypotheses, we select 100 effective projects and their size matched controls. The selection algorithm is as follows:

- Step 1: Select effective projects

    1. All inactive projects are excluded from our selection.
    2. The top 20% of active projects in each SourceForge category are identified.
    3. With proportional representation of categories, 400 projects are randomly selected from the top 20% active projects as selected in 2. We need more than 100 effective projects because we may discard some if we can not find their size matched controls or if they were imported into SourceForge.

- Step 2: Select size matched controls

1. The number of developers of each of 400 effective projects at the first software release date is calculated as the size of the project.

2. The size matched control project of each effective project is determined. The size matched control project should be in the same category and has the same number of developers at the same first release date as its corresponding effective project. Effective projects and their size matched controls have the same starting time and have the same active development period.

- Step 3: Mix effective projects and their size matched control

1. 100 effective projects and their size matched controls are selected respectively from the 400 effective projects pool and their control matched projects pool.

2. All of the projects and their size matched controls are reassigned new numbers, so that raters can rate all projects without the information whether a project is an effective one or a size matched control project.

2.6.3 Methodology and Sample Statistics

All our sample projects are selected from SourceForge data dump in February 2005. During that time, the total number of projects in the data dump is 124393, in which 90200 projects are active. Even in those active projects, many only have a very few developers or user activities, which makes our analysis difficult. We also must exclude imported projects which will cause outliers.

We find all projects which have their first file releases after February 2003. The file release information can be found by extracting data from two tables –

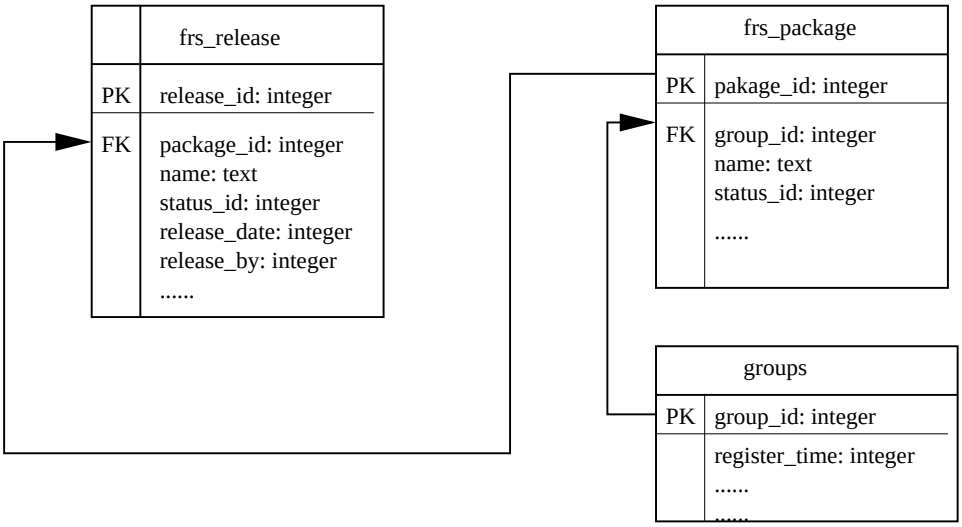


Figure 2.5. ER diagram of file releases. Records in table *groups* correspond to each project. Records in table *frs_package* represent all software packages. Records in table *frs_releases* are information about each file release. *Package_id* in table *frs_release* is the primary key in table *frs_package*. *Group_id* in table *frs_package* is the primary key in table *groups*

frs_release and *frs_package*. Table *frs_release* includes fields such as file release ID, package ID which contains this file release, file release date, and developer ID (who releases this file), etc. Table *frs_package* provides information for the software packages which contains all files and programs for the execution of the project. How to extract file releases is shown in Figure 2.5.

Many active projects have a low level of activities. In order to have enough activities to support our analysis, we restrict the sum of messages in forums and artifacts to be greater than or equal to 5. Artifacts, in the SourceForge database schema, are message boards created by project managers. The four most common artifacts on SourceForge are bug reports, feature requests, support requests, and patch submissions. Besides artifacts, users and developers can post and discuss

questions on forums. Artifacts and forums are two main available sources to observe project activities. In the SourceForge data dump schema, a field named *count* in table *artifact_agg_msg_count* and table *forum_agg_msg_count* records the total number of messages in an artifact or a forum. The extraction of artifacts and forums messages is shown in Figure 2.6.

SourceForge uses a tree structure to store project categories. The tree is stored in a table called *trove_cat*. Table *trove_cat* contains the current subcategory ID and parent ID. SourceForge classifies each project into groups according to different characteristics, such as operating systems, programming languages, topics, and end users. All classifications are stored as different values represented as a field – *trove_cat_root* in table *trove_group_link*. To extract all projects in a topic category, we retrieve each project from its bottom subcategory to its root category. Table 2.3 shows the size of the top 20% projects in each category. The relationships among trove categories are shown in Figure 2.7.

SourceForge has a rank for each project, which measures the degree of a project's popularity. The selection of the top 20% in each topic is based on this rank. We decide to exclude games, because the motivation of game developers and the dynamics on game projects development differ substantially from non-game projects. We also filter out the 5 smallest categories – religion, printing, terminals, sociology, and formats and protocols because each of them contains less than 400 projects, which is too small to support the control matching procedure. When randomly selecting 400 effective projects from the top 20% ranked projects, we guarantee a project will not be repeatedly picked because a project may appear in multiple categories. Thus, our sample projects are unique. To select size matched controls which have the same initial conditions as effective projects, we define

a size matched control of an effective project as a project which is in the same category as the effective project, with developers greater or less than 2 developers of the effective project, and the first file release date before or after 15 days of the first release date of the effective project. If more than 1 size matched control is found, we randomly pick one. We also exclude all picked projects in selecting the size matched controls. After this step, 352 out of 400 effective projects and their control matched projects are picked. Table 2.4 gives a section of our final 200 sample projects.

In order to compare communications among developers and users on effective projects and their size matched controls, we calculate the feedback quality of artifacts and forums for all sample projects. The feedback quality consists of several measurements – the number of messages without responses, the number of messages with responses, the total number of all responses, and the sum of first response times. As shown in Figure 2.8, we extract artifact responses from three tables in SourceForge data dump – *artifact*, *artifact_group_list*, and *artifact_message*. Table *artifact* contains information on artifact messages, i.e. open date, close date, the person who submits the artifact message, and the person who is assigned the artifact task. Table *artifact_message* records all response messages. The response date is stored in a field called *adddate*. We can get the number of responses to an artifact message by counting records in this table. The difference between the earliest response date and the open date is the first response time. Tables containing forum response information are *forum*, *forum_group_list*, and *forum_threadinfo*. Their relationships are shown in Figure 2.8. Each post and its follow_up messages in a forum are regarded as a thread which is recorded in table *forum_threadinfo*. Field *msg_id* is the ID of the initial message and *thread_id* is a

unique ID assigned to a thread. All forum messages are stored in table *forum*. If a message is a follow_up message, we can get the initial message ID from a field called *is_followup_to*. Furthermore, table *forum* contains other information such as posting dates, creators, and subjects. Table 2.5 lists a section of the response results.

2.7 Conclusion

In this chapter, we analyze the data sources for OSS research and describe their characteristics. Based on different data sources, a mining process is presented which consists of four parts – identification and extraction, cleansing and preprocessing, analysis, and report. Examples of web mining and data dump mining are given to give researchers a general solution to collect and analyze data from OSS data repositories.

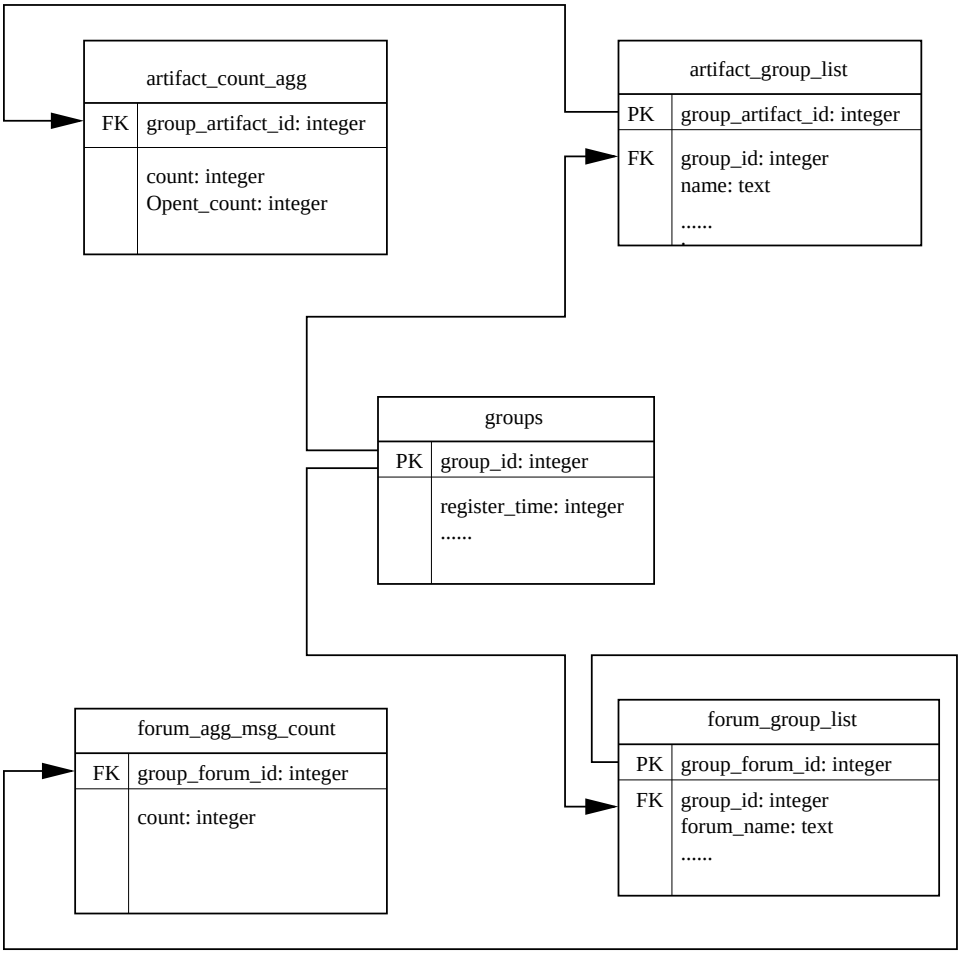


Figure 2.6. ER diagram of artifacts and forums. Table *artifact_group_list* and table *forum_group_list* contain records corresponding to each artifact and forum in every project. Table *artifact_count* and table *forum_agg_msg_count* contain the total number of messages in each artifact and forum.

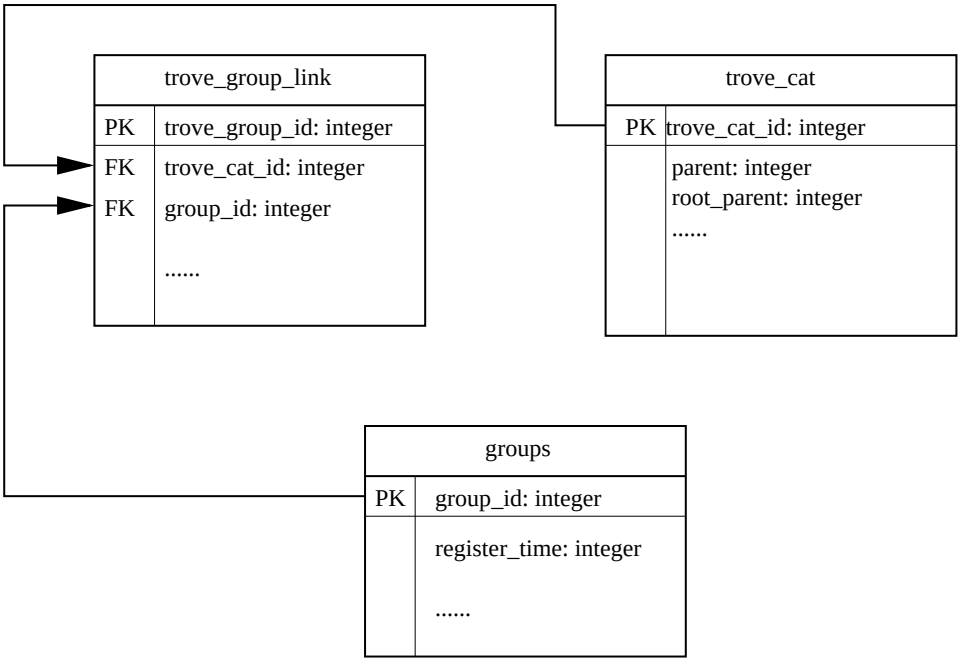


Figure 2.7. ER diagram of trove categories. Table *trove_cat* is the software categories defined by SourceForge. Table *trove_group_link* corresponds to the relationship between projects and their software categories. A project may be included into several categories.

TABLE 2.3

PROJECT CATEGORIZATION

Project Category	Number of Projects	Percentage
Formats and Protocol	70	0.098%
Religion	119	0.17%
Sociology	144	0.20%
Printing	240	0.34%
Terminals	318	0.44%
Other Topics	1079	1.51%
Security	1324	1.85%
Editors	1388	1.94%
Education	1418	1.98%
Desktop	1617	2.26%
Office	2144	3.0%
Database	2809	3.93%
Scientific	4788	6.69%
Games	5465	7.64%
Multimedia	6254	8.74%
Communications	6490	9.07%
Not Categorized	6912	9.66%
Development	8567	11.98%
System	9356	13.08%
Internet	11033	15.42%

Table note: This table contains the number of sample projects in each category. The number of sample projects is in accordance with the proportion of this category to the whole project population. For example, we pick 70 sample projects under category "Formats and Protocol" because the total number of projects in "Formats and Protocol" is 0.098% of the whole project population.

TABLE 2.4

SAMPLE PROJECTS

proj_id	topic	rank	first_release	developers	control_mark
93482	-1	74	20031116	3	0
36400	-1	-1	20031109	1	1
51618	20	1346	20020917	3	0
5315	20	-1	20020902	1	1
58425	43	1150	20020904	1	0
40844	43	3322	20020919	1	1

Table note: Please see Appendix D.1 for all sample projects.
Topic -1 means non_categoriezed topics.
Control_mark 0 means effective projects, 1 means size_matched controls.

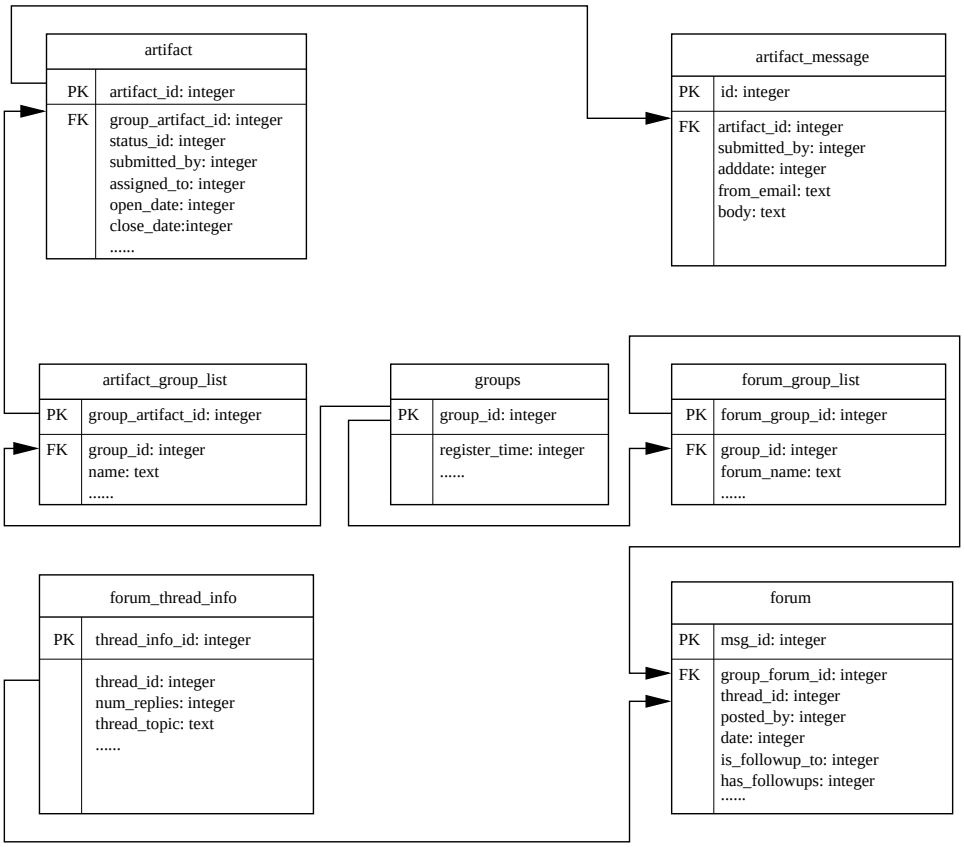


Figure 2.8. ER diagram of artifact and forum response. Table *artifact_message* or *forum_thread_info* contains information about each message in each *artifact* or *forum* for each *group*.

TABLE 2.5

SAMPLE PROJECTS RESPONSE STATISTICS.

proj_id	at_noresponse	at_response	at_close	at_num_replies	at_reply_time	forum_noresponse	forum_response	forum_num_replies	forum_reply_time
249	10	10	1	12	31914529	2	0	0	0
265	0	1	1	1	20235	1	2	2	26386
266	1	1	0	1	13798	1	1	1	41562291
285	25	170	24	307	605546139	65	97	180	151083317
308	5	127	76	216	155275283	39	47	286	137216514
376	0	2	0	2	91179146	4	2	7	37367792
404	2	28	1	36	67810636	3	3	6	41310117
521	1	2	0	2	32466597	2	0	0	0
525	22	18	2	35	2024526	6	1	1	89362418

Table notes: Please see Appendix D.1 for all data. Meanings of columns are: proj_id – Project ID; at_noresponse – the number of artifacts which have no responses messages; at_response – the number of artifacts which have response messages; at_close – the number of artifacts which are close but have no responses; at_num_replies – the total number of all response messages of at_response; at_reply_time – the sum of the first response time of at_response (seconds); forum_noresponse – the number of forum messages which have no response messages; forum_response – the number of forum messages which have response messages; forum_num_replies – the total number of all response messages of forum_response; forum_reply_time – the sum of the first response time of forum_response (seconds).

CHAPTER 3

APPLICATION OF SOCIAL NETWORK ANALYSIS TO THE STUDY OF OPEN SOURCE SOFTWARE

3.1 Introduction

There has recently been increasing application of social network analysis in many research areas [13, 45, 92, 105, 112][1]. A social network is formed when there are social interactions between individuals, e.g., friendship, marriage, or communication. Social network analysis concentrates on mapping and measuring the relationships between people, groups, organizations, or other social entities.

Newman [83] constructs collaboration networks, linked by coauthorships. Network properties are identified and studied, such as typical distances between scientists. Furthermore, he suggests a measure of centrality to study the strength of collaborative ties. Albert and Barabasi [8] study the topology of the World Wide Web and calculate its diameter. Structures of several computer networks are explored to analyze virus and worm infections in [6]. They provide a dynamic mechanism to control contagion for different computer network types. Recently, some researchers have used this method to analyze the Open Source Software

[1]This chapter is based on paper "Application of Social Network Analysis to the Study of Open Source Software", The Economics of Open Source Software Development, eds. Jurgen Bitzer and Philipp J.H. Schroer, Elsevier Press, 2006 [115]. It is also published as a paper "A Topological Analysis of the Open Source Software Development Community, The 38th Hawaii International Conference on Systems Science (HICSS-38), Hawaii, January 2005 [116]

phenomenon. Gao et al. [40, 117] analyze the core developer network on SourceForge hosted projects. They explore network properties and report the presence of scale free behavior in this network. Social network analysis is used to analyze the source code repositories of Open Source Software projects [70]. Moreover, case studies are provided to show how to apply social network analysis to explore the structure, evolution and internal process of software projects.

Open Source Software (OSS) development practices promise enormous advantages such as reduced development cost, simplified team collaboration and improved software quality. OSS development practices differ greatly from commercial software development models in several aspects: commercial software companies prevent access to the source code of their products from outside developers and customers, while open source software allows source code to be freely modified and redistributed under "open source" licenses. More importantly, unlike closed-source software which is developed by centralized development teams, OSS projects are typically developed in a distributed and decentralized way [29, 94]. OSS projects are written, developed, and debugged largely by worldwide volunteers, who in most cases are connected and collaborate solely through the Internet. The OSS development community has developed a large number of outstanding software products, including Apache, Perl, Linux, etc. The success of OSS has been attributed to effective project composition and efficient coordination mechanisms, which are due to the fact the source code is open to inspection and participation is open to any interested individual [24]. Open source software communities are one of the most successful high-performance collaboration communities on the Internet [68].

Despite the growth in OSS research, the phenomenon is not yet fully under-

stood [12, 29, 79]. The intrinsic mechanisms in the OSS development process still need to be investigated. For example, unlike closed-source software, the OSS development process involves developers who irregularly participate in projects. Moreover, roles of users in OSS may be different from those in closed-source software because they have closer contact with developers. Understanding the collaboration among the OSS community may help solve the "software crisis" [94]. Moreover, the success of a project may be related to the underlying social network topology of the OSS development community.

This chapter provides a comprehensive social network analysis of a large segment of the Open Source Software development community. Data is collected and extracted by mining a SourceForge 2003 data dump [59]. By dividing the OSS development community into four subsets, statistical and social network properties are explored to find the effect of different roles in the OSS development community. Based on these social network topological properties, we discover that there are special features in the OSS development community. We give explanations to these topological and evolutionary features and discuss their relationship to the success of OSS projects.

The rest of this chapter is organized as follows. The next section provides background on the OSS social networks and topological network properties. The third section classifies roles of developers by their activities in projects. Then, we present data extraction and mining processes used for our analysis of the SourceForge community. In section 5, we perform statistical analysis on the collected data to find the member distribution in OSS, furthermore, we study network properties to understand the topological features of the OSS development community; finally, conclusions and future work are given.

3.2 Social Network Perspectives

A social network consists of a group of actors connected through relations that hold them together [48]. An actor can be an individual or aggregate unit, e.g. organizations, groups, or other information/knowledge processing entities. Pairs of actors maintain *ties* if they have one or more relations. The behavior of actors is affected more by their ties and the social networks in which they are embedded than by the attributes they possess [111]. Thus, studying actors based on some attribute data may not be enough to show the characteristics of their relations. Social network analysis seeks to explore the network features which affect actors' interactions.

A social network can be modeled as a graph with nodes representing people or groups, and links representing relationships or information flows between nodes. Thus, two people have a link between them if they have a relationship with each other. The path between two nodes in the graph measures the closeness of those two nodes. Social network analysis has been successfully applied in many scientific areas [1]. For example, in a scientific collaboration network where nodes represent scientists, a link between two nodes might indicate that those two scientists have coauthored a paper together.

3.2.1 Open Source Software Social Networks

The Open Source Software (OSS) development movement is a classic example of a dynamic social network [70]; it is also a prototype of a complex evolving network [40, 74, 118]. Developers collaborate with each other through the Internet. We are interested in how these distant developers create and sustain such a social network. The formation of the social network begins when developers

join a project, work with others, and form co-working relationships. This social network bonding is important in maintaining developers' collaborations in the OSS decentralized development environment, because developers can feel a sense of belonging to a group, which is necessary to sustain the group as a whole entity rather than as a set of separate individuals [48]. Benefits of such a feeling include greater performance, greater cooperation, and greater satisfaction [31, 80].

The OSS social networks we constructed have two entities - developers and projects. The social network is a bipartite (two-mode, affiliation) graph, with two kinds of nodes to separately represent developers and projects. Links are formed between developers and projects signifying the relationship that a developer performs some sort of activity or has membership status for a project. As part of our analysis, we will expand the range of activities represented as links to form several larger social networks which are more inclusive of people in the community. The social network as a bipartite graph can be transformed into two unipartite graphs, the developer network and the project network, where all of the nodes are the same type of either developers or projects, respectively. When transformed to the developer network, a link between developers represents a shared project for those two developers. Most developer communication for projects in OSS development occurs through forums, discussion boards, and mailings lists which act as a form of broadcast allowing single individuals to communicate directly to the whole group, so the developer network then represents a communication network based upon the communication mediums provided by projects. When transformed to the project network, a link between two projects indicates they have one or more common developers. The developer network is used to study an individual's impact on the whole network; while the project network is used to study a project's

impact in the whole network. Both networks can be used to study how information flows and diffuses either from individual to individual through a project communication medium or from project to project through the action of an individual. Both developer and project networks are used to study the OSS phenomenon, but we primarily focus on the project network to explore how individuals tie together many of the OSS projects into a larger social community. Figure 1 shows a cluster (a collection of connected nodes) at SourceForge.net along with the corresponding developer network generated from that cluster. A visualization of a large group of projects, a sampling of the complete project network, can be seen in Figure 7.

3.2.2 Topological Network Properties

The topology of a social network can be explored by studying its network properties. Some special phenomenon are already discovered to exist in the topology of many social networks. In this section, we examine network properties in the OSS network.

Many properties are used to characterize the topology of the social network. The following properties are used in our analysis of the OSS social networks:

- Degree Distribution: The degree of a node, k, is the total number of links connected to this node. The degree distribution represents the relative frequency of each value of the node degree, k, in a given network. The degree distribution of social networks was generally believed to be the Poisson distribution meaning that links between nodes were formed randomly; however, recent research has shown that many real world social networks actually have a degree distribution which is a power law [2], which is defined as follows:

$$y = bx^{\alpha} \tag{3.1}$$

where b, α are constants. The relationship between $log(y)$ and $log(x)$ is linear.

- Diameter: The shortest path between two nodes is defined as a sequence of links going from the one node to the other node where the number of links, or length, of the path is minimized. The diameter of a social network is the longest of the shortest paths; that is, of all the shortest paths between all nodes, the path with the longest length signifies the diameter of the network, so it is the maximum number of links that have to be traversed for communication to flow from one node to another. If the network is disconnected, a path does not exist between all nodes; then, the diameter can be defined as infinity or the maximum diameter of its connected clusters. Calculating the longest shortest path is computationally expensive, so an alternative definition that is frequently used is the average shortest path. The average shortest path defines an average diameter which represents the average distance to traverse from one node to another; in this chapter, we use the average diameter because it can be approximated. Newman and Watts developed an approximate calculation of the average diameter by using a generating function [86]. The approximate average diameter in a random network can be computed by

$$d = \frac{log(N/z_1)}{log(z_2/z_1)} + 1 \tag{3.2}$$

where,

d – the diameter

N – the total number of nodes

z_1 – the average number of neighbors 1 link away

z_2 – the average number of neighbors 2 links away

Thus the average diameter of a network is increasing logarithmically with N.

- Cluster: A *cluster* in a social network consists of the set of all nodes which are connected together by some paths; that is, there exists a sequence of links that can be traversed to go from one node to any other nodes. A social network may have numerous clusters, so nodes within a cluster are connected by a path, but no path exists between two nodes in different clusters. Clusters are often considered to represent communities within a social network.

- Clustering Coefficient: The clustering coefficient of a node is defined as the ratio of the number of links to the total possible number of links among its neighbors. The clustering coefficient of a node is an indicator of the connectivity for that node, and the clustering coefficient of a social network is the average of all the clustering coefficients of the nodes. The generating function proposed by Newman and Watts can be generalized to bipartite graphs to get the clustering coefficient [1]:

$$C = \frac{1}{1 + \frac{(\mu_1 - \mu_2)(\nu_1 - \nu_2)^2}{\mu_1 \nu_1 (2\nu_1 - 3\nu_2 + \nu_3)}} \tag{3.3}$$

where $\mu_n = \sum_k k^n P_d(k)$ and $\nu_n = \sum_k k^n P_p(k)$. In the project-developer bipartite network, $P_d(k)$ represents the fraction of developers who join k

projects, while $P_p(k)$ means the fraction of projects which have k developers.

3.2.3 Strong and Weak Ties

In a social network, an actor forms different kinds of connections to others; thus, ties have different strength. Strong ties are formed between actors with similar background, experience and resources. For example, strong ties exist between friends and relatives. Actors who are strongly tied usually have an intimate or special relationship, frequent interactions, and common needs. Weak ties, by contrast, involve infrequent and limited interactions. A sales lady working in a grocery store might have a weak tie with you. You only have a relationship with her when you shop in her store.

Strong ties are important because we are more likely to exchange information with our strong ties and get direct help from them. Strong ties reinforce our companionship and provide us with support. People in a group maintain strong ties with other members in the same group. They share the same goals and work closely. The study of strong ties can help us understand the structure and local information of a social group.

Weak ties bring us more new opportunities and resources which cannot be obtained from close relations [43]. For example, although counterintuitive, weak ties may be more useful in helping a person to find a job than close friends. With weak ties, information, ideas and resources can be exchanged and flowed more broadly. Study of weak ties give us a global view of the whole community.

3.2.4 Small World Phenomenon and Scale Free Network

The *small world phenomenon* is the principle that everyone in the world can be reached through a short chain of acquaintances. In 1960s, psychologist Stanley Milgram performed a small world experiment to trace paths through the social network of residents of the United States. He found that two random persons were connected by an average of six acquaintances, which is called "six degrees of separation" [81]. Duncan Watts and Steve Strogatz provided evidence that the small world phenomenon exists in many real networks [109]. They showed that the addition of a few random links can turn a "large world" into a "small world" network. They defined a small world network to include two features – a high clustering coefficient and a small network diameter.

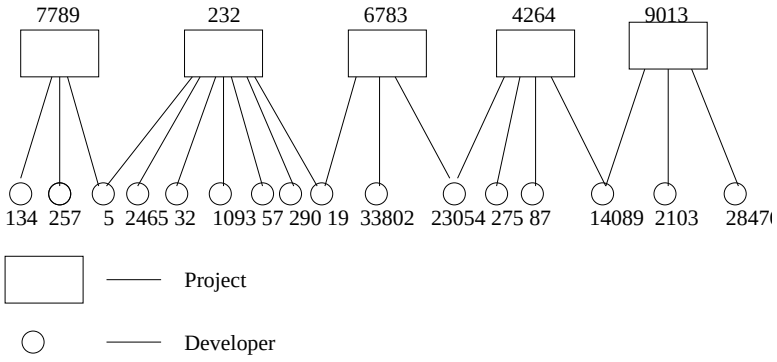
Barabasi and Albert found that some small world networks have another special property. Such networks contain relatively few nodes (called "hubs") which are highly connected to other nodes, while the vast majority of nodes are only connected to a few other nodes. These networks are called *scale-free* networks. According to Barabasi and Albert [8], such a network is generated by two rules. First, the network grows by the sequential addition of new nodes; second, there exists *preferential attachment* – the probability for a newly added node to be connected to an existing node increases with the degree of the existing node. This property is sometimes called the "rich gets richer" phenomenon. In scale-free networks, the degree distribution of nodes follows a power law distribution. Scale-free networks have been found in many networks such as power grids, the stock market, the Internet, and the spread of sexually transmitted diseases [2, 78].

Scale-free networks have different robustness characteristics compared to random networks in the presence of failures. In a random network, multiple node

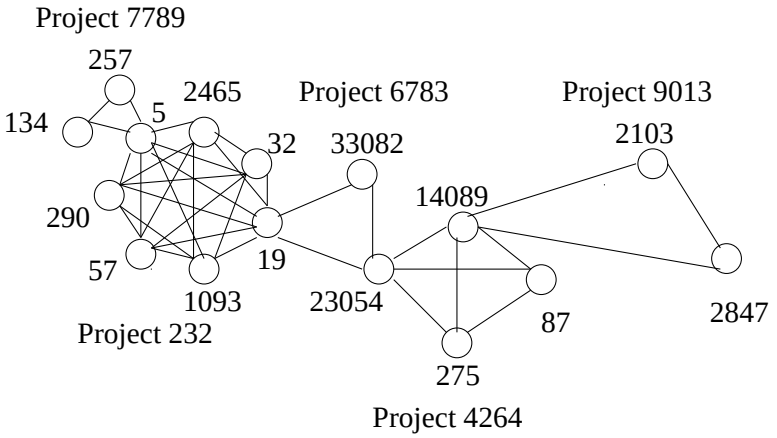
failures occurring randomly will eventually trigger a collapse of the whole network, leaving most nodes disconnected from each other. Scale-free networks have a large number of nodes with only a few connections and only a very small number of highly connected nodes, so random node failures will with high probability only disconnect one of the low connectivity nodes. Thus, such random failures do not affect the network and leave its structure intact. However, if failures are not random and are targeted at the hubs, then the network can collapse leaving it fragmented in smaller clusters. A highly connected hub becomes a single point of failure, but knowledge of the scale-free structure of the network may prove useful; for example, the spread of disease through a disease transmission network can be halted by controlling and quarantining the hubs. The robustness property of scale-free networks is important for communication and collaboration networks because information and resources can easily and quickly diffuse through the network even though nodes are continuously joining and leaving the network.

3.3 OSS Development Community

Different member roles exist in an OSS project. Usually, an open source software project is initiated by an individual or a small group of people with ideas they share for some intellectual, personal or business reasons [42, 107]. Explanations of motivations for participation range from 1) the personal, including intrinsic motivations, altruism, future rewards and development of individual skills, to 2) the public, including the innovation and creation of public goods under the collective action model [11, 18, 107]. Then, the source code is made publicly available through the Internet. Interested people download and use the code, report bugs,



(a) Project–Developer cluster (bipartite graph)



(b) Developer cluster (unipartite graph)

Figure 3.1. Modeling OSS as a social network. A cluster of 5 projects and 16 developers (Projects and developers are anonymized to preserve privacy)

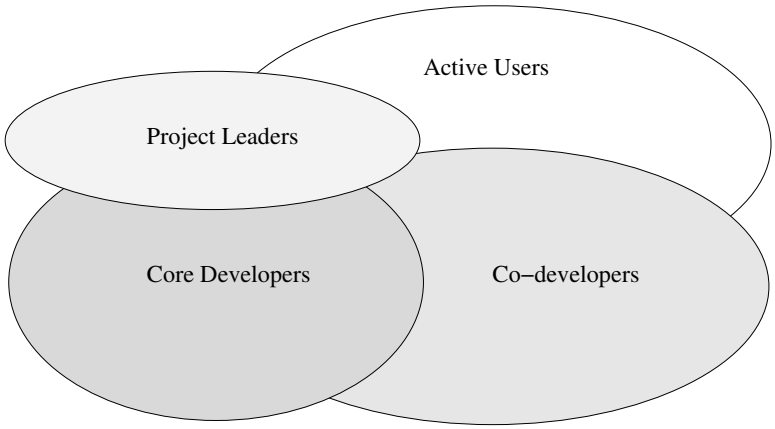


Figure 3.2. OSS development community classification

rewrite or modify code, suggest new features, and submit patches. Adding new code is controlled by a small group of developers. Thus, we can define member roles according to their contributions to the project. According to Xu [121], OSS members can be classified into either the user group or the developer group. The user group includes passive users and active users. *Passive users* have no direct contribution other than forming a larger user base. They just download code and use it for their needs. *Active users* discover and report bugs, suggest new features, and exchange other information by posting messages to forums or mailing lists. The developer group can be further categorized into peripheral developers, central developers, core developers and project leaders. *Peripheral developers* irregularly fix bugs, add features, provide support, write documents, and exchange other information. *Central developers* regularly fix bugs, add features, submit patches, provide support, write documents and exchange other information. *Core developers* extensively contribute to projects, manage CVS releases and coordinate peripheral developers and central developers. *Project leaders* guide the vision and direction of a project.

In this chapter, we assume that the OSS development community includes all of the above members except passive users, because passive users do not make direct software development contributions and thus are not considered to be part of the development community. Thus, as shown in Figure 3.2, our OSS development community includes the following groups:

1. *Project leaders* who are also called project administrators,
2. *Core developers* who regularly contribute on projects and manage CVS releases,
3. *Co-developers* including both peripheral developers and central developers, and
4. *Active users* who have some contributions except modifying code.

Because an individual may participate in multiple projects, that person can belong to different groups in the development community (overlap in Figure 3.2). Our definition of the OSS development community is considerably larger than the traditional closed-source development team which approximately coincides with our project leader and core developer groups. Our co-developer and active user groups are typically not part of the development communities for traditional closed source software, but are what has been identified as end-user contribution to the product innovation process [107].

3.4 Data Collection and Extraction

One challenge in studying OSS development is to collect and extract data. Data collection has proved to be tedious and time consuming [65]. Many difficulties exist in collecting, cleaning, screening and interpreting data [63]. In several

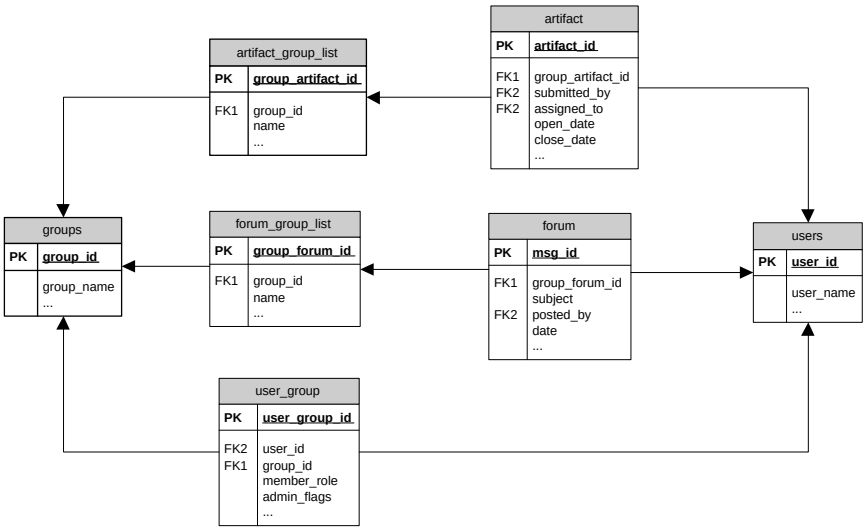


Figure 3.3. A subset of SourceForge database schema

other studies [40, 72, 119], web-bots were used to retrieve and extract data from web pages at SourceForge.net. This process often takes days, and frequently is plagued by incomplete and missing data. In this section, we discuss a much improved data collection and extraction process used in mining the SourceForge data.

We extracted data from a 2003 data dump obtained from SourceForge. The data dump is derived from the "back-tier relational database used to drive the SourceForge.net website. The data dump contains information about the community, projects, and developers. We examined this data to characterize the entire SourceForge community, across multiple numbers of projects, investigating behaviors and mechanisms at the project and developer levels.

In the SourceForge data dump, information about the roles of developers on each project is distributed over seven tables. Two roles, the project leader and core developer roles, are explicitly defined and stored in a single table. The other

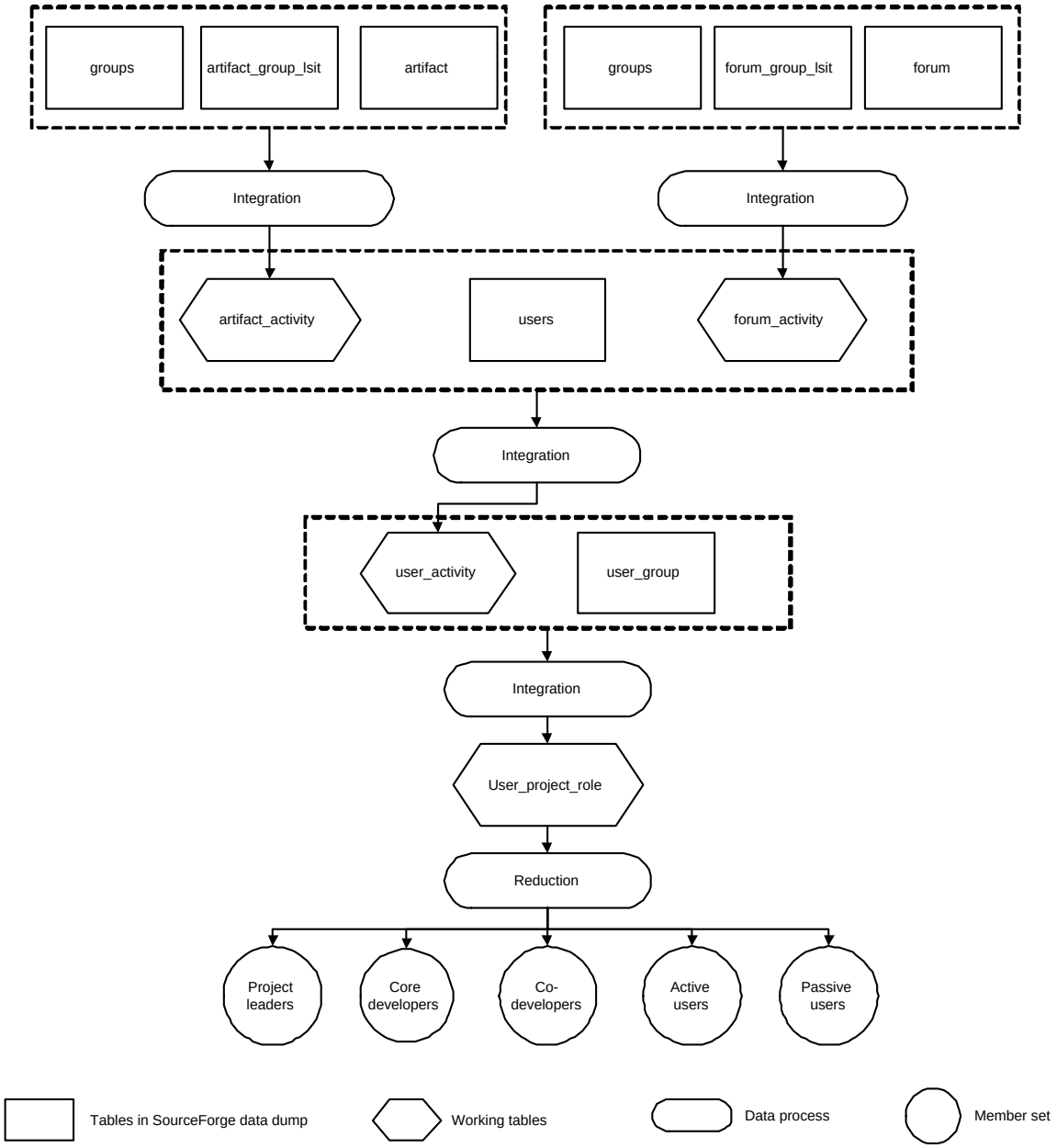


Figure 3.4: Data collection and extraction process

two roles, co-developers and active users, must be inferred and extracted from project activity data, such as bug reports, patch submissions, forum discussions, etc. The seven tables and their relationships are shown in Figure 3.3. Table *groups* is the list of all projects. Tables *artifact_group_list* and *artifact* contain members' activities such as bug tracking, patch submission, feature requests, document writing, etc. Tables *forum_group_list* and *forum* reflect members' participation in open discussion forums. Table *users* contains all users' information including their identification numbers, the date they joined the community, etc. Table *user_group* contains the relationships between projects and project leaders as well as core developers.

By processing the above tables, we can identify members and their participation activities for each project. The data extraction process is shown in Figure 3.4. We used a three-step data integration and data reduction process on the data. Data integration combines data from multiple sources into a coherent store. In the first step, we integrate *artifact* and *forum* separately to create two tables – *artifact_activity* and *forum_activity* to contain each member's activities in *artifacts* and *forums*. Because some attributes are different in *artifacts* and *forums*, we combine *artifact_activity* and *forum_activity* with *users* to get all members' activities for all projects, which is then recorded in *user_activity*. By integrating this table with *user_group*, we can identify each member's role in a project. We put this information into a table called *user_project_role*; lastly, data reduction is used to reduce the huge data set to a smaller representative subset according to members' role in a project.

3.5 Data Analysis

3.5.1 Analysis of OSS Member Distribution

As described in Section 3.3, we classified member roles as follows: project leaders are administrators in each project; core developers are developers listed in each project at SourceForge.net; co-developers are people who are assigned to tasks such as bug fixing and document writing, but are not listed as project leaders and core developers; active users are those who submit requests and post messages, but are not included in the project leaders, core developers and co-developers groups; passive users are obtained by excluding all developers from all users. Because a person can have different roles in different projects, we put every person into his/her highest ranked group. For example, if a person is listed as a project leader in one project and a core developer in another project, that person will be counted in project leader group. Figure 3.5 shows the distribution of member roles in the entire SourceForge population. About 65% of the SourceForge population are passive users who make no observable contributions to the development of projects. Among the development community, which comprise 35% of the entire population, there are 28% project leaders, 16% core developers, 34% co-developers and 22% active users. We observe that co-developers have almost the same percentage as the sum of project leaders and core developers. This is because a large portion of projects on SourceForge are small and most developers on them are also initiators of projects.

TABLE 3.1

DEVELOPERS DISTRIBUTION AMONG DIFFERENT SIZED PROJECTS

Community Size	**Project Number**	**Project Leaders**	**Core Developers**	**Co-developers**	**Active Users**
≤ 88	64847	80329 (47.8%)	34659 (20.6%)	33275 (19.8%)	19941 (11.8%)
> 88 and ≤ 279	193	590 (2.1%)	1703 (5.7%)	17334 (60.3%)	9124 (31.7%)
> 279	70	798 (0.9%)	2576 (2.7%)	53030 (55.8%)	38593 (40.6%)

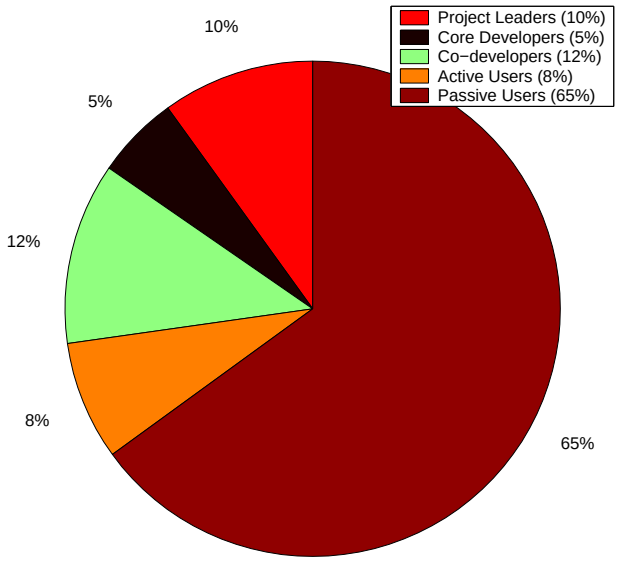


Figure 3.5. Distribution of SourceForge population

If we count the number of developers on a project, including the people from our four member roles, we get 261 different totals for all the projects ranging from projects with one developer to a project with 21710 developers on it. To further investigate the relationship between the project size and the developer community, we divide all projects into three categories, each representing approximately one-third of the range of project sizes: large projects, middle projects, and small projects; then analyzed the distribution of the four member roles for each project category. Large projects are those with more than 279 members; small projects contain less than 89 members, and middle projects have membership between 89 and 279 members. From a software engineering perspective, a small project team is usually composed of 1-7 developers; mid-sized teams range from 10-50 developers, and large projects teams have 50+ developers, so our project categorization appears to distort the notion of small, middle, and large project teams. However,

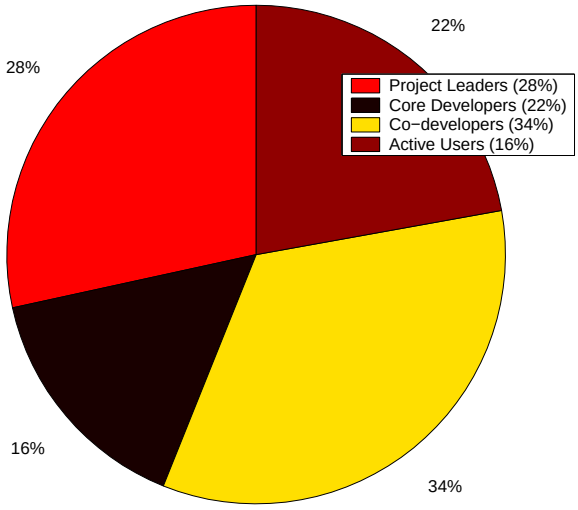


Figure 3.6. Distribution of SourceForge development community

because we take both co-developers and active users into account, our notion of a development community includes many more members than a traditional software engineering project team. Table 3.1 gives the development community distribution of these groups. In small projects, the main part of the community are project leaders (47.8%) and core developers (20.6%). As the size increases, the share of co-developers and active users increases. In large projects, project leaders and core developers comprise only 3.6% of the whole community, while co-developers and active users are 55.8% and 40.6%, separately. The fact that co-developers and active users comprise a large part in those more popular projects implies that they may play a crucial role in OSS projects.

3.5.2 OSS Network Topology

To understand how OSS development members (especially co-developers and active users) collaborate and their effect on the whole community, we divide the

OSS development community into four nested subsets. Each subset grows by including more individuals based on their roles in the SourceForge OSS community:

- Subset A = {project leaders};
- Subset B = {project leaders} ∪ {core developers};
- Subset C = {project leaders} ∪ {core developers} ∪ {co-developers} ;
- Subset D = {project leaders} ∪ {core developers} ∪ {co-developers} ∪ {active users} ;

We analyze the topological properties of these four subsets to help identify the different characteristics of each subset and determine the effect of different community members.

3.5.2.1 Degree Distribution

We hypothesize that the OSS community is a scale-free network because it may be growing and self-organizing due to sequential growth and preferential attachment. In the OSS community network, the number of community members and projects grow over time. With the evolution of projects, community members sequentially join projects. Furthermore, the OSS development community is highly decentralized. Developers freely participate on projects which attract them. Some projects are more popular than others, and those projects tend to attract more developers and users; thus in this network, there exists a preferential attachment of developers to projects.

To support our hypothesis, we compute the degree distributions of SourceForge project and developer networks, shown in Figure 3.7 and Figure 3.8. All left sub-

TABLE 3.2

REGRESSION PARAMETERS OF DEGREE DISTRIBUTIONS

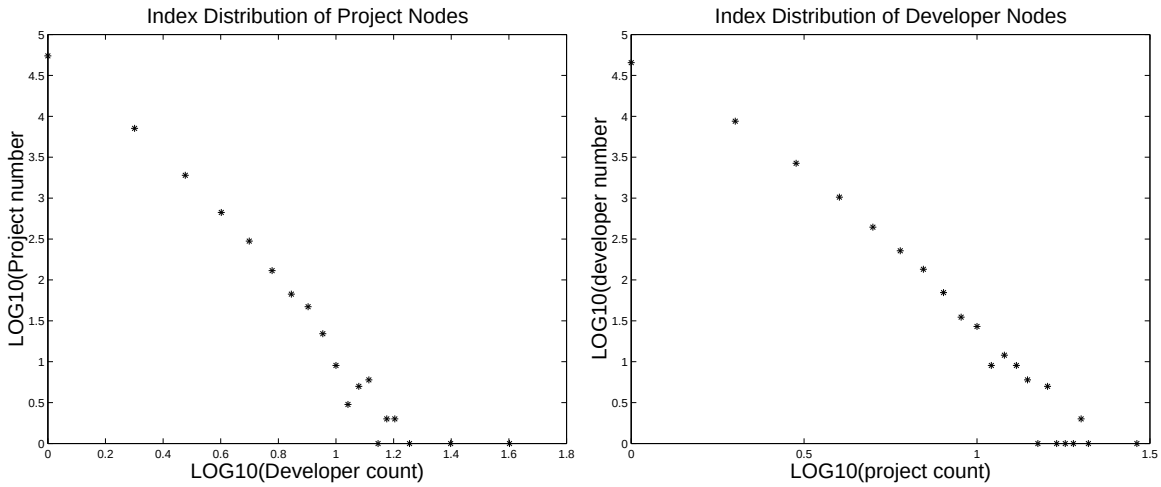
	Parameters	**Subset A**	**Subset B**	**Subset C**	**Subset D**
Project-side	R^2	0.9396	0.9704	0.6905	0.7221
	Slope	-3.5841	-2.6968	-1.3020	-1.2220
Developer-side	R^2	0.9870	0.9846	0.9469	0.9830
	Slope	-3.3747	-3.4676	-3.7793	-3.2743

graphs represent the distributions of the log-log transformation of project size frequencies. The right subgraphs display the distributions of the log-log transformation of the project membership of developers. All eight subgraphs show degree distributions which are highly skewed. For example, on the right subgraphs, a large number of members only participate on one project (45261 in subset A, 63103 in subset B, 105234 in subset C, 113999 in subset D). But some members join multiple projects. When comparing subset A to subset D, the largest number of projects a member joins increases from 29 to 95. These members are linchpin nodes in the community network, who link projects together into clusters. We can observe that all distributions display the power law relationship. Table 3.2 shows the linear regressions of all eight subgraphs. Such a power law relationship suggests that the SourceForge development network is a scale free network. In this network, a successful project can attract more developers, while many projects will stagnate after a short while. The number of members in a project may be an important positive feedback factor in determining the attractiveness of a project. We further use Pajek [91] to draw a cluster of the project network. As shown in Figure 3.9, this cluster includes a random sampling of 2495 out of 65110 total

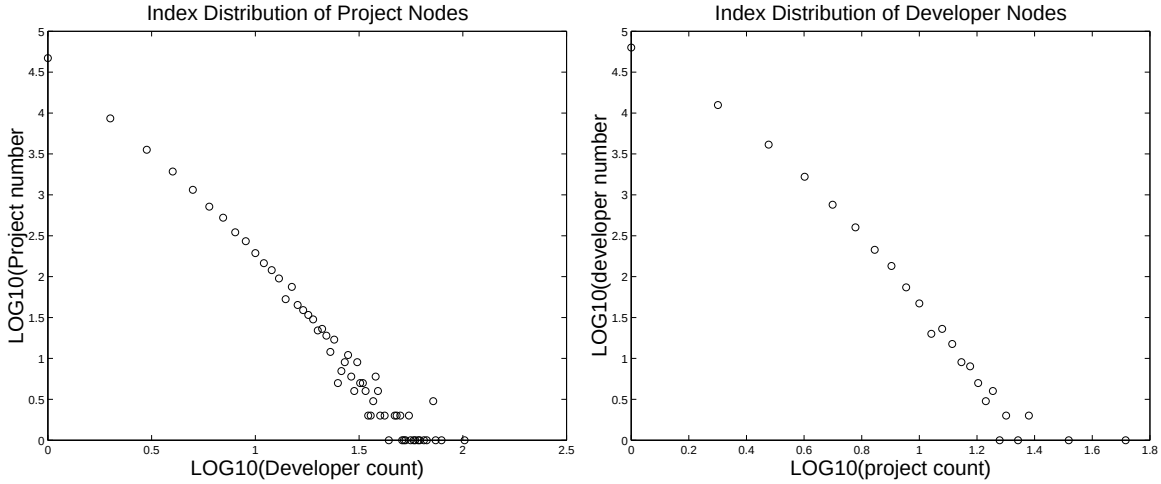
projects. The size of each node represents the logarithmic value of its developer count, and edges indicate common developers with other projects in the random sampling. While visually deceiving, the figure shows the project network has a large number of small nodes, i.e. projects with few developers, compared to a small number of large nodes, projects with many developers. This is an observed property of scale-free networks.

3.5.2.2 Diameter

Using Equation 3.2, we calculated the average shortest path length of the four community subsets. As shown in Table 3.3, if the community is composed of only project leaders, the average shortest path length computed using Equation 3.2 in this network is infinity. This reflects the fact that the project leader network is highly disconnected. Perhaps project leaders are too busy to join other projects? In subset B, which includes both project leaders and core developers, the average shortest path shrinks to a length of about 10 for a total of $83,118$ members; the introduction of core developers makes the social network much more connected creating a large component of $15,091$ members. Project leaders and core developers are important in OSS networks because there is more information exchange within them and they reinforce the cooperation among members in the same project group. Thus, they work as strong ties in OSS networks. Core developers are more willing (or able?) to participate on other projects. Co-developers and active users have a much greater likelihood to join multiple projects. They act as the weak ties in the development community, and the result is that the social network becomes even more connected. Subset C and subset D both have an average

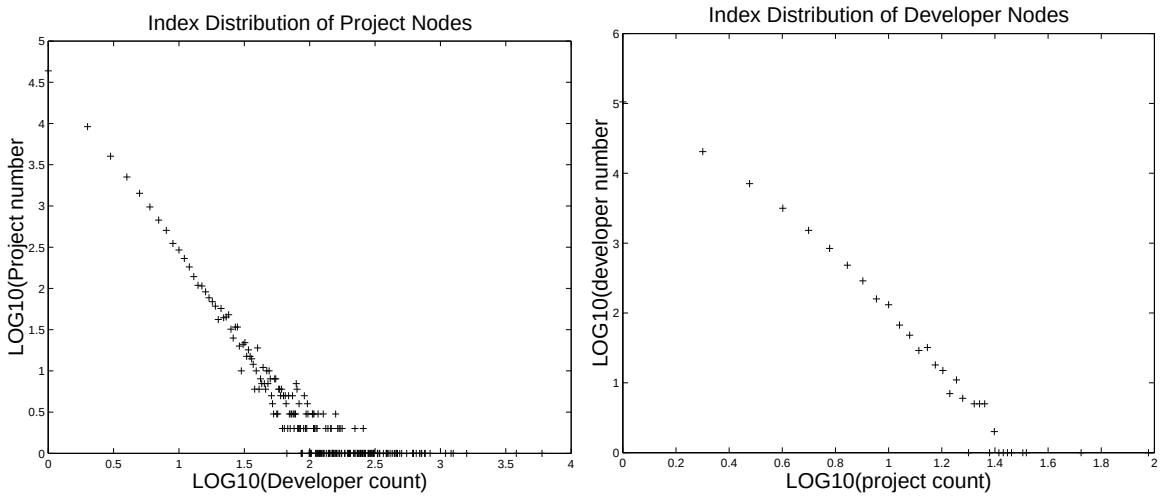


(a) Subset A

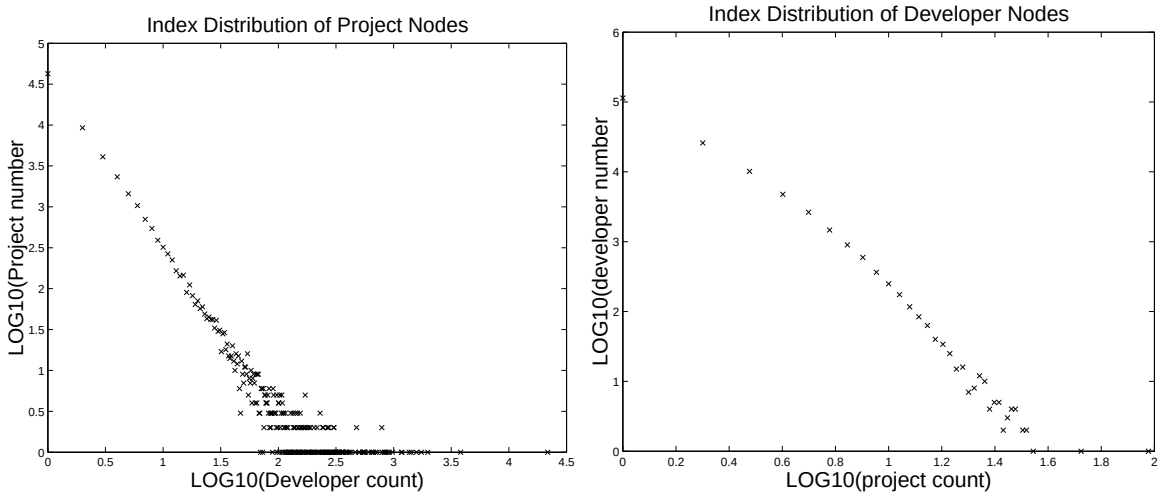


(b) Subset B

Figure 3.7: The SourceForge project and developer community scale free degree distributions



(a) Subset C



(b) Subset D

Figure 3.8: The SourceForge project and developer community scale free degree distributions (cont.)

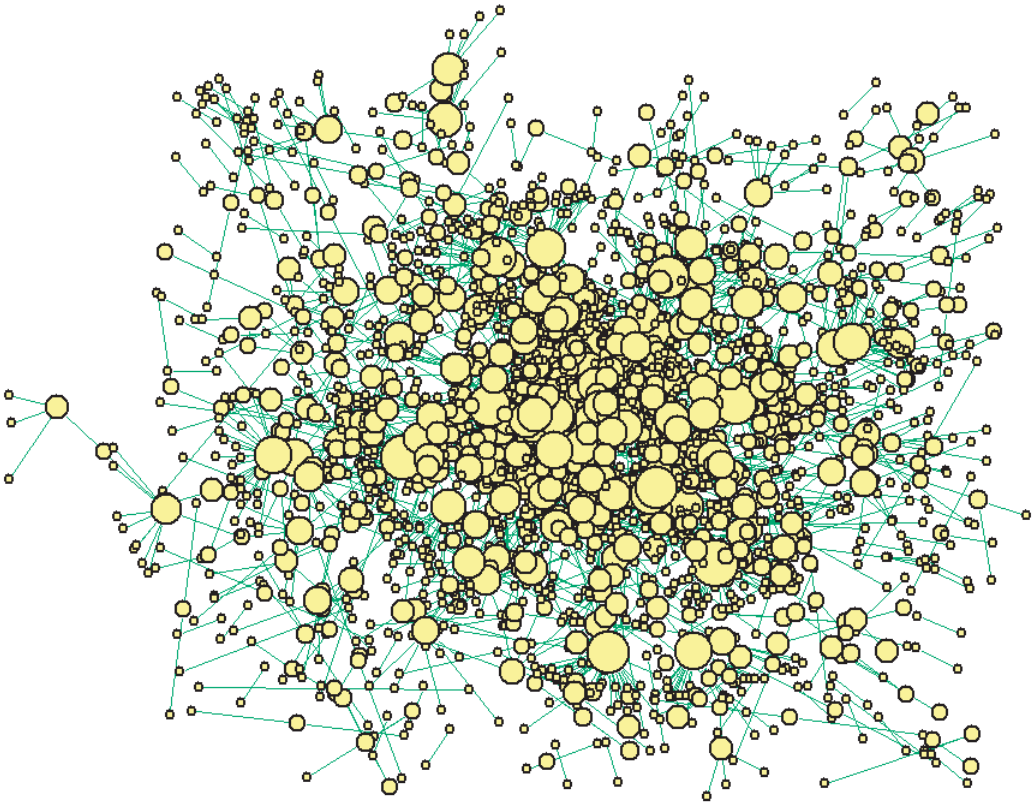


Figure 3.9. A scale-free project network (unipartite graph). This is an actual graph of 2495 randomly selected projects. Node size is logarithmically proportional to the number of developers, and edges represent a common developer between projects.

TABLE 3.3

THE PROPERTIES OF THE DEVELOPMENT COMMUNITY

Property	**Subset A**	**Subset B**	**Subset C**	**Subset D**
Size	58651	83118	139570	161691
$\mathbf{z_1}$	1	6	508	3241
$\mathbf{z_2}$	1	17	13398	31998
Diameter	Inf.	10.2	2.7	2.7
Clustering Coefficient	0.8406	0.8078	0.8867	0.8297
Largest Project Cluster	737	15091	30794	40175
2^{nd} Largest Project Cluster	197	34	20	20
# of Project Clusters	43826	34280	27983	21659

shortest path of about 3; that means, on average, it takes only 3 links for any member in the SourceForge development community to reach another member in the community. With the participation of co-developers and active users, the degree of separation between any two members is significantly decreased. By understanding this fact, certain phenomenon can be understood in terms of the short distance between members in the OSS community; information such as ideas and discussions can spread fast in the OSS development community because the information only has to travel a few links to reach anybody in the network. Likewise, because many of the communication forums for projects are broadcast mediums, information spread by members reach many other members with just a single message post. This property is described in the following subsection.

3.5.2.3 Clusters – Sizes and Numbers

Two projects are linked if they share a community member. All linked projects form a project cluster (See Figure 3.1 for an example). Each project cluster has a corresponding developer cluster, so we limit our discussion in this section to project clusters. All four subsets of the SourceForge development community contain many separated clusters (shown in Table 3.3). Also in Table 3.3, we see that the largest cluster is much bigger than the second largest cluster and grows as we move from subset A to subset D. However, the second largest cluster decreases with the increase of the community size. The reason is that some clusters are linked to become part of the largest cluster. The larger a cluster is, the more likely it will attach to the largest cluster as the community size increases.

This type of cluster analysis may be used to identify groups of related projects or developers with similar interests. This discovery may reflect that some projects are more similar than others. They may have some common characteristics. This may be applied to project management. For example, by identifying similar projects, we can study the life cycle of an old project to predict the future of a young project. Alternatively, recommender systems such as those found at online shopping sites may be developed to assist users and developers looking for software or projects to participate on.

3.5.2.4 Clustering Coefficient

We use Equation 3.3 to get the approximate clustering coefficients of the four subsets of the SourceForge community. The clustering coefficient tells us how many of a member's collaborators are collaborators with each other. As shown in

Table 3.3, the clustering coefficients of all four subsets are above 0.8. The high clustering coefficients are not surprising because members are fully connected in each project.

3.5.3 Discussion

We found a small world phenomenon, the small diameter and the high clustering coefficient, in the SourceForge development community. The small distance results from the fact that a member may participate in multiple projects. In this way, the member connects separate clusters and creates a path among members in those clusters. Moreover, a large percentage of members participate on one project (70.5% in subset D). This fact explains the observed high clustering coefficient; though, the high clustering coefficient may also be an artifact of how we construct our social network with all developers on the same project connected to each other. Furthermore, the power law distribution found in the SourceForge OSS network supports the claim that it is a scale-free network. OSS projects are often developed by collaborating volunteers, who sequentially join projects based on their interests.

In other studies [40, 74], the power law distribution and the small world phenomenon in the SourceForge project leaders and core developers community (classified as subset B in this chapter) was observed. In this chapter, we observe that with the participation of co-developers and active users, while the cluster is increasing, the diameter is much smaller (only about 3 links between pairs of community members). Co-developers and active users are weak ties in the whole network. Weak ties are often more important in spreading information or resources because

they tend to serve as bridges between otherwise disconnected social groups [43]. This fact supports the assertion that co-developers and active users play a crucial role in connecting the SourceForge development community. Their existence can make communication flow faster throughout the whole OSS community. This fast communication may be an example of a self-organizing, optimal reallocation of resources. For example, our SourceForge data contains 676 text editor projects. Of all developers on those projects, about 50% of developers are on the top 6 largest projects. One reason for this might be the short communication path in the development community through which people can find projects which attract them (e.g., are better organized, have more compatible goals, or are making better progress). The feature of fast communication in OSS development community may be a factor of its success because closed-source software development does not typically have co-developers, and active users may not be in close contact with developers.

Examination of project evolution shows that large and medium projects are robust. Because these large and medium projects often act as hubs in the OSS network, the robustness of hubs translates into a robustness of the OSS network. One reason for the robustness results from the open decentralized nature of OSS development: unlike commercial software companies which are developed in a centralized way, OSS projects are developed in an open decentralized manner. If a key participant on a project leaves, the open decentralized development process can draw on a larger community of active users and co-developers to find a person to step into the vacant role.

OSS projects have development life cycles. The development of OSS projects is often driven by innovation [107]. A new project often involves new ideas and

cutting-edge technology. It may attract both existing developers and new developers. The introduction of such a new project will change the network topology. On the other hand, some old projects may become mature or unsuccessful. Developers in these projects may leave them to join some more attractive ones.

3.6 Conclusions

In this chapter, we apply social network analysis to study the Open Source Software development community at SourceForge. Based on a SourceForge 2003 data dump, we perform quantitative analysis on the SourceForge OSS developer and project networks. Using statistical analysis, we found that different sized projects have different member distributions. Large projects consist mainly of co-developers and active users, while project leaders and core-developers are main parts of small projects. Furthermore, we conduct topological analysis on four subsets of the OSS development community. Properties of the community network show that the SourceForge OSS development community is a self-organizing system which obeys scale-free behaviors. Moreover, small-world phenomenon, the small average distance and the high clustering coefficient, exists in the community. We identify the important effect of co-developers and active users because their addition can turn the OSS community into a fast communication network. Our research provides useful information of the underlying structure and evolution of the OSS community, and the diffusion of information between projects.

CHAPTER 4

THE OPEN SOURCE SOFTWARE COMMUNITY STRUCTURE

4.1 Introduction

Social network analysis has recently been used to study the OSS development community [15, 70, 74, 117, 119, 120]. Among the essential static and dynamic properties of social networks, *community structure* can help us understand the similarity and difference among groups in a network[1]. In this chapter, we identify the community structure for the SourceForge project network. Furthermore, we examine software categories to explore possible reasons for the formation of the community structure. Our research provides useful information to study the interaction between projects, and the communication and information flow in OSS virtual community.

One network property that has attracted increasing interests from the scientific network analysis is *community structure*. In some networks, some nodes are grouped together by a high density of edges, while there are few edges between those groups. These tightly-connected groups are called *communities* into a network [85]. (Note: these communities of projects are not to be confused with the project communities composed of developers and active users that were discussed earlier.) This phenomenon is also called *clustering*.

[1]This chapter is based on paper "The Open Source Software Community Structure", NAACSOS 2005, Notre Dame, IN, June 2005 [114]

Detecting communities in networks can reflect some important characteristics of networks. Nodes in the same community are more likely to have some common features. For example, community structure identification can be used in the analysis of World Wide Web to find topically related web sites; in a biological system, communities could indicate highly related genes. Communities in a social network reflect friendship and collaboration among people. Nodes between communities play a key role in connecting those communities in a network. These nodes are usually called *hubs*. If one or several of these nodes are removed, the whole network becomes disconnected. Through the identification of community structure, we can also identify hubs of the whole network. Detectinging communities among a network helps us understand the backbone of relationships among nodes and the paths for the signal propagation among them.

Community structure has been found in many networks. For example, Girvan et al[41] study communities in the collaboration network of scientists at the Santa Fe Institute. The collaboration network consists of 271 scientists in residence at the Santa Fe Institute during the calendar year 1999 or 2000. An edge lies between a pair of scientists if they have coauthorship for articles during that time. Communities are found to exist among scientists grouped together by similar research topics or methodologies. Wilkinson et al [113] study a network of gene symbols created by co-occurrences from Medline article abstracts. In this network, each node represents a gene. Two genes are connected by an edge if they co-occur in an article. Communities in this gene network identify functionally related genes. Tyler et al [103] apply community structure identification to an email corpus of nearly one million messages collected over a two-month span, where nodes are senders and recipients of email messages and an edge indicates a direct email

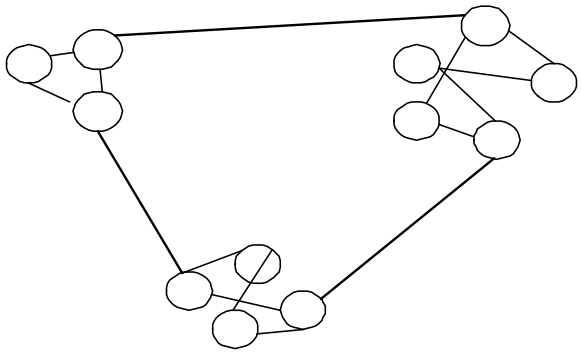


Figure 4.1. An example of a project network with community structure

message between nodes. Communities in this email network represent actual social relationships. Massen et al [77] conduct community structure research on energy landscapes. They construct energy landscapes as a network of minuma linked by transition states. In this network, nodes grouped in a community correspond to minima with similar potential energies.

Open Source Software (OSS) development community can be modeled as a decentralized, unorganized but self-organizing social network, which can be further divided into the OSS developer network and the OSS project network. In this chapter, we focus on the OSS project network because common experience suggests that some OSS projects may be more closely related to each other than other projects. The OSS project network may possess community structure. In order to differentiate from communities in previous chapters, we define a community in OSS project network as a collection of projects such that each member project has more edges within the community than outside of the community. A project community contains highly related projects.

Detecting communities in the OSS network can have profound effects: first, we

can find projects which might have related subjects, similar programming environment, or common developers. By identifying these communities, we can study their interactions during their growth. Such research could provide insight into the management, organization, and interests of a specific portion of developers; second, we can get information about the communication path and knowledge flow within or between communities. Such information can help us adjust and improve the robustness of communications in OSS development, i.e. by identifying the key(hub) nodes, improvements can be made to avoid single points of failure. Figure 4.1 is an example of a project network with community structure.

This chapter is an extension of previous social network analysis [74, 117, 119, 120] on SourceForge [59]. In this chapter, we explore the community structure existing in the OSS project network. This network is constructed based on data collected and extracted from a SourceForge 2003 data dump. We apply a community structure identification algorithm on the largest component of the project network. Moreover, we give possible explanations for such communities by studying mixing patterns of SourceForge projects.

This chapter is organized as follows. Section 2 describes two methods to detect community structures. Hierarchical clustering is a traditional method commonly used in network studies to find communities. Girvan and Newman proposed a new algorithm to improve the quality of community detection based on a concept called *edge betweenness*. Section 3 gives an community structure detection algorithm which finds communities in a network by a greedy way. This algorithm modifies the Girvan-Newman method decreasing its time complexity. The next section conducts community structure detection on the OSS network. A possible explanation of community structure is given in Section 5. Lastly, conclusions are

given.

4.2 Community Structure Detection Methods

Community structure detection in a network is nontrivial, especially when the number of vertices is very large. A traditional technique used in the social network study is called *hierarchical clustering* [99, 108]. This method has been incorporated into some network analysis softwares such as UCInet and Pajek. Recently, Girvan and Newman designed a new algorithm to identify communities based on the importance of edges in a network. This method has been applied in many research areas and becomes more popular. We discuss these two methods in detail in this section.

4.2.1 Hierarchical Clustering

Hierarchical clustering finds relatively homogeneous communities based on the measured strength for each pair of vertices in the network. Such strength can be the shortest path between vertices, or their inverse, "maxflow" methods (counts of numbers of vertex- or edge-independent paths) between vertices or weighted counts of total number of paths between vertices.

The hierarchical clustering process starts with each node in a separate cluster. Then, clusters are combined sequentially in decreasing strength of pairs of vertices. The number of clusters is reduced at each step until only one cluster is left. When there are N nodes, this process involves N-1 clustering steps. One can stop at any step and check the communities gotten so far. The whole process proceeds as follows:

1. assign each node to a separate cluster. If you have N nodes, you now have

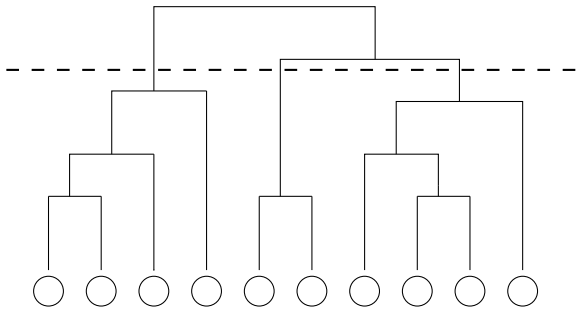


Figure 4.2. An example of a dendrogram by hierarchical clustering

N clusters, each containing one node;

2. compute the strength between pairs of clusters;

3. join a pair of clusters with the strongest strength;

4. Repeat steps 2 and 3 until all items are grouped into a single cluster of size N.

This hierarchical clustering process can be represented as a tree data structure, or dendrogram, where each step in the clustering process is illustrated by a join of the tree. Figure 4.2 is the example of the dendrogram by using hierarchical clustering process.

The hierarchical clustering method is widely used in the network analysis. However, this method has some problems. In some cases, hierarchical clustering fails to find community structure. For example, leaf nodes (nodes having one edge) are frequently treated as separate communities on their own in the network. It seems intuitive that such leaf nodes should be grouped into the community its edge leads to. To improve the quality of identifying community structures, Girvan and Newmman present a more accurate method.

4.2.2 Girvan - Newman (GN) Method

Detecting network communities is a difficult problem due to the high complexity of graph computation. Many algorithms have been designed to approximate the computation. A recent algorithm designed by Girvan and Newman [41, 85] is based on edge betweenness. This method focuses on edges which are most important to the connection of the whole network. By progressively removing edges with the highest betweenness, groups can be separated from the whole network and reflect the underlying structure.

The idea of edge betweenness is derived from node betweenness. In a network, the betweenness of a node is a measure of the centrality, which determines the relative importance of a vertex within a network. The value of betweenness of a node is related to the pattern of connection of the node. It measures the control of the node over the network.

Freeman [30] gives a definition of betweenness as follows: for a graph $G(V, E)$ where V is the set of vertices and E is the set of edges, the betweenness $C_B(V)$ for vertex v is:

$$C_B(v) = \sum_{i \neq v \neq j \in V} \frac{N_{ij}(v)}{N_{ij}} \tag{4.1}$$

where N_{ij} is the number of geodesic shortest paths from i to j, and $N_{ij}(v)$ is the number of geodesic shortest paths from i to j that pass through a vertex v. Thus, vertices that occur on many geodesic shortest paths between other vertices have higher betweenness than those that do not. The removal of the node with the highest betweenness will cause the increase of geodesic distance between the largest numbers of other vertex pairs.

Givan and Newman extend the concept of betweenness to edges in a network. The edge betweenness in the GN algorithm represents the number of geodesic

shortest paths between pairs of vertices that run along that edge. In a network, inter-community edges have higher edge betweenness than inner-community edges, because the number of inter-community edges are fewer than the number of inner-community edges, and more shortest paths go through inter-community edges to connect separate communities. Thus, eliminating edges with high edge betweenness will split a network into different groups.

The GN method proceeds as follows:

1. calculate the edge betweenness of every edge in the network;
2. remove the edge with the highest betweenness. If there are more than one edge with the same highest betweenness, randomly pick one;
3. repeat step 1 and step 2 until no edge remains.

The above algorithm divides communities step by step until each node is in a separate group. In real applications, we don't know the community structure in advance. We must know when to stop the procedure to get a good community structure. To solve this problem, Newman proposed a measure to calculate the quality of division [85], which is called *modularity*. For a network of k communities, we can define a $k \times k$ symmetric matrix where each element e_{ij} represents the fraction of all edges connecting community i to community j. Modularity is defined as:

$$Q = \sum_i e_{ii} - \sum_{ijk} e_{ij} e_{ki} = Tre - ||e^2|| \tag{4.2}$$

Where $a_i = \sum_j e_{ij}$, which represents the fraction of edges connecting to community i. $Tre = \sum_i e_{ii}$ is the fraction of edges that connect vertices in the same community. $||X||$ represents the sum of all elements of matrix X. Q measures the fraction of within-community edges minus the expectation of the same quantity in

a random network with the same community divisions. If the network is a random network, we get $Q = 0$. The maximum value of Q is 1. During the execution of GN algorithm, we can compute Q at each step. The peak value of Q means the current community structure is the best one we can get.

The GN algorithm runs in $O(m^2n)$ time on a network with m edges and n vertices. The calculation of edge betweenness takes time $O(MN)$. The recalculation of all edges gives a worst case running time of $O(M^2N)$. On a sparse graph, where $m \approx n$, it runs in $O(n^3)$. This algorithm is hard to apply on a network with a large number of nodes and edges.

4.3 A Faster Algorithm

The high complexity of GN algorithm makes computation for a large network impractical. A faster algorithm is proposed to detect network communities by a greedy algorithm [84]. Unlike the previous divisive algorithm, this algorithm works in an agglomerative way: at the starting point, each node is a group; at each step, two communities are joined which results in the best grouping. This algorithm runs in time $O((m+n)n)$ on a random network, or $O(n^2)$ on a sparse graph, where n is the number of nodes and m is the number of edges in the network. An improved method is presented by Clauset et al. al [19] which performs the same optimization, but implements a more sophisticated data structure. This algorithm runs more quickly, in time $O(mdlog(n))$, where d is the depth of the "dendrogram". For networks which have a hierarchical structure with communities at many scales, which has $d\,log(n)$, this algorithm can run in an approximately linear time, $O(n(logn)^2)$.

We use the algorithm proposed by Clauset [19] to analyze the community

structure of the OSS project network. This method is based on the greedy optimization of the quantity known as *modularity*, which measures if the division of community is meaningful, that is, after the division, there should be many edges within communities and a few edges between them. The modularity Q can be computed using equations 4.3 and 4.4.

$$Q = \sum_i (e_{ii} - {a_i}^2) \tag{4.3}$$

$$a_i = \frac{k_i}{\sum e_{ij}} \tag{4.4}$$

e_{ij} is the fraction of edges which connect nodes in group i to group j versus the total edges in the network, a_i is the fraction of edges which connect to nodes in group i versus the total edges in the network, and k_i is the number of edges which connect to group i. The method uses a greedy optimization to repeatedly group communities whose amalgamation results in the largest increase in Q. The best community structure is given when Q is the largest.

In GN algorithm, the adjacency matrix of modularity Q is maintained and the change of modularity ΔQ_{ij} is calculated at each joining. Since joining two communities with no edges between them results in no increase of Q, an optimization in Clauset algorithm is to maintain and update a matrix of ΔQ_{ij} instead of Q. Furthermore, this matrix can be represented and the largest ΔQ_{ij} can be kept with efficient data structures. These data structures designed to implement the algorithm include:

- A matrix contains ΔQ_{ij} for each pair of group i and group j. Each row of the matrix is stored as a balanced binary tree for $O(logn)$ search and insertion time and as a max-heap for constant time to find the largest element.

- A max heap contains the largest element of each row.
- An array contains a_i.

At the beginning, each vertex is a community. Thus, e_{ij} in matrix ΔQ_{ij} equal to $1/2m$ if i and j are connected and 0 otherwise, and $a_i = k_i/2m$. The algorithm proceeds as follows:

1. The initial values of ΔQ_{ij} and a_i are calculated by 4.5 and 4.6 and stored into data structures.

$$\Delta Q_{ij} = \begin{cases} \frac{1}{\sum e_{ij}} - \frac{k_i k_j}{\sum {e_{ij}}^2} & \text{if } j,\ j \text{ are connected,} \\ 0 & \text{otherwise} \end{cases} \tag{4.5}$$

$$a_i = k_i / \sum e_{ij} \tag{4.6}$$

2. The largest value of ΔQ is selected from the max-heap and the corresponding communities are grouped.

3. The ΔQ matrix and a_i vector are updated by Equation 4.7 and Equation 4.8 and step 2 is repeated until there is only 1 group. In matrix ΔQ, only a few of elements need to be updated. If community i and community j are joined, the newly formed community will be labeled as community j. Then, the jth row and column in matrix ΔQ must be updated and the ith row and column must be removed. All communities connected to the previous community i and community j are also updated by following rules:

$$\Delta Q_{jk}{}' = \begin{cases} \Delta Q_{ik} + \Delta Q_{jk} & \text{if group } k \text{ is connected to both } i \text{ and } j, \\ \Delta Q_{ik} - 2a_j a_k & \text{if group } k \text{ is connected to } i, \text{ but not } j, \\ \Delta Q_{jk} - 2a_i a_k & \text{if group } k \text{ is connected to } j, \text{ but not } i \end{cases} \tag{4.7}$$

$$a_j{}' = a_j + a_i \tag{4.8}$$

If we represent degrees of i and j as $|i|$ and $|j|$, to update the jth row in ΔQ, if group k is connected to both i and j, we insert elements of the ith row into jth row and sum them up. Because rows are stored as balanced binary trees, each of $|i|$ insertions take $O(log|j|) \leq O(log(n))$ time. If group k is connected to either i or j, updating other elements of the jth row takes at most $(|i| + |j|)$ time. For kth row, updating one element takes $O(log|k|) \leq O(log(n))$ time. There are at most $|i| + |j|$ elements in kth row to be updated, which take $O((|i| + |j|)log(n))$ time. Updating max-heaps for each row takes $O(log|k|)$ time. The overall max-heap H takes $O((|i| + |j|)log(n))$ time to be updated. Change of a_j is trivial and costs constant time. Thus, each join takes $O((|i| + |j|)log(n))$ time. In the worst case where the degree of a community in the dendrogram with depth d is the sum of the degrees of all the vertices in the original network, since the total degree of all nodes is $2m$, the running time of this algorithm is $O(mdlog(n))$.

4.4 Community Structure of the OSS Project Network

We studied the OSS project network in SourceForge. In this network, two projects are connected if they have one or more common developers. We applied

the algorithm to the project network to detect communities. We examined the largest component of the project network in January 2003. We also exclude project SourceForge [59] because it has links to over $10,000$ other projects. Thus, the project network we studied consists of $27,834$ nodes and $173,644$ edges.

Figure 4.3 gives the value of modularity Q over all rounds. The highest value of the modularity is $Q = 0.2227$, which occurs when there are 611 groups. It contains several large communities and many small communities. The largest group consists of 3467 projects and there are many groups of size less than 10. Figure 4.4 shows the distribution of the group size. We observe that the group size follows a power law distribution. The slope of its log transformation is 0.9432. Table 4.1 gives the 10 largest groups in this network. These 10 largest groups include 68.4% of the whole SourceForge projects, while the rest 601 groups only take about 30% of all projects.

Our results also provide information about communications and knowledge flow about the project network. The important communication paths are those connections between communities. According to their interests, some developers participated on some projects which target to different topics and belong to different communities. These developers become key to transfering information between two seemingly unrelated project groups.

4.5 OSS Assortative Mixing

Several explanations exist for the formation of the community structure. Communities can be the result of the mutual acquaintance mechanism [66], that is, two nodes with a common neighbor are more likely to link to each other than those without common neighbors. This mechanism has been proved to work in social

TABLE 4.1

THE 10 LARGEST COMMUNITIES IN THE PROJECT NETWORK

Community size	**Number of such communities**
3467	1
3404	1
3023	1
2931	1
2511	1
2198	1
2096	1
1898	1
911	1
576	1

networks [66]. However, in many social networks, communities cannot be simply explained by mutual acquaintances. Communities may be present for nodes with the same attributes. Thus, homophily plays an important role for the community structure. For example, when we study communities of people, we may consider their ages, languages, and nationalities. Assortative mixing [85] is a way to study the preferential association of nodes to others that are like them in some way.

The level of assortative mixing can be measured by an *assortative coefficient*, which is defined as:

$$r = \frac{\sum_i e_{ii} - \sum_i a_i b_i}{1 - \sum_i a_i b_i} \tag{4.9}$$

where e_{ij} is the fraction of edges in a network which connect nodes of type i to nodes of type j, a_i and b_j are the fraction of edges of each type which are attached

to nodes of type i. They satisfy the following rules:

$$\sum_{ij} e_{ij} = 1 \tag{4.10}$$

$$\sum_{j} e_{ij} = a_i \tag{4.11}$$

$$\sum_{i} e_{ij} = b_j \tag{4.12}$$

SourceForge provides several categories for hosted projects [101]. A project can be classified to its topic, operating system, user interface, development status, intended audience, and programming language. Each category contains hierarchical subcategories. In our study, we only consider the first level subcategories.

In SourceForge, a project might be listed into multiple subcategories in those categories. For such kind of projects, we treat them as a separate project in each subcategory. This creates multiple edges with different kinds for a pair of projects.

We calculated the assortative mixing coefficient for each category based on equation (3). Table 2 shows SourceForge categories, their corresponding first-level subcategories, and the assortative coefficient for each category. There are about 30% projects on SourceForge which are not assigned to a category. We exclude these projects when we calculate assortative coefficients. Assortative coefficients for all categories are positive, which means projects in the same categories prefer to associate with each other. Among those categories, programming language, operating system, and topic have strong assortative mixing effects. Because our project network is linked by people who join multiple projects, those strong assortative mixing coefficients can be explained by the fact that when an existing

developer makes a decision to join a project, he/she tends to join projects which have the same programming language, operating system and topic as his/her current projects.

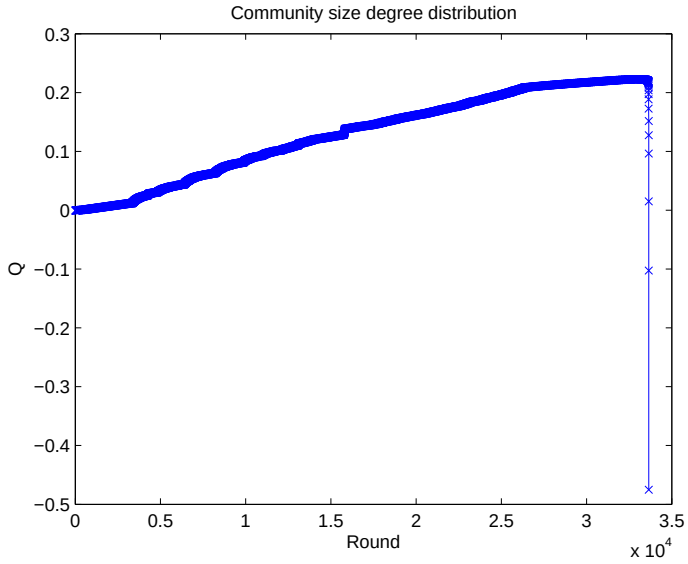


Figure 4.3: Values of modularity Q over all rounds. Q increases first, and gets to the maximum when the best grouping is found, then it decreases.

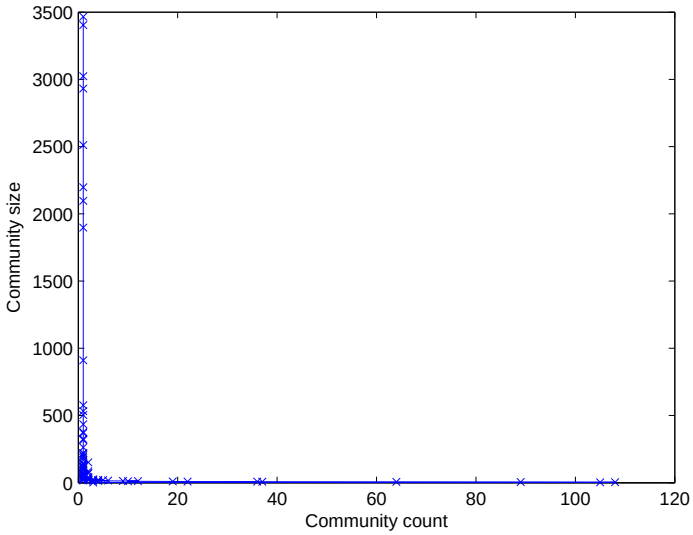


Figure 4.4: Community degree distribution of the largest component of the project Network. The group distribution follows the power law, where there are small numbers of large groups and large numbers of small groups.

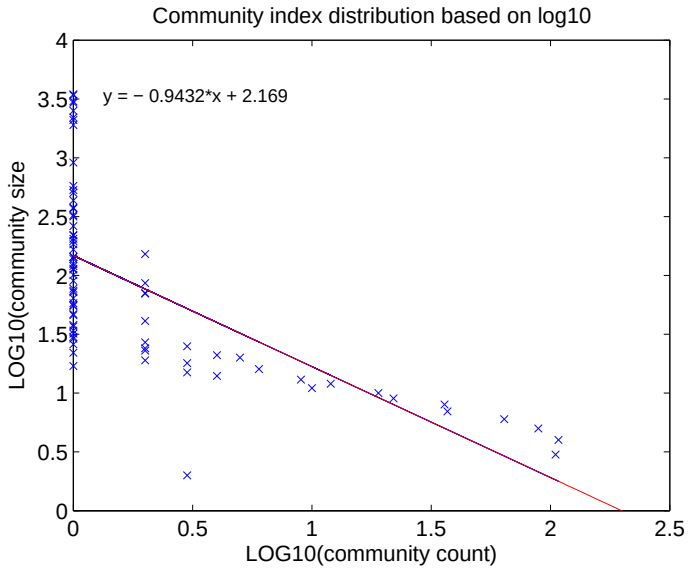


Figure 4.5: Community degree LOG distribution of the largest component of the project network, which can be fitted as a decreasing line function.

TABLE 4.2

SOURCEFORGE PROJECTS CATEGORIES AND SUBCATEGORIES

Category	**First-level Subcategories and Their Percentage**	**r**
Topic	Communications(7.7%), security(1.3%), development(8.1%), desktop(1.8%), editors(1.3%), database(3.0%), education(1.5%),	0.1009
	games(7.3%), internet(12.0%), scientific(3.9%), multimedia(5.9%), office(2.3%), religion(0.1%), system(9.7%), printing(0.2%), terminals(0.3%), other(1.2%)r, unknown(32.2%)	
Operating System	Posix(25.2%), Microsoft(14.0%), os2(0.1%), macos(1.9%), beos(0.4%), independent(14.7%), pdasystems(0.5%), other(0.8%), unknown(33.7%)	0.1078
User Interface	Console(10.6%), x11(10.2%), win32(9.3%), web(11.7%), daemon(3.5%), cocoa(0.7%), handhelds(0.4%), other(4.7%), unknown(36.3%)	0.0893
Development Status	Planning(12.8%), prealpha(8.6%), alpha(7.7%), beta(9.1%), production(7.0%), mature(7.3%), inactive(0.1%), unknown(31.7%)	0.0553
Intended Audience	End users(21.9%), developers(24.9%), system administrators(10.7%), customer service(0.2%), education(0.8%), financial insurance(0.09%), health care industry(0.07%), information technology(1.0%), legal industry(0.4%), manufacturing(0.9%), religion(0.4%), science research(0.7%), telecommunications(0.2%), other(5.4%), unknown(32.6%)	0.0449
Programming Language	C(11.1%), c++(10.6%), java(8.4%), php(6.4%), perl(4.7%), python(2.4%), visual basic(1.3%), other(0.6%), Unknown(32.9%)	0.1541

4.6 Conclusions

In this chapter, we conduct the identification of the community structure for the SourceForge project network. This network is constructed by common developers among Sourceforge projects. We found that groups exist in the SourceForge project network. Furthermore, we explore possible reasons for the formation of those groups by examining assortative mixing coefficients for projects categories. Among them, we found projects with the same programming languages, operating systems and topics are more likely to be grouped together. Our research provides useful information to study the interaction between projects and the communication and information flow in OSS virtual organization.

CHAPTER 5

SIMULATION AND VALIDATION

5.1 Introduction

Analysis involves validation and interpretation of the mined patterns. With data extracted from the web and information discovered from generalization, models can be built to simulate the studied phenomenon [1]. Those simulation models need to be validated for their correctness.

One objective of Open Source Software studies is to model OSS developers so as to understand their behaviors and be able to predict the developers' network development. This task can be divided into two-steps: simulate the developers' network and validate the simulation models. We use agent-based tools to simulate and validate the OSS developers' network. This chapter will be focused on how one method is used to perform validation [2]

Agent-based modeling has become an attractive computational methodology in recent years. Its popularity results from the fact that it allows for complex systems to be simulated in a relatively straightforward way. Unlike traditional

[1]This chapter is based on paper "A Multi-Model Docking Experiment of Dynamic Social Network Simulations", Agent2003, Chicago, IL, October 2003 [117]. It is also published as a paper "A docking experiment: Swarm and Repast for social network modeling", Seventh Annual Swarm Researchers Meeting (Swarm2003), Notre Dame, IN, 2003 [118].

[2]The original model and simulation implemented using the Swarm toolkit, and used for the validations studies in this chapter, were developed by Y. Gao [32].

mathematical simulation tools, agent-based modeling simulates artificial worlds based on components called agents and defines rules to determine the interactions of agents. Although agent-based modeling is used commonly in simulations, it is not guaranteed to provide an accurate representation of a particular empirical application [4]. In this context, Axtell et al claimed "It seems fundamental to us to be able to determine whether two models claiming to deal with the same phenomenon can, or cannot, produce the same result" [5].

There are three major approaches to validating an agent-based simulation. The first approach is to compare the simulation output with the real phenomenon. This way is relatively simple and straightforward. However, often we cannot get complete real data on all aspects of the phenomenon. The second approach compares agent-based simulation results with results of mathematical models. The disadvantage of this way is that we need to construct mathematical models which may be difficult to formulate for a complex system. The third way is by docking with other simulations of the same phenomenon. Docking is the process of aligning two dissimilar models to address the same question or problem, to investigate their similarities and their differences, but most importantly, to gain new understanding of the question or issue [14].

Axtell, Axelrod, Epstein and Cohen describe a docking or alignment process and experiment for verifying simulations [5]. By comparing simulations built independently using different simulation tools, the docking or alignment process may discover bugs, misinterpretations of model specification, and inherent differences in toolkit implementations. If the behaviors of the multiple simulations are similar, then validation confidence is increased. North and Macal report on such an experiment using Mathematica, Swarm and RePast to simulate the Beer Distribu-

tion Game (originally simulated using system dynamics simulation methods) [87]. In Louie and Ashworth [3], docking is done by comparing results of the canonical Garbage Can model with those of "NK Model". Although the above docking experiments show the importance and advantages of docking, there are only a few docking studies and none has used topological properties of social networks as docking parameters.

The rest of this chapter is organized as follows. The first section provides background on our Open Source Software (OSS) study and simulation. The following section discusses different network models and their features. Section 3 describes the formation of OSS social network. Section 4 presents our docking simulation tools – Swarm and RePast. Section 5 describes the OSS simulation models. Docking results are given in Section 6. The last section presents conclusions and future plans.

5.2 Social Network Models

Social network theory is a conceptual framework through which we view the OSS developer movement. The theory, built on mathematical graph theory, depicts interrelated social agents as nodes or vertices of a graph and their relationships as links or edges drawn between the nodes [108]. The number of edges (or links) connected to a node (or vertex) is called the index or degree of the node.

The Open Source Software (OSS) development movement is a classic example of a dynamic social network. It is also a prototype of a complex evolving network. Prior research suggests that the OSS network can be considered a complex, self-organizing system [72, 73, 75, 76]. These systems are typically comprised of large numbers of locally interacting elements.

Special interests in social networks are the evolutionary processes and associated topological formation in dynamic growing networks. Early work in this field by Erdős and Rényi focuses on random graphs, i.e., those where edges between vertices were attached in a random process (called ER graphs here) [7]. Some other evolutionary mechanisms include: 1) the Watts-Strogatz (WS) model [109], 2) the Barabasi-Albert (BA) model with preferential attachment [2, 8, 9], 3) the modified BA model with fitness [7, 10], and 4) an extension of the BA model (with fitness) to include dynamic fitness based on the project life cycle reported in [33, 34, 36, 37, 76].

5.2.1 Erdős - Rényi (ER) Model

Paul Erdős and Alfréd Rényi proposed a simple network model to capture some properties of networks. Their graph theory is based on probabilistic methods. A random graph $G_{n,p}$ is defined as n labeled nodes connected by m edges with probability $p^m(1-p)^{(M-m)}$, where $M = \frac{1}{2}n(n-1)$ is the number of maximum possible edges. They also define an alternative model $G_{n,m}$, which is a graph with n nodes and m edges, which are selected with equal probability from $\frac{1}{2}n(n-1)$ possible edges.

The ER model reproduces the small-world effect in real networks. The number of nodes at a distance l from a given node is z^l. To encompass the entire network, z^l is approximately equal to n. The typical distance between a pair of nodes in a ER model is found to be proportional to $logn/logz$.

However, the distributions of index values for the random graphs do not agree with the observed power law distribution for many social networks. The reason is that in a random graph, the majority of nodes have the same degree because

the edges are selected randomly. The degree distribution of a random graph is a Poisson distribution. Furthermore, the clustering coefficient of a random graph is $C = p$, while real networks have large clustering coefficients.

5.2.2 Other Network Models

Based on ER model, some other network models have been further developed to capture more properties of real networks. Watts and Strogatz introduced a model which captures the clustering property of real networks and was extended to include some random reattachments to capture the small world property. The WS network is constructed as follows: Start with a one-dimensional lattice of N vertices, connect each vertex to its neighbors k or fewer lattice spacing away; Randomly rewire a small fraction of edges with probability p such that duplicate and self-connected edges are avoided. However, WS model failed to display the power-law distribution of index values. The shape of the degree distribution of a WS model is similar to that of a random graph. All nodes have approximately the same number of edges.

Barabási and Albert addressed the power-law degree distribution of many large networks. Such networks are called *scale free*. BA model introduces concepts of network growth and preferential attachment. Unlike ER model and WS model which start with a fixed number N vertices, BA model starts with a small number $(m_0)$of nodes, at each step, a new node is added with $m \leq m_0$ edges connecting to m different nodes in m_0. The connected edges are not chosen randomly, instead, the probability $\prod(k_i)$ to choose a node depends on the degree k_i of node i, that is $\prod(k_i) = \frac{k_i}{\sum_j} k_j$. The BA model added preferential attachment, both preserving the realistic properties of the WS model and also displaying the power-law distri-

bution. In this model, the oldest nodes have the largest number of edges because they have the longest lifetime to accumulate edges. This can not explain the fact that degree and growth rate of many real networks do not depend only on ages. For example, many new web sites can get many links in a short time, the "young upstart" phenomenon.

Barabási and Albert extended BA model to include a competitive aspect. In this model, each node has a intrinsic ability to compete for new edges, which is represented as a fitness parameter η_i, where η_i is a constant associated with node i. At each step, a new node j is added with η_j, where η_j is selected from a distribution $p(\eta)$. The probability of connecting node i to node j is proportional to both the degree and the fitness of node i, that is, $\prod_i = \frac{\eta_i k_i}{\sum_j \eta_j k_j}$. In this model, a relatively new node can attract more edges if it has a high fitness.

As part of an Open Source Software development study, Gao introduced a dynamic fitness into BA model [32]. The fitness parameter η_i is not constant, instead, it decays with time with a uniform rate for all nodes. This model explains OSS development better than the other three models.

5.3 OSS Network

The Open Source Software community can be described as a dynamic social network. In our model of the OSS collaboration network, there are two entities – developer and project. The network can be illustrated as a graph. In this network, nodes are developers. An edge will be added if two developers are participating in the same project. Edges can be removed if two developers are no longer participating on the same project. The study of the OSS collaboration network can help us understand the evolution of the social network's topology, the develop-

ment patterns of each individual object and the impact of the interaction among objects on the evolution of the overall network system.

We use agent-based modeling to simulate the OSS development community. Unlike developers, projects are passive elements of the social network. Thus, we only define developers as the agents which encapsulate a real developer's possible daily interactions with the development network. Our simulation is time stepped instead of event driven, with one day of real time as a time step. Each day, a certain number of new developers are created. Newly created developers use decision rules to create new projects or join other projects. Also, each day existing developers can decide to abandon a randomly selected project, to continue their current projects, or to create a new project. A developer's selection is determined by a Java method based on the relative parameter and the degree of the developer.

5.4 Simulation Tools

5.4.1 Agent-based Simulation

Many natural phenomena existing in the world are not easily modeled in a linear way. Those phenomena comprised of small entities, each of which has its own behavior and interacts with others to work as a whole. However, the whole system is not just the simple sum of its small parts, but a more complex entity with new and unpredictable features [71]. Such systems are called Complex Adaptive systems (CAS). Examples of CAS include weather, economy, stock market, societies, as well as human beings, etc.

Compared to other systems, a CAS has some special features. The system is *self-organizing*, which evolves without outside or external involvement. In other words, its behaviors result from the interaction among its components. A CAS

is *adaptive*. Instead of passively responding to its environment, it learns the environment and changes its patterns of behavior. Moreover, some new features may appear as the result of the interaction of its component entities. This phenomenon is called *emergence*. Individual components respond to emergent features and change their behaviors.

Because the consequences of a CAS is non-linear and non-predictable, traditional mathematics or simulation techniques can not be applied to study a CAS. Agent-Based Simulation, also called *Individual-Based Modeling* is a technique to study CAS. Agent-based simulations simulate the global consequences of local interaction of individual entities, called *agents*. An *agent* is an entity, such as an organism, person or a social organization whose behaviors follow some rules to interact with other agents in the system, and change its behavior to adapt to the system. Instead of having aggregate global properties, each individual agent can have its own characteristics. These characteristics can change as the agents interact with their environment. Agents can be identical or different with their own behavior and interaction rules, respectively. An agent-based simulation model usually consists of agents with individual behaviors and the environment where agents interact with each other. By specifying rules for agents' behavior and their interaction, we can use this model to capture the emergent consequences of the studied system. Many tools are developed for agent-based simulation. Among them, we use *Swarm* and *Repast* because they are open source, powerful, and popular.

5.4.2 Swarm

Swarm was originally started at Santa Fe Institute (SFI) in New Mexico to develop a set of software package for agent-based simulations for Complex Adaptive Systems. It is now under the control of Swarm Development Group.

Swarm provides a set of libraries which work on a wide range of computer platforms to support agent-based simulations [60]. These libraries are written by a programming language, Objective-C, to enable individual instances be created by "classes". Swarm recently incorporated Java extensions to call library functions. Swarm also provides display, control and analysis tools. The Swarm software is Open-Source software to the general public under GNU licensing terms.

Swarm simulations have a hierarchical structure. Swarms are the basic component in a Swarm system. A swarm is a collection of objects and a schedule of events over those objects. A Swarm system separates tasks of actual model and observation by using two level swarms – *Model Swarm* and *Observer Swarm*.

Model Swarm encapsulates the real simulation. The model swarm creates the individual agents, schedules their activities, and collects information about agents An agent can comprise of swarms of other agents. Modelers can define a set of rules to describe the interaction of agents. In addition, the model swarm has a set of inputs which are parameters and outputs which are observable by *Observer Swarm*.

Like *Model Swarm*, *Observer Swarm* consists of a collection of agents, a schedule of events, as well as inputs and outputs. *Model Swarm* is one component of *Observer Swarm*. The purpose of the observer swarm is to provide both real-time data presentation and storage of data for later analysis. The observer swarm schedules its objects to read data from the model and draw graphs. The inputs to

the observer swarm are configurations of the observer tools: i.e., tool types. The outputs are observations.

5.4.3 Repast

The Recursive Porous Agent Simulation Toolkit (RePast) is a software toolkit for agent-based simulation created by Social Science Research Computing at the University of Chicago [95]. Repast is now maintained by the non-profit volunteer Repast Organization for Architecture and Development (ROAD).

Like Swarm, RePast provides an integrated library of classes for creating, running, displaying, and collecting data from an agent-based simulation [20]. In addition, RePast is written in pure Java which has better portability and extensibility than Swarm. Furthermore, RePast provides some different library packages which provide features such as network display, QuickTime movies and snapshot.

A typical Repast simulation consists of at least two classes – the agent class and the model class. An agent class includes properties and movement of the agent. If users want to display agents, graphics interfaces or methods must be coded in the agent class.

A model sets up and controls the actual simulation. All Repast simulation models implement the *SimModel* interface which is implemented by an abstract class *SimModelImpl*. A model class is composed of several parts:

1. Parameters and Infrastructure variables: Parameters are initial variables for a run of the model. Infrastructure variables are things like a schedule or a graphic surface for displaying agents.

2. Template methods: Methods to create agents, environment, display, as well as schedule.

3. Accessor Methods: Methods to set and get parameters.

4. Interface Required Methods: Methods which are required by compilers, i.e. methods to initialize a run.

5.5 OSS Developer-Project Bipartite Network Simulation Model

We use agent-based simulations described in Section 5.4 to model the formation and growth of OSS developer-project bipartite network. Although Swarm and Repast simulations are created separately to dock the simulation results, these two simulations have the same model structure. Among the two entities of the network, developers and projects, developers are the active entities. Thus, we define a developer as an agent in our model.

The model is time-step triggered. At each time step, which is defined as a day, a certain number of developers are created and added into the network. The number of newly created developers is a random number following a uniform distribution. The model maintains three lists – developer list, project list, and edge list. Both new developers and existed developers take some actions and the network is updated during each time step. By this model, we can get a developer-project network at any time.

As shown in Figure 5.1, at each step, a new developer can create or join a project. The decision is based on two parameters – $P(CN)$ (create) and $P(JN)$ (join). If a new developer decides to create a new project, all lists will be updated. New elements (the new developer and the project created) will be added into developer list and project list, respectively. A link will be created between the developer and the projects. If a new developer joins a project, only developer list and edge list get updated. However, the very first developer must create a new

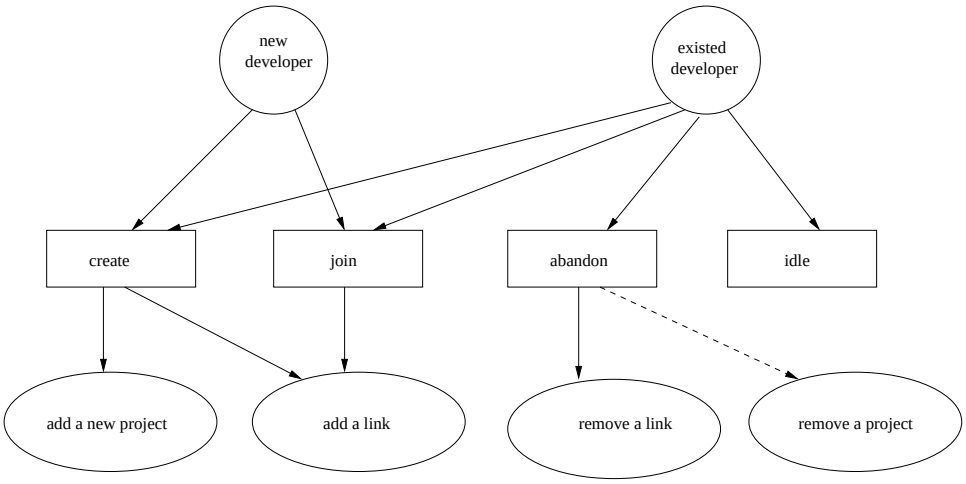


Figure 5.1. A Developer's possible actions

project to get the model started.

An existing developer has four kinds of actions – creating, joining, abandoning, and idling, based on four parameters – $P(CO)$ (create), $P(JO)$ (join), $P(AO)$ (abandon), and $P(IO)$(idle). Creating and joining are the same as those for new developers, except that there is no new developer added into developer list. If a developer abandons a project, their link should be removed. If there is no any developer link to a project, the project will be deleted from the edge list. For idling, no updating is performed.

This model structure is applied to all four social network models. Their difference lies in the rules a developer uses to pick an action and a project, which are represented by different implementations of parameters. For the random network model, all rules are generated randomly by a uniform random generator. The six probability action parameters are calculated from the means of the empirical data.

Our scale-free model incorporates preferential attachment into the random network model. We assume the probability that a developer takes an action

depends on his degree – the number of projects he participates. Thus, there is a probability range for each action at each developer degree. The rule to select a project also considers the project degree – the number of developers participate in this project. The probability of a project i being selected is calculated as $P(i) = \frac{k_i}{\sum_j k_j}$.

In our scale-free network with the constant fitness model, the action rules are defined as those in the scale-free model. However, each project associates with a fitness η whose distribution follows an exponential distribution $p(\eta) = \lambda e^{-\lambda\eta}$, where $\lambda = 2$. The fitness is constant for the whole life of a project. The rule to select a project i is modified as $P_i = \frac{\eta_i k_i}{\sum_j \eta_j k_j}$.

We introduce a changed fitness into our fourth model. The fitness with each project is not constant, instead, it decays as the project ages. In our implementation, the fitness of each project decreases linearly by months. We make the rate of decay uniform for all projects.

5.6 A Multi-model Docking Experiment

In this section, we present the results of docking a RePast simulation and a Java/Swarm simulation of four dynamic social network models of the Open Source Software (OSS) community. Developer membership in projects is used to model the social network of developers. Social networks based on random graphs, preferential attachment, preferential attachment with constant fitness, and preferential attachment with dynamic fitness are modeled and compared to collected data. We describe how properties of social networks such as degree distribution, diameter and clustering coefficient are used to dock RePast and Swarm simulations of four social networks. The simulations grow "artificial societies" representing the

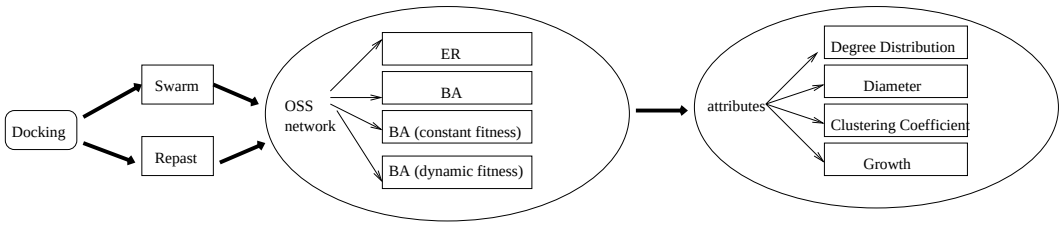


Figure 5.2. Docking process

SourceForge developer/project community.

5.6.1 The Docking Process

Validation using a docking process is an important stage in a simulation-based study of an OSS phenomenon [53]. The initial simulation was written using Swarm [32]. There are several reasons why validation using docking is necessary in this project. First, docking is used to test the correctness of the Swarm implementation. Second, docking provides the RePast version of the OSS simulation which we would like to use in our future research. RePast has several advantages in this project: it is written in pure Java which makes debugging easier; it provides us a graphical representation of the network layout; and most importantly, RePast 2.0 provides a distributed running environment [21].

As shown in Figure 5.2, Swarm simulations and RePast simulations are docked for four models of the OSS network. Our docking process began when the author of the swarm simulation (Y. Gao) wrote the docking specification. Then, the RePast version was written based on the docking specification. Simulations are validated by comparing network attributes generated by running these two simulation models.

Our swarm simulation has a hierarchical structure which consists of a *developer*

class, a *modelswarm* class, an *observerswarm* class and a *main* program. The *modelswarm* handles creating developers and controls the activities of developers. In *modelswarm*, a schedule is generated to define a set of activities of agents. The *observerswarm* is used to implement data collection and draw graphs. The *main* program is a driver to start the whole simulation.

The core of a swarm simulation consists of a group of agents. Agents in our simulation are developers. Each developer is an instance of a Java class. A developer has an identification id, a degree which is the number of links, and a list of projects participated by this developer. Furthermore, a developer class has methods to describe possible daily actions: create, join, abandon a project or continue the developer's current collaborations. A separate Java method models each of the first three possibilities. A fourth method encapsulates a developer's selection of one of the three alternatives. Here, three model parameters appear. Each represents the probability of one of the three developer activities. Comparison of a randomly generated number to these probabilities determines which behavioral method the agent will enact.

Our RePast simulation of the OSS developer network consists of a *model* class, a *developer* class, an *edge* class and a *project* class. The class structure of the simulation is different from that of the Swarm simulation. This is due in part to the graphical network display feature of RePast. The model class is responsible for creation and control of the activities of developers. Furthermore, information collection and display are also encapsulated in the *model* class. The *developer* class is similar to that in Swarm simulation. An *edge* class is used to define an edge in OSS network. We also create a *project* class with properties and methods to simulate a project.

5.6.2 Docking Procedure

Our docking process sought to verify our RePast migration against the original Swarm simulation. To do so, the process began with a comparison of degree distributions between corresponding models. Upon finding differences, we compared each developer's actions.

The first attempt at docking compared degree distributions between these two simulations. The Swarm simulation used its built-in random number generator. The RePast simulation used the COLT random number generator from the European Laboratory for Particle Physics (CERN). From a graphical comparison of degree distributions for projects and developers over multiple runs of Swarm and RePast, we observed systematic differences between the two simulations' output. Over one subdomain of the developer degree distribution, Swarm had a higher count than RePast. Over another subdomain, Swarm had a lower count. The next step in the docking process determined that the random number generators did not cause this systematic difference. We ran the two simulations using the exact same set of random numbers: each simulation used the same random number generator with the same seed. The developer and project degree distributions from these runs, however, revealed similar systematic differences between the two simulations.

To determine the exact reasons for this difference, we had the simulations log the action that each developer took during each step. Comparing these logs, two reasons for the differences emerged.

First, we determined that one simulation would occasionally throw an SQL Exception (we store simulation data in a relational database for post-simulation analysis). To recover from such an error, the simulation does not log the devel-

oper's actions: it moves on to the next developer. Since the developer's previous actions affect his/her future actions, one error can cause more discrepancies between the two simulations at future time steps. We found the cause of this error to be a problem with the primary keys in the links table of our SQL database (this is a programming bug). Further inspection of the data logs showed that each simulation's data snapshots which are used in analyzing macroscopic graph properties were out of phase by one unit time. Even if the corresponding simulation ran identically, this extra time step prevented the output from matching. We found that the Swarm scheduler begins at time step 0 while the RePast scheduler begins with time 1. Thus, when snapshots were logged at time step 30, Swarm had actually performed one extra time step.

With these two problems corrected, the corresponding logs of the developers' actions matched. Using the same sequence of random numbers, the Swarm and RePast simulations produced identical output.

5.6.3 Experimental Results

This section describes docking of RePast and Swarm simulations on four OSS network models – ER, BA, BA with constant fitness and BA with dynamic fitness, and then presents results and comparisons.

Degree distribution $p(k)$ is the distribution of the degree k throughout the network. The degree k of a node equals the total number of other nodes to which it is connected. Degree distribution was believed to be a normal distribution, but Albert and Barabasi recently found it fit a power law distribution in many real networks [2]. Figure 5.3 gives developer distributions in four models implemented by Swarm and RePast. The X coordinate is the number of projects in which

each developer participated, and the Y coordinate is the number of developers in the related categories. From the figure, we can observe that there is no power law distribution in the ER model. The distribution looks more like the mathematically proven normal distribution. Developer distributions in the other three models match the power law distribution. However, there are slight differences between Swarm results and RePast results. We believe this difference is caused by different random generators associated with RePast and Swarm.

The diameter of a network is the maximum distance between any pair of connected nodes. The diameter can also be defined as the average length of the shortest paths between any pair of nodes in the graph. In this chapter, the second definition is used since the average value is more suitable for studying the topology of the OSS network. Figure 5.4 shows the evolution of the diameter of the network. We can see that RePast simulations and Swarm simulations are docked. In the real SourceForge developer collaboration network, the diameter of the network decreases as the network grows. In our models, we can observe that ER model does not fit the SourceForge network, while other three models match the real network.

The neighborhood of a node consists of the set of nodes to which it is connected. The clustering coefficient of a node is a fraction representing the number of links actually present relative to the total possible number of links among the nodes in its neighborhood. The clustering coefficient of a graph is the average of all the clustering coefficients of the nodes represented. Clustering is an important characteristic of the topology of real networks. So the clustering coefficient, a quantitative measure of clustering, is also an attribute we investigated. Clustering coefficients for the developer network as a function of time are shown in Figure

5.5. All models are docked very well. We can observe the decaying trend of the clustering coefficient in all four models. The reason is that with the evolution of the developer network, two co-developers will less likely join a new project together because the number of projects they participated are approaching their limits.

Figure 5.6 shows the total number of developers and projects relative to the time period in four models, which describe the developing trends of size of developers and projects in the network. The number of developers and the size of projects are almost the same for Swarm and RePast simulations.

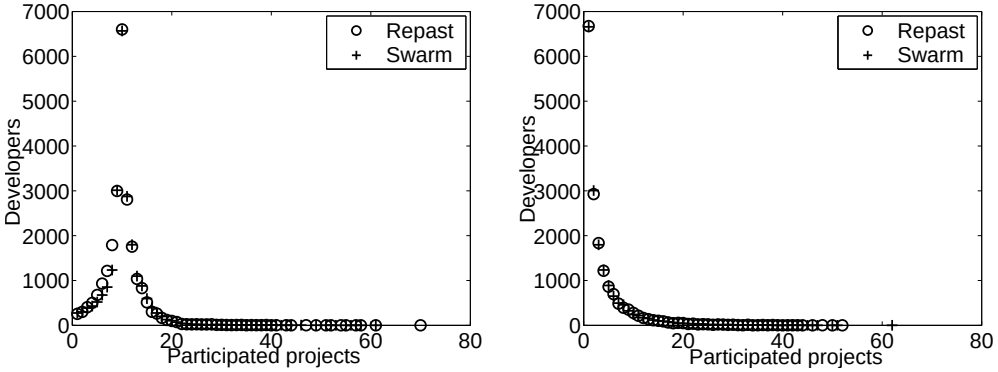
5.7 Conclusion

This chapter discusses validation of agent-based simulations using the docking process. It describes four simulation models of OSS developer network using Swarm and RePast. Properties of social networks such as degree distribution, diameter and clustering coefficient are used to dock RePast and Swarm simulations of four social networks. Experimental results show that docking two agent-based simulations can help validate a simulation. This chapter showed that a docking process can also be used to validate a migration of a simulation from one software package to another. In our case, the docking process helped with the transfer to RePast to take advantages of its features. RePast simulation runs faster than Swarm simulation because RePast is written in pure Java while Swarm is originally written in Object C which may cause some overheads for Java Swarm. Furthermore, RePast provides more display library packages such as network packages which help users to do analysis.

There are several directions of future work. First, we just compare the results

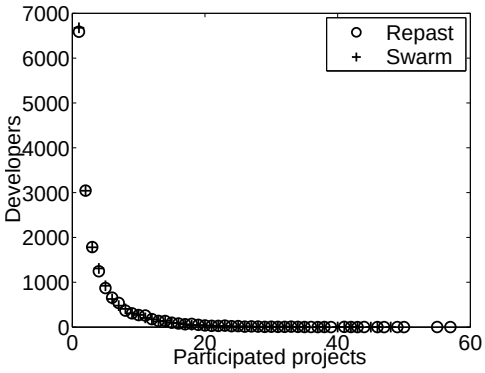
of one simulation run. There is slight difference between docking results, which we believe is caused by different random generators. We will run both Swarm simulations and RePast simulations many times to see if this difference can be ignored. We will do statistic analysis on these results. For example, a two-sample t-test will be used to compare means of Swarm and RePast results.

Moreover, our previous work just docked several network properties. More properties should also be docked to validate our simulation models. Such network parameters include average degree, cluster size distribution, fitness and life cycle.

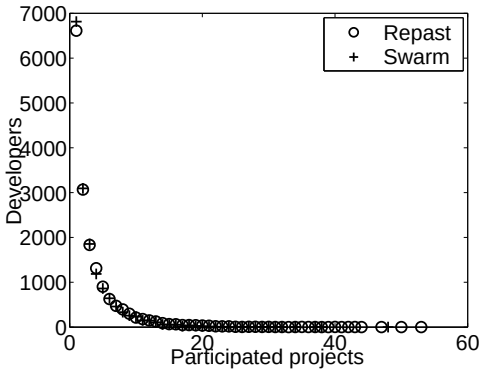


(a) ER

(b) BA

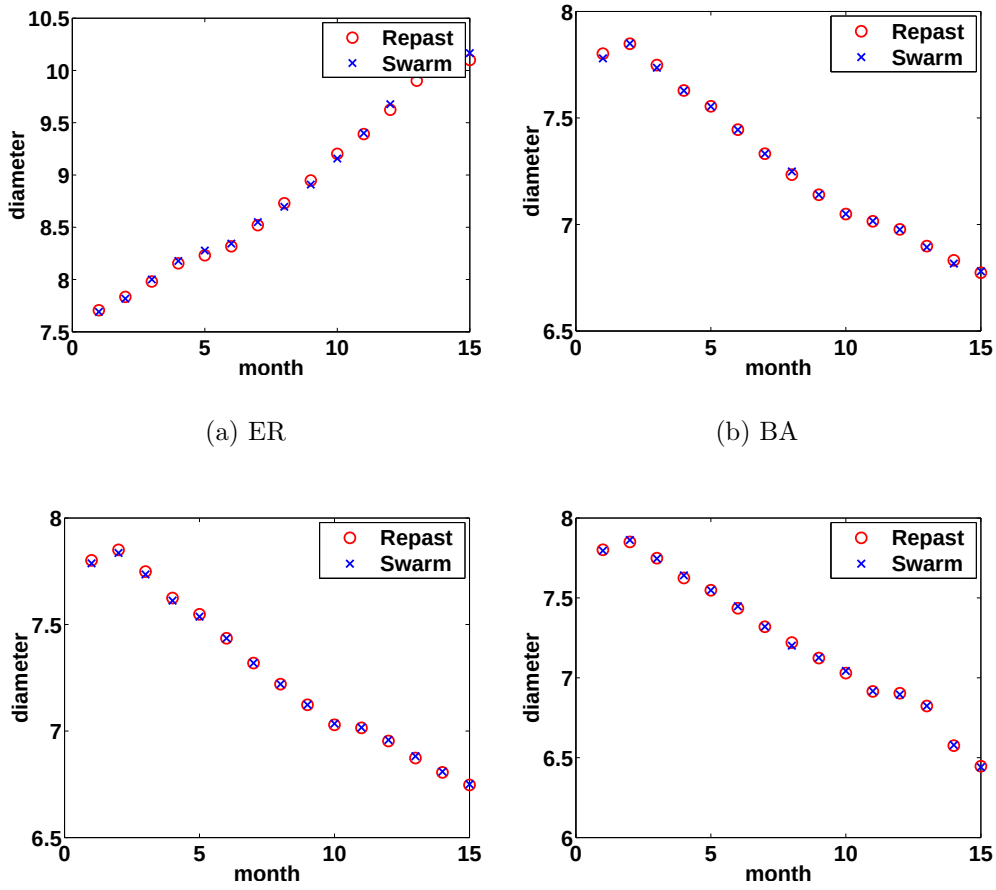


(c) BA with constant fitness



(d) BA with dynamic fitness

Figure 5.3. Degree distribution



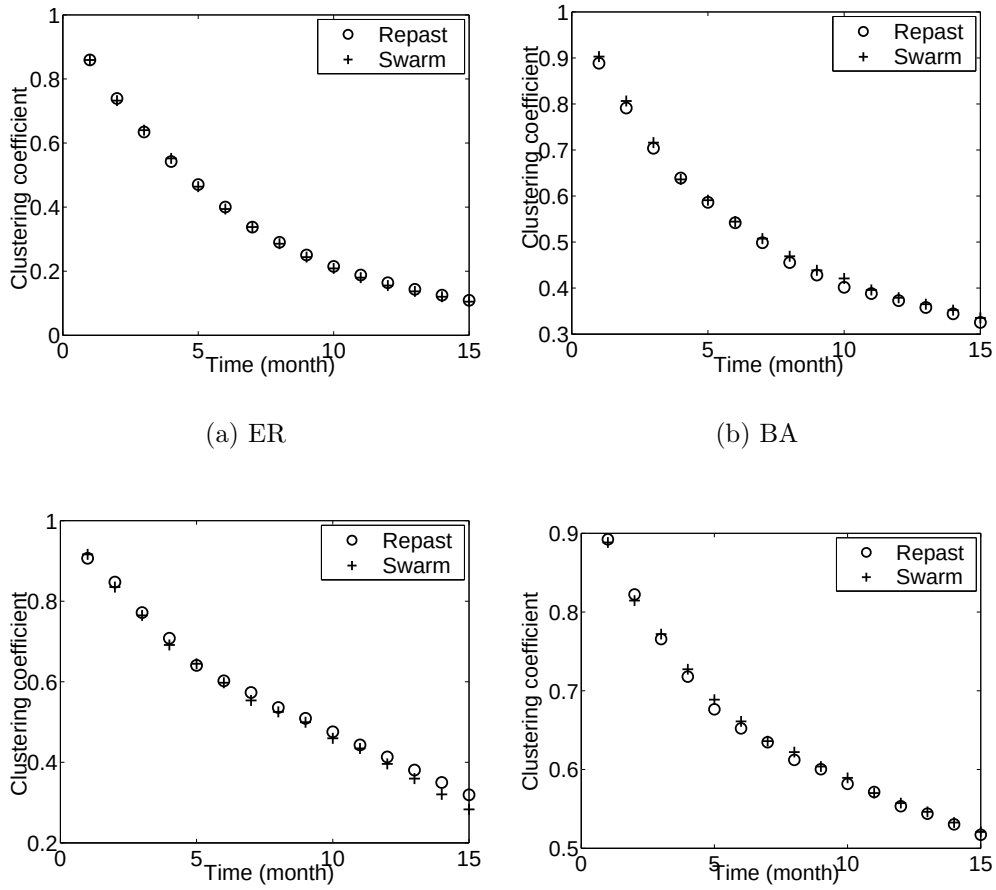
(a) ER

(b) BA

(c) BA with constant fitness

(d) BA with dynamic fitness

Figure 5.4. Diameter



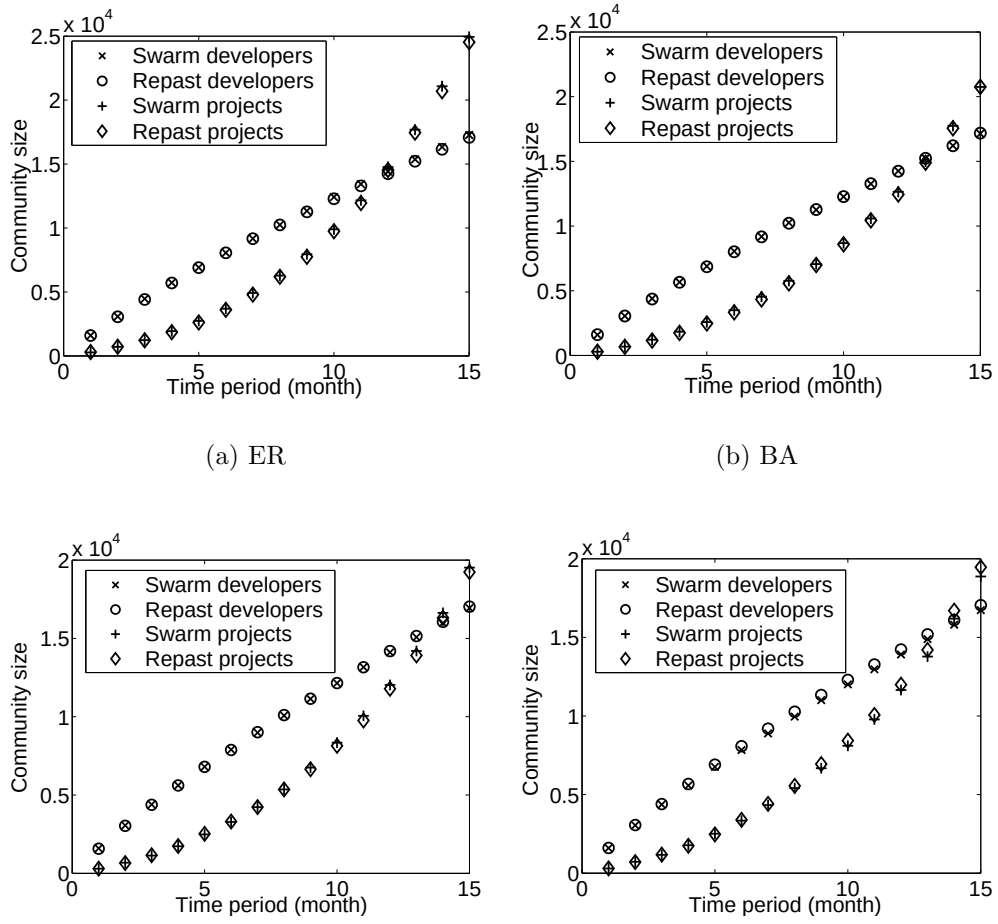
(a) ER

(b) BA

(c) BA with constant fitness

(d) BA with dynamic fitness

Figure 5.5. Clustering coefficient



(a) ER

(b) BA

(c) BA with constant fitness

(d) BA with dynamic fitness

Figure 5.6. Community size development

CHAPTER 6

CONCLUSIONS AND FUTURE WORK

In this dissertation, we model the OSS community as a social network and study network properties on OSS community network. Our OSS study investigate behaviors, interactions, and mechanisms across multiple projects. We address the intrinsic mechanisms which lie in OSS networks to explain some OSS specific features such as roles of developers, communications, reliability of the OSS community. This chapter draws some conclusions and discusses potential future work.

6.1 Conclusions

We address the challenge of data collection for OSS study. Characteristics of data sources for OSS study are presented. Our mining process incorporates web mining and data mining. The whole process consists of four parts – identification and extraction, cleansing and preprocessing, analysis, and reporting. Our web mining and data mining framework gives researchers a general solution to collect and analyze data from on-line data repositories.

The OSS community can be modeled as a complex social network where developers can be defined as different roles according to their contributions. We apply statistical and social network analysis on the Open Source Software project and developer networks at SourceForge.net. Our statistical study finds that different sized projects have different member distributions where large projects consist

mainly of co-developers and active users, while project leaders and core-developers are the main part of small projects. Our network topological analysis shows that the SourceForge OSS development community is a self-organizing system which obeys scale-free behaviors and is a small-world with the small average distance and the high clustering coefficient. Moreover, We identify the important effect of co-developers and active users in the OSS network. Our research provides useful information on the underlying structure and evolution of the OSS community, and the diffusion of information between projects.

To continue our social network analysis, we identify community structures for the SourceForge project network. We found that there exist closely related groups among SourceForge projects. The formation of groups in projects can be explained by assortative mixing coefficients for projects categories. Projects with the same programming languages, operating systems and topics are more likely to be grouped together. Our research provides useful information to study the interaction between projects and the communication and information flow in OSS virtual organization.

This dissertation also discusses the docking process to validate OSS agent-based simulation models. Our OSS simulations are built using Swarm and RePast. We dock these two simulation models by comparing properties of social networks for four different kinds of networks. Our results show that a docking process can also be used to validate a migration of a simulation from one software package to another.

6.2 Future Work

There are several directions of future work. First, our current OSS networks contain no weights on vertices and edges. The limitation of such a construction is that all nodes and edges are considered to be the same important in the whole network, which is not the case in the real OSS development. Some developers and projects are more crucial in the interaction of OSS community. For example, core developers should be assigned more weight because they have more contributions in the project development. If two developers or projects have more interactions, the edge between them might also have higher weight. Our future work will be focused on how to create such weighted networks and explore network characteristics in such networks. We believe a weighted network can reflect OSS phenomenon more accurately.

Secondly, we plan to apply more social network property analysis on our OSS networks. For example, we will study *Closeness* of the OSS network. *Closeness* distinguishes nodes which have shortest paths to all others. It reflects the ability of the node to get information or communication in the network. This property is important in the study of OSS because we can find if a successful project has a higher closeness than other projects. Other properties we are interested include network reach which measures the importance of edges and betweenness centrality which indicates the importance of a node's location in the network.

Lastly, we will apply our analysis on Sourceforege.net to other OSS hosting web sites. Although SourceForge is the largest OSS web site, it doesn't represent all OSS projects. Many successful OSS projects such as Linux, Perl, and Apache have their own web sites. We want to know if we can get the same conclusions on other OSS project web sites as SourceForge.net. Furthermore, the public archives

maintained on those sites may have different formats and contents. Thus, more web mining and data mining algorithms and techniques will be designed in the future work.

APPENDIX A

WEB ROBOT FOR GRABBING INFORMATION ON SOURCEFORGE.NET

We designed a web crawler, which is implemented using Perl because Perl provides useful modules to help web retrieval such as HTTP communication, HTML parsing, etc. The crawler consists of a URL access method, a web page parser with some extractors, and a back-end database. The key component of our web crawler is the parser, which includes a word extractor, a table extractor and a link extractor. A table extractor identifies the location of the data in a table. A link extractor retrieves links contained in a web page.

```
#!/usr/local/bin/perl
use lib qw( ..);
use lib "/afs/nd.edu/user9/jxu1/research/webmining/jin/\
lib/HTML-TableExtract-1.08/lib";
use HTML::TableExtract;
use LWP::Simple;
use Data::Dumper;
my $str = "http://sourceforge.net/project/stats/?group_id=";

for (my $s = @ARGV[0]; $s <= @ARGV[1]; $s++){
  $file = "result".$s."k.txt";
   $m = ">>".$file;
  print $m . "\n";
my $lo = $s."000";
my $hi = $s."999";
for (my $i = $lo; $i <= $hi; $i++){
  print $i. "\n";
  my $te  = new HTML::TableExtract( depth=>1, count=>3, gridmap=>0);
```

```
  $url = $str . $i;
  #print $url . "\n";
#my $content = system ("wget". "-o  " " .$url);
my $content = get($url);
my $random = int (rand 3);
my $cmd = "sleep ".$random;
print "before ".$cmd . "\n";
system($cmd);
print "end\n";
if(defined $content){
  #print $content;
}
else {
 # print "Error: Get failed";
}
#print ("content". "\n");
$te->parse($content);
#print ("parse" . "\n");
foreach my $ts ($te->table_states)
{
   foreach $row ($ts->rows)
   {
    open (outfile, $m );

    print outfile $i;
    print outfile "|";
    @a = @$row;
    $length = @a;
    if ($a[0] !~ /Lifespan/){
      for (my $j = 0; $j < @a; $j++){
#print  $a[$j];
  print outfile $a[$j];
  print outfile "|";
       }
       print outfile "\n";
    }
   close (outfile);
   }
 }
}
}
```

APPENDIX B

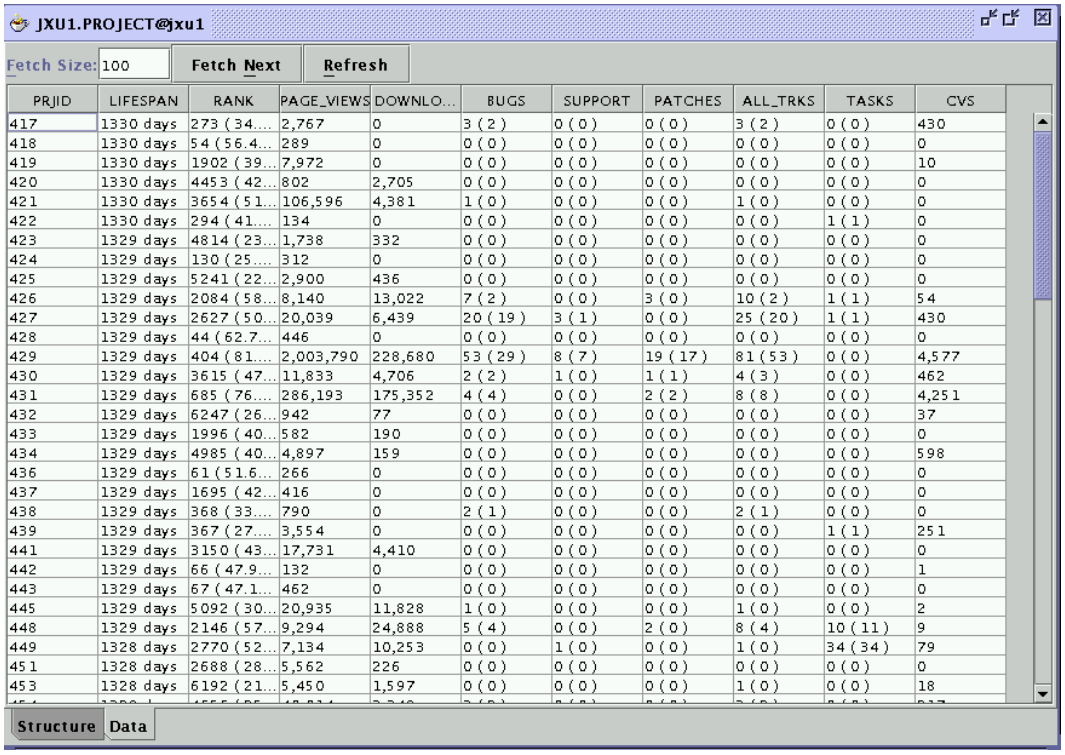
DATA MINING PRACTICES ON WEB RETRIEVD DATA

The web raw data we retrieved from SourceForge needs to be processed so that they can be feed into the analysis software [1]. This includes several steps. First of all, some fields of the web data we collected are not numerical or categorical, as shown in Figure B.1. We need to extract numerical data from these fields. This can be implemented using SQL. The resulting data are of numerical format, as shown in Figure B.2. The numerical fields of the resulting data are continuous. To make the data mining software run faster, we need to discretize the data. In other words, we create 10 buckets for each field, such that each bucket contains the same number of records. This step can be implemented using the following SQL script, as shown in Figure B.3.

The data is now ready to be feed into the data mining software. We use the Oracle Data Mining Software [90] and JDeveloper [56] to explore the application of data mining to web data. In the next section, we show in detail the process of applying data mining techniques to analyze the processed data.

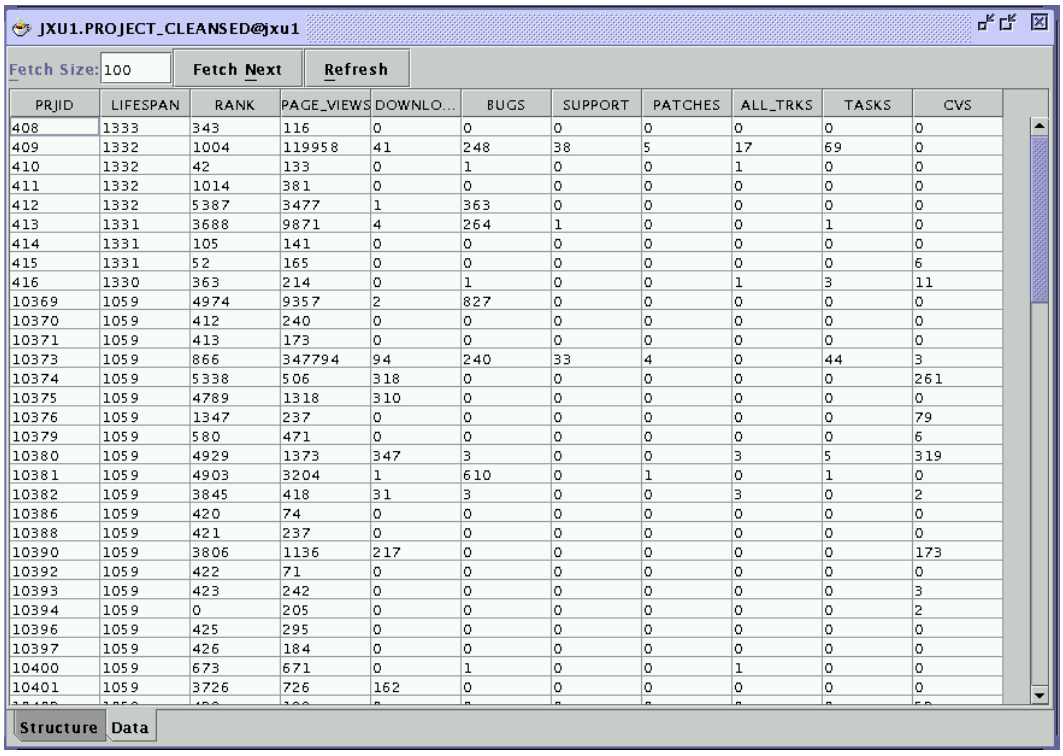
The purpose of this section is to show how we can do analysis on web retrieved data. We apply several data mining algorithms on OSS web data from SourceForge. Among them, we use Naive Bayes, Classification and Regression

[1] The work in this part is collaborative with Yingping Huang [119]



PRJID	LIFESPAN	RANK	PAGE_VIEWS	DOWNLO...	BUGS	SUPPORT	PATCHES	ALL_TRKS	TASKS	CVS
417	1330 days	273 (34....	2,767	0	3 (2)	0 (0)	0 (0)	3 (2)	0 (0)	430
418	1330 days	54 (56.4...	289	0	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0
419	1330 days	1902 (39...	7,972	0	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	10
420	1330 days	4453 (42...	802	2,705	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0
421	1330 days	3654 (51...	106,596	4,381	1 (0)	0 (0)	0 (0)	1 (0)	0 (0)	0
422	1330 days	294 (41....	134	0	0 (0)	0 (0)	0 (0)	0 (0)	1 (1)	0
423	1329 days	4814 (23...	1,738	332	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0
424	1329 days	130 (25....	312	0	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0
425	1329 days	5241 (22...	2,900	436	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0
426	1329 days	2084 (58...	8,140	13,022	7 (2)	0 (0)	3 (0)	10 (2)	1 (1)	54
427	1329 days	2627 (50...	20,039	6,439	20 (19)	3 (1)	0 (0)	25 (20)	1 (1)	430
428	1329 days	44 (62.7...	446	0	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0
429	1329 days	404 (81....	2,003,790	228,680	53 (29)	8 (7)	19 (17)	81 (53)	0 (0)	4,577
430	1329 days	3615 (47...	11,833	4,706	2 (2)	1 (0)	1 (1)	4 (3)	0 (0)	462
431	1329 days	685 (76....	286,193	175,352	4 (4)	0 (0)	2 (2)	8 (8)	0 (0)	4,251
432	1329 days	6247 (26...	942	77	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	37
433	1329 days	1996 (40...	582	190	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0
434	1329 days	4985 (40...	4,897	159	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	598
436	1329 days	61 (51.6...	266	0	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0
437	1329 days	1695 (42...	416	0	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0
438	1329 days	368 (33....	790	0	2 (1)	0 (0)	0 (0)	2 (1)	0 (0)	0
439	1329 days	367 (27....	3,554	0	0 (0)	0 (0)	0 (0)	0 (0)	1 (1)	251
441	1329 days	3150 (43...	17,731	4,410	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0
442	1329 days	66 (47.9...	132	0	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	1
443	1329 days	67 (47.1...	462	0	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0
445	1329 days	5092 (30...	20,935	11,828	1 (0)	0 (0)	0 (0)	1 (0)	0 (0)	2
448	1329 days	2146 (57...	9,294	24,888	5 (4)	0 (0)	2 (0)	8 (4)	10 (11)	9
449	1328 days	2770 (52...	7,134	10,253	0 (0)	1 (0)	0 (0)	1 (0)	34 (34)	79
451	1328 days	2688 (28...	5,562	226	0 (0)	0 (0)	0 (0)	0 (0)	0 (0)	0
453	1328 days	6192 (21...	5,450	1,597	0 (0)	0 (0)	0 (0)	1 (0)	0 (0)	18

Figure B.1. Project statistics web data. Data under LIFESPAN contain "days"; data under RANK, BUGS, SUPPORT, PATCHES, ALL_TRKS, TASKS contain parenthesis; data under PAGE_VIEWS, DOWNLOADS contain commas. Such data need to be transformed into numerical representations.



JXU1.PROJECT_CLEANSED@jxu1

Fetch Size: 100 | Fetch Next | Refresh

PRJID	LIFESPAN	RANK	PAGE_VIEWS	DOWNLO...	BUGS	SUPPORT	PATCHES	ALL_TRKS	TASKS	CVS
408	1333	343	116	0	0	0	0	0	0	0
409	1332	1004	119958	41	248	38	5	17	69	0
410	1332	42	133	0	1	0	0	1	0	0
411	1332	1014	381	0	0	0	0	0	0	0
412	1332	5387	3477	1	363	0	0	0	0	0
413	1331	3688	9871	4	264	1	0	0	1	0
414	1331	105	141	0	0	0	0	0	0	0
415	1331	52	165	0	0	0	0	0	0	6
416	1330	363	214	0	1	0	0	1	3	11
10369	1059	4974	9357	2	827	0	0	0	0	0
10370	1059	412	240	0	0	0	0	0	0	0
10371	1059	413	173	0	0	0	0	0	0	0
10373	1059	866	347794	94	240	33	4	0	44	3
10374	1059	5338	506	318	0	0	0	0	0	261
10375	1059	4789	1318	310	0	0	0	0	0	0
10376	1059	1347	237	0	0	0	0	0	0	79
10379	1059	580	471	0	0	0	0	0	0	6
10380	1059	4929	1373	347	3	0	0	3	5	319
10381	1059	4903	3204	1	610	0	1	0	1	0
10382	1059	3845	418	31	3	0	0	3	0	2
10386	1059	420	74	0	0	0	0	0	0	0
10388	1059	421	237	0	0	0	0	0	0	0
10390	1059	3806	1136	217	0	0	0	0	0	173
10392	1059	422	71	0	0	0	0	0	0	0
10393	1059	423	242	0	0	0	0	0	0	3
10394	1059	0	205	0	0	0	0	0	0	2
10396	1059	425	295	0	0	0	0	0	0	0
10397	1059	426	184	0	0	0	0	0	0	0
10400	1059	673	671	0	1	0	0	1	0	0
10401	1059	3726	726	162	0	0	0	0	0	0

Structure | Data

Figure B.2. The transformed data are in numerical format. Strings, commas, and parenthesis are eliminated.

```
create table project_build_binned as
select prjid, ntile(10) over (order by lifespan) lifespan,
              ntile(10) over (order by rank ) rank,
              ntile(10) over (order by page_views) page_views,
              ntile(10) over (order by downloads) downloads,
              ntile(10) over (order by bugs) bugs,
              ntile(10) over (order by support) support,
              ntile(10) over (order by patches) patches,
              ntile(10) over (order by all_trks) all_trks,
              ntile(10) over (order by tasks) tasks,
              ntile(10) over (order by cvs) cvs
  from project_build_unbinned
/
```

Figure B.3. The SQL script to discretize numerical data

Tree (CART) for classification, A Priori for association rules mining, K-means and Orthogonal-Cluster for clustering.

B.1 Classification

The Naive Bayes algorithm makes predictions using Bayes' Theorem. Naive Bayes assumes that each attribute is independent of others, which is not the case in our study. For example, the "downloads" feature is closely related to the "cvs" feature; the "rank" feature is closely related to other features, since it is calculated from other features.

CART builds classification and regression trees for predicting continuous dependent variables (regression) and categorical predictor variables (classification). We are only interested in the classification type of problems since the software we are using only handles this type of problems. Oracle's implementation of CART is called Adaptive Bayes Network (ABN). ABN predicts binary and multi-class targets. Thus discretizing the target attribute is desirable.

Classification includes the following steps: build models, test models, compute lift, and apply models. The table "projects_build_binned which stores the transformed data, contains 65,000 records. We randomly partition the table into three parts based on the 6:3:1 criteria. Each part is for building, testing and computing lift respectively. We call these parts model-build-data, model-test-data and lift-compute-data.

In our case, we try to predict "downloads from other features. As stated previously, the "downloads feature is binned into ten equal buckets. We predict which buckets "downloads resides based on the values of other features. As expected, the Naive Bayes algorithm is not suitable for predicting "downloads, since it is

related to other features, such as "cvs. Figure B.4 shows that the accuracy of Naive Bayes is less than 10%.

While Naive Bayes performs badly on predicting "downloads", the ABN algorithms can predict "downloads" quite accurately. As shown in Figure B.5, the accuracy is about 63%. At first sight, the accuracy 63% is not attractive, but it is a good prediction since we could only get 10% of correct predictions without classification. The lift computation confirms that the resulting classification model is quite good, as shown in Figure B.6. These figures show that all records whose "downloads" feature is 1 in just the first 20% of records. Figure B.7 shows the rules built by the ABN classification model. As we see from the figure, "downloads" is closely related to "cvs".

We conclude that the ABN algorithm is suitable for predicting "downloads" in our study. The following table compares the two algorithms, namely, ABN and Naive Bayes. From the table, we see that ABN takes much longer time to build a classification model, but the resulting model is much more accurate.

B.2 Association Rules

Association rule algorithm is applied to find the correlations between features of projects. The association rules mining problem can be decomposed into two subproblems:

- Find all combinations of items, called frequent itemsets, whose support is greater than the minimum support.
- Use the frequent itemsets to generate the association rules. For example, if AB and A are frequent itemsets, then the rule $A \rightarrow B$ holds if the ratio of

Name: PrjClassTest5
Type: Classification Model Test
Start Date: 8/21/03
Start Time: 2:22 PM
End Date: 8/21/03
End Time: 2:24 PM

Model: PrjClassBuild5

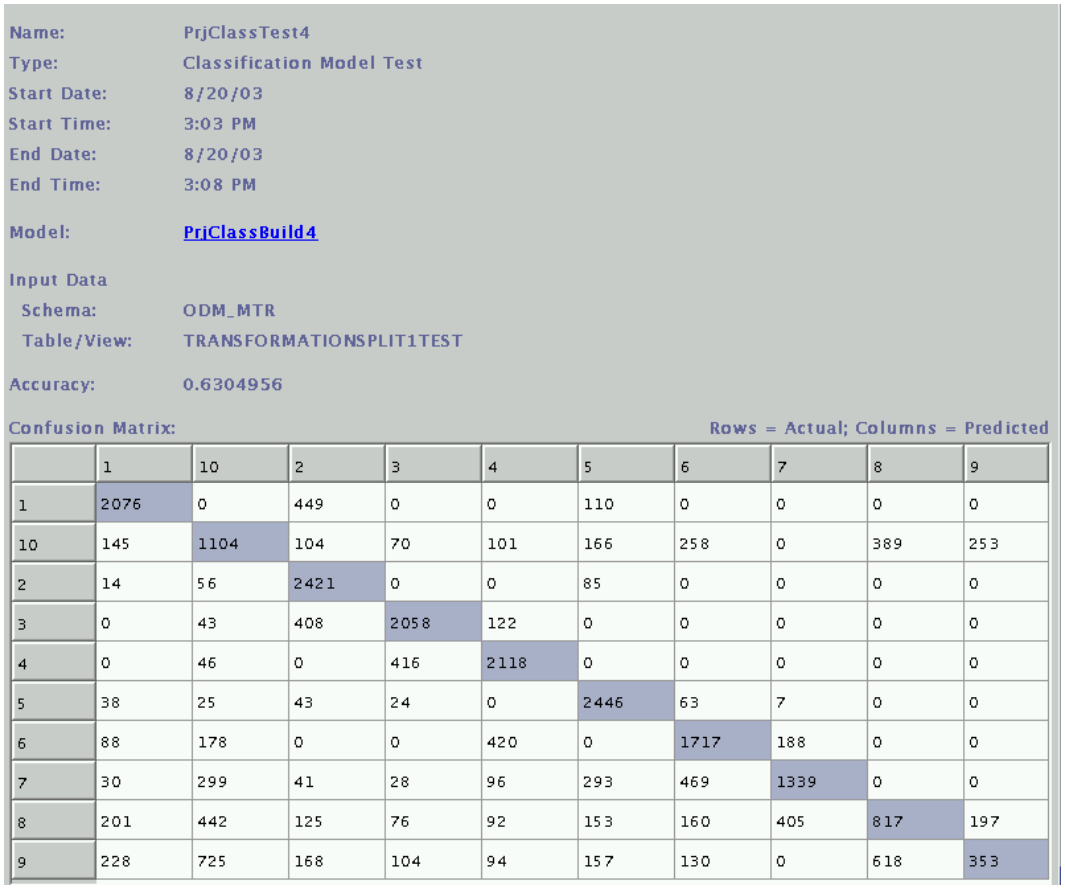
Input Data
Schema: ODM_MTR
Table/View: TRANSFORMATIONSPLIT1TEST

Accuracy: 0.09873893

Confusion Matrix: Rows = Actual; Columns = Predicted

	1	10	2	3	4	5	6	7	8	9
1	0	0	2635	0	0	0	0	0	0	0
10	0	0	2590	0	0	0	0	0	0	0
2	0	0	2576	0	0	0	0	0	0	0
3	0	0	2631	0	0	0	0	0	0	0
4	0	0	2580	0	0	0	0	0	0	0
5	0	0	2646	0	0	0	0	0	0	0
6	0	0	2591	0	0	0	0	0	0	0
7	0	0	2595	0	0	0	0	0	0	0
8	0	0	2668	0	0	0	0	0	0	0
9	0	0	2577	0	0	0	0	0	0	0

Figure B.4. Naive Bayes prediction for downloads. The "downloads" is predicted based on other project's statistics. Rows contain actual data. Columns contain predicted data. The accuracy of Naive Bayes is less than 10%



Name: PrjClassTest4
Type: Classification Model Test
Start Date: 8/20/03
Start Time: 3:03 PM
End Date: 8/20/03
End Time: 3:08 PM

Model: PrjClassBuild4

Input Data
Schema: ODM_MTR
Table/View: TRANSFORMATIONSPLIT1TEST

Accuracy: 0.6304956

Confusion Matrix: Rows = Actual; Columns = Predicted

	1	10	2	3	4	5	6	7	8	9
1	2076	0	449	0	0	110	0	0	0	0
10	145	1104	104	70	101	166	258	0	389	253
2	14	56	2421	0	0	85	0	0	0	0
3	0	43	408	2058	122	0	0	0	0	0
4	0	46	0	416	2118	0	0	0	0	0
5	38	25	43	24	0	2446	63	7	0	0
6	88	178	0	0	420	0	1717	188	0	0
7	30	299	41	28	96	293	469	1339	0	0
8	201	442	125	76	92	153	160	405	817	197
9	228	725	168	104	94	157	130	0	618	353

Figure B.5. ABN prediction for downloads. The "download" is predicted based on other project's statistics. Rows contain actual data. Columns contain predicted data. The accuracy of ABN is 63%.

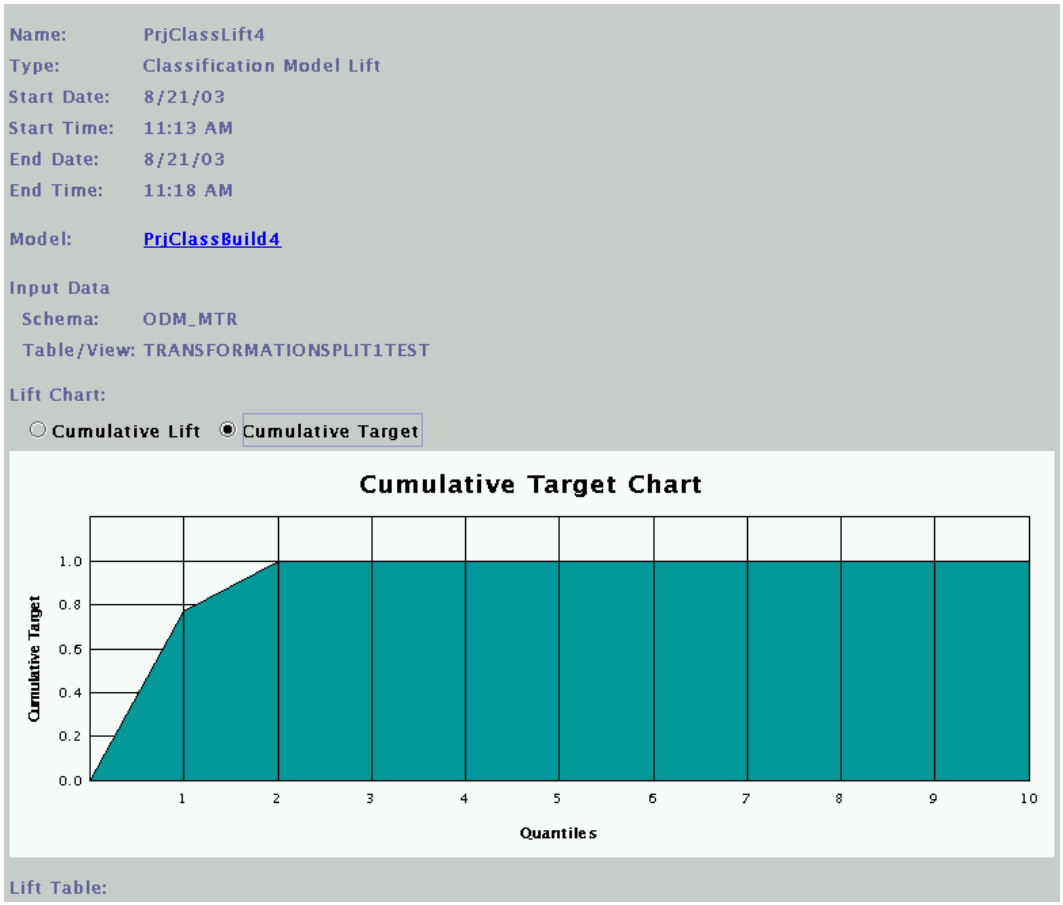


Figure B.6. The Lift chart for ABN prediction. By using ABN algorithm, the cumulative target of "downloads" is 1 in just the first 20% of records, which indicates the prediction is accurate.

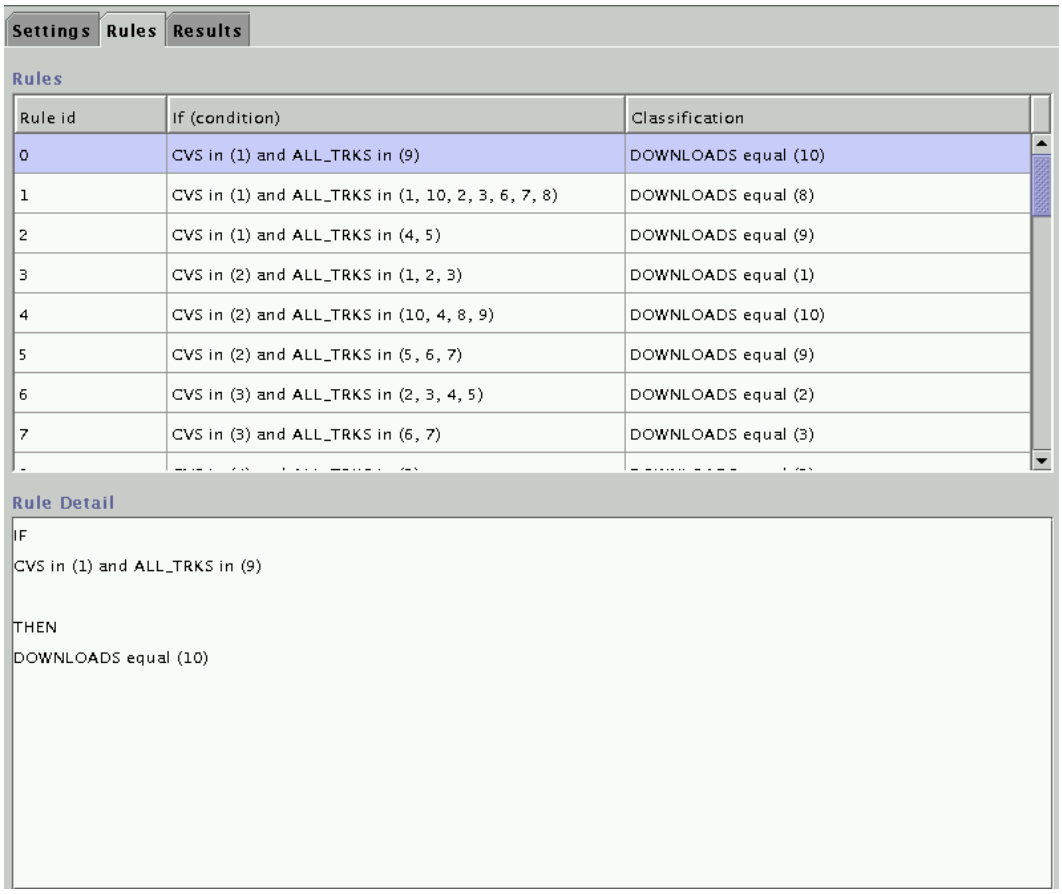


Figure B.7. The rules built by ABN classification. The rule computation contains if-condition field and classification field. "Downloads" is closely related to "cvs".

TABLE B.1

COMPARISON OF ABN AND NAIVE BAYES.

Name	Build Time	Accuracy
ABN	0:19:56	63%
Naive Bayes	0:0:30	9%

TAble notes: ABN takes more time but is more accurate than Naive Bayes algorithm.

support(AB) to support(A) is greater than the minimum confidence.

One well known association rule mining algorithm is called A Priori. Oracle implements this algorithm using PL/SQL. We use the algorithm to find correlations between features of projects. The algorithm takes two input data – the minimum support and the minimum confidence. We choose 0.01 for minimum support and 0.5 for minimum confidence. Figure B.8 shows the results of the association rules mining. From the figure, we see that the feature "all_trks", "cvs" and "downloads" are "associated". More rules can be seen from the above figure. Not all of the rules discovered are of interest.

B.3 Clustering

Clustering is a data analysis which is used to find a collection of similar patterns [64]. We wish to put projects with similar features together to form clusters. Two algorithms can be used to accomplish this: k-means and o-cluster. The k-means algorithm is a distance-based clustering algorithm, which partitions the data into predefined number of clusters. The o-cluster algorithm is a hierar-

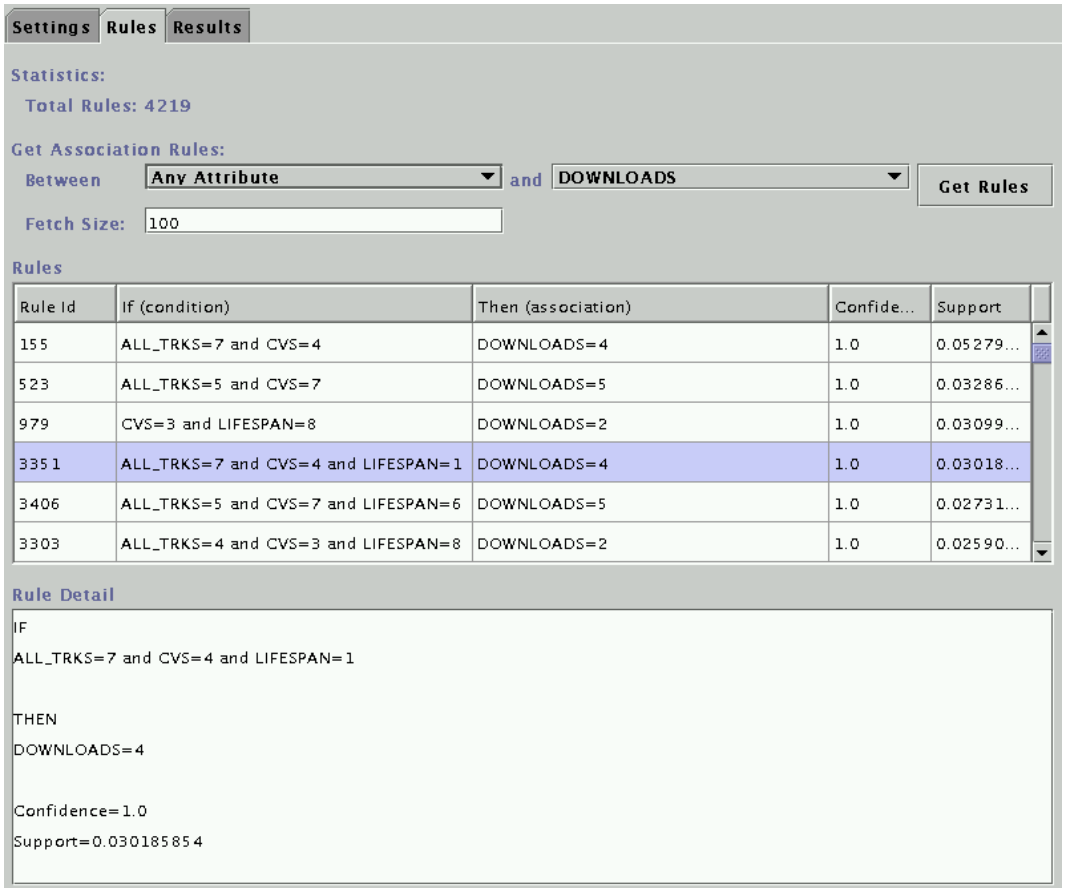


Figure B.8. The rules built by A Priori. Features "all_trks", "cvs" and "downloads" are "associated" because the confidence and support are greater than their thresholds.

chical and grid-based algorithm. The resulting clusters define dense areas in the attribute space. The dimension of the attribute space is the number of attributes involved in the clustering algorithm. We apply the two clustering algorithms to projects in this case study. Figure B.9 and Figure B.10 show the resulting clusters and rules that define the clusters.

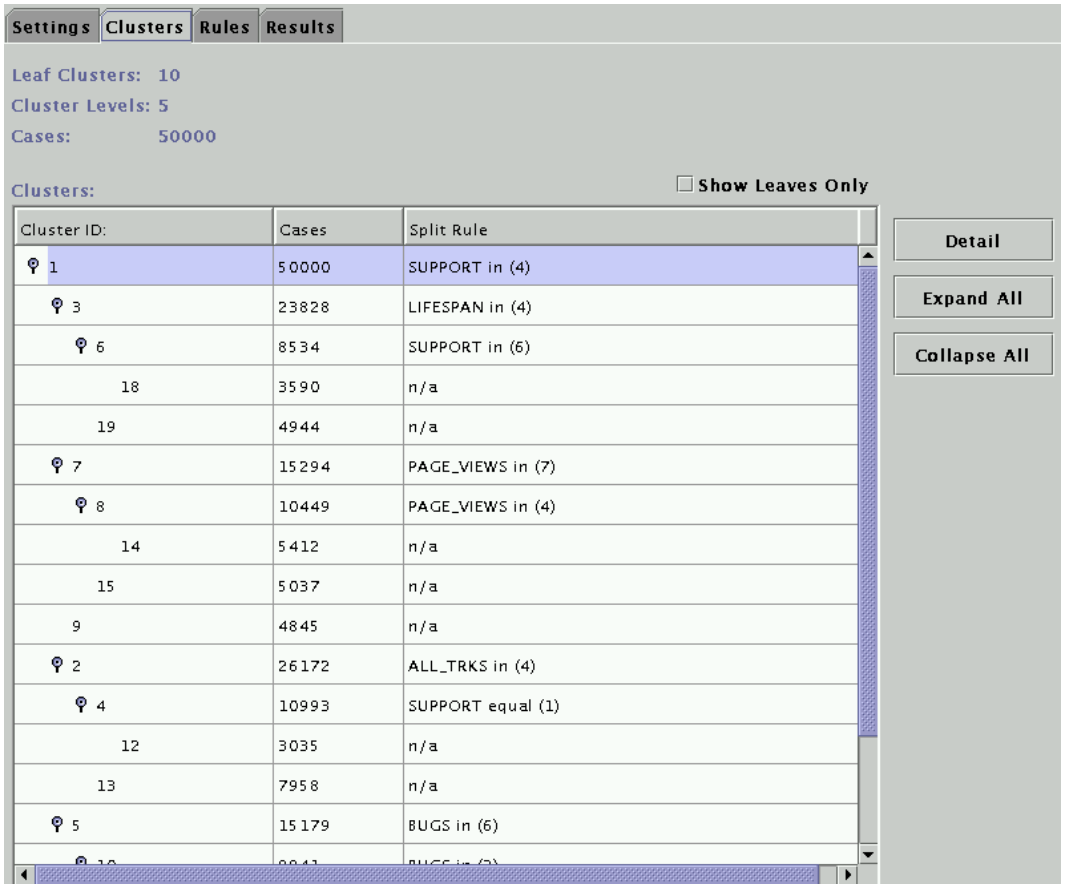


Figure B.9. The clusters. There are total of 50,000 projects. They are divided into 5 groups, each of which contains subgroups. For example, cluster 6, which contains 8534 projects, can be divided into group 18 and group 19.

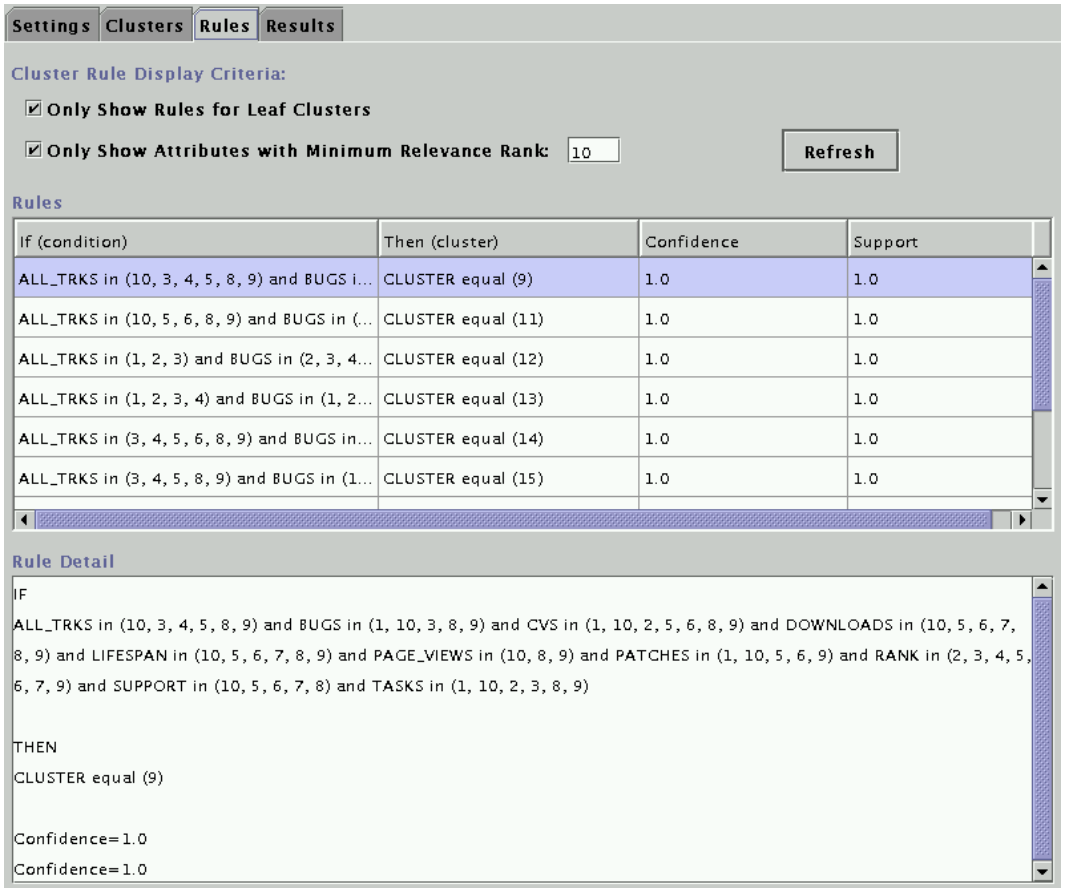


Figure B.10. The rules that define clusters. Statistical features of a project are divided into 10 categories. Clusters are formed according to categories in each statistical feature.

APPENDIX C

ALL SAMPLE PROJECTS

This table is a pool for our sample projects in Chapter2. It contains the top 20% in each topic based on a project's rank on SourceForge. We exclude projects in games, religion, printing, terminals, sociology, formats and protocols.

TABLE C.1

ALL SAMPLE PROJECTS

proj_id	topic	rank	percentile	first_release	developers	control_mark
96728	-1	2872	82.37	20031209	2	0
91681	-1	-1	-100	20031204	4	1
88275	-1	2744	83.16	20031126	4	0
85722	-1	-1	-100	20031129	4	1
93482	-1	74	99.55	20031116	3	0
36400	-1	-1	-100	20031109	1	1
51618	20	1346	91.74	20020917	3	0
5315	20	-1	-100	20020902	1	1
10131	20	1792	89	20030213	8	0
51710	20	12283	24.59	20030228	6	1
9285	20	431	97.36	20040320	6	0
41872	20	-1	-100	20040316	5	1
61518	20	936	94.26	20030207	2	0
239	20	945	94.2	20030210	2	1
32699	20	193	98.82	20010804	1	0
50621	136	2276	86.03	20020507	1	0
33040	136	9740	40.2	20020511	1	1
6241	136	136	99.17	20030425	7	0
62810	136	-1	-100	20030430	5	1

- Topic -1 means non_categoriezed topic.

- Control_mark 0 means effective projects, 1 means size_matched controls.

TABLE C.2

ALL SAMPLE PROJECTS (CONT. 1)

proj_id	topic	rank	percentile	first_release	developers	control_mark
8423	20	7581	53.46	20010802	1	1
118579	20	1094	93.29	20040908	3	0
18103	20	7544	53.69	20040901	1	1
90042	20	130	99.21	20031009	1	0
80880	20	12552	22.94	20031011	3	1
9008	20	471	97.11	20000703	1	0
266	20	-1	-100	20000706	2	1
40359	20	623	96.18	20020101	1	0
30215	20	11556	29.05	20020116	2	1
80227	20	1116	93.15	20030508	2	0
24251	20	3013	81.51	20030504	2	1
66311	20	1421	91.28	20021103	1	0
54748	20	9881	39.34	20021109	2	1
58425	43	1150	92.95	20020904	1	0
40844	43	3322	79.61	20020919	1	1
103893	43	1735	89.35	20040910	1	0
8057	43	8325	48.89	20040907	1	1
15364	45	1165	92.85	20001126	1	0
376	45	-1	-100	20001112	1	1
127198	45	570	96.51	20050103	3	0
5971	45	1079	93.38	20050108	4	1
61771	45	500	96.94	20030904	2	0
19081	45	8672	46.76	20030901	2	1
102352	45	1985	87.82	20040311	7	0
47050	45	15307	6.02	20040302	6	1
69034	45	716	95.61	20030427	1	0
4213	45	5609	65.57	20030430	2	1
107955	45	859	94.73	20040427	1	0
84112	45	-1	-100	20040415	2	1
104240	45	2538	84.42	20050119	1	0

TABLE C.3

ALL SAMPLE PROJECTS (CONT. 2)

proj_id	topic	rank	percentile	first_release	developers	control_mark
8419	45	10087	38.07	20050104	1	1
64773	45	593	96.37	20030922	13	0
55599	45	4529	72.2	20030929	14	1
23523	45	876	94.63	20010329	3	0
1280	45	-1	-100	20010318	2	1
93562	45	525	96.78	20031030	1	0
72773	45	-1	-100	20031026	2	1
75155	45	476	97.08	20030228	1	0
10382	45	-1	-100	20030225	1	1
60894	45	1114	93.17	20030526	5	0
7500	45	3026	81.43	20030529	5	1
6208	45	230	98.59	20020304	8	0
26590	45	2649	83.74	20020305	9	1
44890	45	2414	85.18	20030326	8	0
13005	45	4737	70.92	20030317	6	1
5649	55	1527	90.63	20000523	2	0
3702	55	7000	57.03	20000511	2	1
80679	55	674	95.87	20030519	2	0
63455	55	7072	56.59	20030508	1	1
42641	55	1256	92.29	20011223	1	0
34295	55	-1	-100	20011230	2	1
9366	63	448	97.26	20011217	4	0
28515	63	12819	21.3	20011220	3	1
109512	63	2081	87.23	20040601	1	0
90801	63	-1	-100	20040610	2	1
285	63	2338	85.65	20000818	3	0
9362	63	-1	-100	20000815	3	1
109355	66	299	98.17	20040512	1	0
38943	66	1993	87.77	20040515	2	1
32409	66	2236	86.28	20010927	3	0

TABLE C.4

ALL SAMPLE PROJECTS (CONT. 3)

proj_id	topic	rank	percentile	first_release	developers	control_mark
27910	66	11168	31.44	20010916	2	1
58645	66	412	97.48	20030318	4	0
2770	66	13749	15.59	20030329	2	1
82837	66	458	97.19	20030902	1	0
65753	66	1390	91.47	20030915	1	1
107819	66	62	99.63	20040715	8	0
61349	66	-1	-100	20040729	9	1
15930	71	1609	90.13	20040127	1	0
46679	71	-1	-100	20040116	3	1
101817	71	2861	82.44	20041016	6	0
121556	71	8189	49.73	20041019	8	1
2266	87	1529	90.62	20000207	1	0
525	87	13138	19.34	20000207	1	1
43261	87	870	94.66	20020220	1	0
10662	87	-1	-100	20020225	2	1
88640	87	2207	86.46	20030905	2	0
54718	87	5717	64.9	20030908	3	1
93916	87	1593	90.23	20031108	1	0
66461	87	16134	0.95	20031121	3	1
125039	87	2678	83.56	20041208	4	0
95217	87	7998	50.9	20041213	3	1
70930	87	1551	90.48	20030119	6	0
9295	87	4227	74.05	20030115	7	1
55870	87	1232	92.44	20020806	1	0
8693	87	16213	0.46	20020818	1	1
86242	87	1694	89.61	20031125	1	0
54049	87	-1	-100	20031119	3	1
118524	87	6	99.97	20041118	2	0
5532	87	5248	67.78	20041123	2	1
10064	87	514	96.85	20000819	1	0

TABLE C.5

ALL SAMPLE PROJECTS (CONT. 4)

proj_id	topic	rank	percentile	first_release	developers	control_mark
3813	87	8637	46.98	20000818	3	1
30464	87	2147	86.82	20010710	1	0
12728	87	-1	-100	20010715	1	1
19103	87	2643	83.78	20010527	2	0
5565	87	-1	-100	20010530	1	1
56385	87	2492	84.71	20030813	2	0
26896	87	9659	40.7	20030817	3	1
72483	87	501	96.93	20030327	7	0
65451	87	4854	70.2	20030325	6	1
22615	87	1895	88.37	20030701	2	0
2632	87	3125	80.82	20030713	2	1
55009	87	1675	89.72	20020604	1	0
39571	87	2476	84.8	20020602	2	1
48726	87	287	98.24	20031201	2	0
14045	87	510	96.87	20031208	3	1
40021	97	3200	80.36	20020626	3	0
22200	97	11014	32.38	20020624	4	1
1045	97	1257	92.29	20001007	1	0
9154	97	3849	76.37	20001017	1	1
9736	97	2973	81.75	20000814	1	0
5322	97	7803	52.1	20000815	3	1
43805	97	1771	89.13	20020809	6	0
29600	97	-1	-100	20020806	6	1
5511	97	2479	84.79	20000522	2	0
590	97	11384	30.11	20000511	4	1
29749	97	1475	90.95	20010621	1	0
2379	97	7526	53.8	20010617	2	1
76693	97	291	98.22	20040408	2	0
73698	97	13181	19.08	20040418	1	1
107028	99	1309	91.97	20041104	1	0
75595	99	3297	79.76	20041118	1	1
67586	99	1610	90.12	20021213	2	0

TABLE C.6

ALL SAMPLE PROJECTS (CONT. 5)

proj_id	topic	rank	percentile	first_release	developers	control_mark
1689	99	2254	86.17	20000219	1	0
404	99	8141	50.02	20000229	1	1
114308	129	741	95.46	20040804	2	0
102644	129	6678	59	20040807	1	1
104707	129	1540	90.55	20040322	1	0
86820	129	-1	-100	20040315	1	1
114114	129	1137	93.03	20040831	3	0
52783	129	8038	50.65	20040817	5	1
92412	129	124	99.24	20031015	1	0
90976	129	2298	85.9	20031013	3	1
105292	136	30	99.82	20041031	3	0
89428	136	3059	81.22	20041023	1	1
43021	136	2065	87.33	20020101	1	0
22075	136	-1	-100	20020113	1	1
6212	136	956	94.14	20000529	1	0
5042	136	-1	-100	20000526	3	1
40604	136	204	98.75	20021226	18	0
16307	136	8950	45.05	20021220	16	1
55528	136	2524	84.51	20020616	1	0
19991	136	-1	-100	20020603	1	1
122798	136	2450	84.96	20041129	1	0
117146	136	7125	56.26	20041125	1	1
50973	136	2269	86.07	20020411	1	0
17055	136	5631	65.43	20020419	1	1
93348	136	1181	92.75	20031026	1	0
9756	136	3439	78.89	20031016	2	1
43764	136	2408	85.22	20020110	1	0
17913	136	-1	-100	20020105	2	1
62227	136	528	96.76	20040324	5	0
72096	136	4865	70.14	20040316	6	1
115098	136	981	93.98	20040722	1	0

TABLE C.7

ALL SAMPLE PROJECTS (CONT. 6)

proj_id	topic	rank	percentile	first_release	developers	control_mark
20106	99	5316	67.37	20021210	3	1
75590	99	1868	88.54	20040716	6	0
91457	99	2565	84.26	20040705	6	1
28498	99	1822	88.82	20010601	1	0
7813	99	-1	-100	20010603	3	1
88236	99	2213	86.42	20030823	1	0
75946	99	-1	-100	20030831	1	1
37982	99	1345	91.75	20011120	2	0
5165	99	2241	86.25	20011127	4	1
128244	99	2040	87.48	20050114	2	0
107093	99	-1	-100	20050103	2	1
102676	99	295	98.19	20040420	4	0
42020	99	-1	-100	20040416	4	1
118306	99	159	99.03	20040921	4	0
104310	99	4821	70.41	20040925	3	1
71670	99	1583	90.29	20041206	2	0
64341	99	2063	87.34	20041212	2	1
7118	99	21	99.88	20000627	2	0
3417	99	-1	-100	20000629	1	1
68802	136	5811	64.33	20040726	1	1
121515	136	66	99.6	20041015	2	0
57402	136	2566	84.25	20041029	2	1
5593	136	902	94.47	20000513	2	0
2360	136	-1	-100	20000501	1	1
82533	234	165	98.99	20041115	3	0
102757	234	12148	25.42	20041103	1	1

APPENDIX D

ALL SAMPLE PROJECTS RESPONSE STATISTICS

This table contains all response records we retrieve from SourceForge database. The data are aggregations for messages in artifacts and forums. Column *a*t_noresponse is the number of artifacts which have no responses messages; Column *at_response* is the number of artifacts which have response messages; Column *at_close* is the number of artifacts which are close but have no responses; Column *at_num_replies* is the total number of all response messages of at_response; Column *at_reply_time* is the sum of the first response time of at_response (seconds); Column *forum_noresponse* is the number of forum messages which have no response messages; Column *forum_response* is the number of forum messages which have response messages; Column *forum_num_replies* is the total number of all response messages of forum_response; Column *forum_reply_time* is the sum of the first response time of forum_response (seconds). We describe the data retrival process in Chapter 2. These data are used to support hypotheses discussed in [104].

TABLE D.1

ALL SAMPLE PROJECTS RESPONSE STATISTICS

proj_id	at_noresponse	at_response	at_close	at_num_replies	at_reply_time	forum_noresponse	forum_response	forum_num_replies	forum_reply_time
1042	0	1	0	1	193	2	0	0	0
1068	0	2	5	4	19197926	2	7	26	24839000
2098	0	4	1	5	9338375	2	0	0	0
1949	1	14	0	27	28077532	2	0	0	0
2266	0	2	0	5	26476	6	10	17	60848101
525	14	17	2	34	2024312	5	1	1	89362418
2179	0	0	0	0	0	41	103	359	46697529
2364	0	1	0	1	24114084	4	2	6	7924795
2599	0	25	1	82	2717939	0	0	0	0
1689	0	2	0	5	224521	2	0	0	0
587	0	0	0	0	0	3	1	1	1750642
404	0	1	0	1	64377	2	1	3	40438397
2603	0	1	0	3	3534	8	5	6	3130975
3461	1	4	0	6	240800	2	2	4	76409335
3921	1	0	0	0	0	3	0	0	0
2556	0	4	1	12	4917766	4	1	2	11987014
2752	0	0	0	0	0	2	1	1	4772979
4359	0	111	10	204	57735550	3	1	1	7519009
4261	0	9	0	14	95192863	10	13	35	61737314
2360	0	0	0	0	0	2	0	0	0

TABLE D.2

ALL SAMPLE PROJECTS RESPONSE STATISTICS (CONT. 1)

proj_id	at_noresponse	at_response	at_close	at_num_replies	at_reply_time	forum_noresponse	forum_response	forum_num_replies	forum_reply_time
5093	0	0	0	0	0	2	0	0	0
3702	0	6	1	8	22468919	2	4	7	50118396
590	1	6	0	10	34138775	2	0	0	0
5593	1	4	0	4	210914362	2	3	4	340926
5665	0	17	35	45	28650525	2	0	0	0
5511	1	78	10	118	44394417	15	10	47	413194
5649	9	14	0	22	275209460	4	1	1	3018855
5042	0	2	0	3	35422877	4	0	0	0
5788	0	2	0	5	641	2	1	1	8545418
6212	0	1	3	2	3036	8	15	33	16021192
6907	0	0	0	0	0	3	1	1	30105
1802	1	19	8	29	44847472	0	0	0	0
7147	2	0	0	0	0	4	2	8	36825438
7232	0	0	0	0	0	3	4	11	581891
3914	0	38	0	71	103922322	5	3	16	60942686
7118	0	9	4	12	432278204	2	1	1	90743619
3417	0	1	0	2	391217	3	27	63	95486038
9008	0	0	1	0	0	2	0	0	0
266	1	1	0	1	13798	1	1	1	41562291
1475	0	2	0	3	1506506	2	2	2	56238

TABLE D.3

ALL SAMPLE PROJECTS RESPONSE STATISTICS (CONT. 2)

proj_id	at_noresponse	at_response	at_close	at_num_replies	at_reply_time	forum_noresponse	forum_response	forum_num_replies	forum_reply_time
285	2	75	10	140	164632145	17	29	57	14025413
3813	0	2	0	2	47593665	3	3	3	50650942
10064	0	0	0	0	0	7	3	3	1546115
10350	0	1	0	1	575878	2	0	0	0
9847	0	22	12	41	12760918	3	3	6	333649
11641	2	1	1	1	469651	4	6	8	20701922
1045	2	35	9	79	16487425	1	67	185	68111601
12833	0	0	0	0	0	2	0	0	0
9154	0	0	5	0	0	2	0	0	0
2992	1	4	1	6	9328238	2	0	0	0
13177	0	3	0	5	44996283	4	4	7	233641
13257	0	2	0	2	10887578	3	0	0	0
376	0	2	0	2	91179146	3	2	7	37367792
12992	0	41	4	88	55783383	8	2	23	147828
10489	2	4	1	4	680988	2	4	6	12548372
3311	0	0	0	0	0	4	0	0	0
15364	14	19	48	23	330787748	2	0	0	0
15366	0	0	1	0	0	5	3	6	63633468
14653	4	30	50	86	307894850	19	78	185	155230102
15435	1	24	4	43	4160991	8	5	9	22302650

TABLE D.4

ALL SAMPLE PROJECTS RESPONSE STATISTICS (CONT. 3)

proj_id	at_noresponse	at_response	at_close	at_num_replies	at_reply_time	forum_noresponse	forum_response	forum_num_replies	forum_reply_time
16161	0	25	0	53	38303607	0	0	0	0
11217	0	0	0	0	0	2	2	7	144850
13167	0	1	1	4	3772246	2	0	0	0
11899	0	0	0	0	0	5	0	0	0
12762	0	21	6	36	536301696	15	9	14	79175056
6902	0	0	0	0	0	2	1	1	24709
17809	0	4	0	6	2937358	2	0	0	0
14292	0	0	0	0	0	2	0	0	0
17583	1	5	0	5	25702419	3	2	6	34988308
14990	0	1	0	4	80363	1	1	3	15929333
21109	2	0	0	0	0	2	0	0	0
20729	1	1	0	1	2864963	2	1	3	14214
1280	0	0	0	0	0	5	0	0	0
21786	6	0	0	0	0	2	0	0	0
23523	0	0	0	0	0	9	30	77	14360578
7984	3	3	1	6	167885	2	0	0	0
15633	3	27	10	54	65564416	38	81	205	78518440
9147	0	1	0	1	1815012	2	0	0	0
22965	0	1	0	1	129452	1	1	5	180517
24686	6	55	25	97	166947806	2	0	0	0

TABLE D.5

ALL SAMPLE PROJECTS RESPONSE STATISTICS (CONT. 4)

proj_id	at_noresponse	at_response	at_close	at_num_replies	at_reply_time	forum_noresponse	forum_response	forum_num_replies	forum_reply_time
2463	2	3	2	4	21153660	3	0	0	0
26031	1	47	4	148	17013520	2	20	52	9576504
26907	1	2	0	2	3651672	2	0	0	0
27567	12	19	3	27	243238844	2	0	0	0
19103	4	18	5	24	153625818	7	18	38	7512740
5565	0	0	0	0	0	7	1	1	14011527
28498	0	4	1	21	33470667	2	0	0	0
7813	3	3	0	10	59268	5	4	8	30107132
28870	0	1	1	1	17013181	4	0	0	0
2379	0	8	0	9	30273417	2	0	0	0
29749	0	0	0	0	0	4	0	0	0
30151	6	0	2	0	0	2	0	0	0
11280	1	8	1	10	2347188	2	0	0	0
30464	0	7	1	15	8155458	1	7	13	12008481
12728	2	0	0	0	0	3	0	0	0
30841	0	0	0	0	0	2	0	0	0
621	11	114	14	226	376411133	2	0	0	0
2739	2	0	0	0	0	3	3	10	2468101
12907	0	1	4	2	0	1	2	3	770112
8423	3	4	0	4	40236957	2	0	0	0

TABLE D.6

ALL SAMPLE PROJECTS RESPONSE STATISTICS (CONT. 5)

proj_id	at_noresponse	at_response	at_close	at_num_replies	at_reply_time	forum_noresponse	forum_response	forum_num_replies	forum_reply_time
27386	0	0	0	0	0	2	0	0	0
33717	1	89	9	132	435637707	1	0	0	0
34097	19	83	0	122	342866191	2	0	0	0
34659	1	0	0	0	0	2	0	0	0
35951	1	4	0	7	1154515	0	0	0	0
16200	0	0	0	0	0	1	2	3	91211093
27910	0	1	0	1	13942964	0	0	0	0
29057	2	484	21	1378	627685924	33	440	1316	96000322
29449	2	59	13	95	98177697	3	0	0	0
32409	1	13	35	19	47127646	3	16	26	11810254
19995	1	3	0	3	5352882	2	0	0	0
16636	5	411	14	886	360885312	14	193	447	68169198
28148	7	7	12	7	14188114	4	1	1	32
37698	0	0	0	0	0	1	2	3	11038381
21292	0	2	1	2	4450969	3	0	0	0
36022	0	9	0	14	832288	2	0	0	0
35585	10	72	47	97	389208086	10	6	18	1429646
27130	0	0	0	0	0	5	2	3	359718
39203	1	1	0	2	40600947	2	0	0	0
31279	1	4	1	6	1298264	2	1	1	24769700

TABLE D.7

ALL SAMPLE PROJECTS RESPONSE STATISTICS (CONT. 6)

proj_id	at_noresponse	at_response	at_close	at_num_replies	at_reply_time	forum_noresponse	forum_response	forum_num_replies	forum_reply_time
17513	0	6	1	8	104960252	0	0	0	0
35309	0	2	10	5	4300801	2	1	2	14341
21640	9	10	9	12	18578525	2	0	0	0
38515	0	0	1	0	0	4	1	1	3530451
37700	1	1	0	1	149079	2	1	1	2324865
3458	1	78	8	152	244264569	166	498	2173	49760586
21351	0	0	0	0	0	2	0	0	0
9366	0	121	1	202	353693815	2	0	0	0
28515	0	0	0	0	0	0	5	7	155465765
42641	0	3	0	5	1938287	3	3	22	3695124
21193	0	3	0	3	6965724	1	0	0	0
33758	0	9	5	16	25732435	20	91	224	63515864
34041	0	11	28	20	3104130	2	0	0	0
15581	0	3	0	3	304159	5	4	8	1397298
44002	0	1	1	1	48701385	1	1	1	7936
45605	0	8	0	18	51423617	15	52	145	94510359
1866	0	1	1	1	2306	12	0	0	0
41738	3	18	0	39	79888537	3	5	30	27921816
14163	0	0	0	0	0	2	1	3	40773542
30794	0	0	0	0	0	5	0	0	0

TABLE D.8

ALL SAMPLE PROJECTS RESPONSE STATISTICS (CONT. 7)

proj_id	at_noresponse	at_response	at_close	at_num_replies	at_reply_time	forum_noresponse	forum_response	forum_num_replies	forum_reply_time
17726	0	11	10	20	35142739	5	10	17	24899986
17723	0	3	0	5	57958086	2	0	0	0
49180	0	2	0	2	13723	3	1	1	341667
46038	2	85	4	144	224014165	15	100	257	60685482
39716	1	12	8	17	172167321	8	65	276	14661020
50071	0	3	1	4	55428929	25	69	209	159172286
249	5	8	1	10	22703237	2	0	0	0
50272	0	6	0	7	279949391	4	10	22	58373434
52204	0	4	0	6	2860890	2	0	0	0
38364	1	3	0	4	31031111	14	3	3	11814343
6473	17	269	15	582	791235895	2	0	0	0
5506	3	11	1	20	202356939	2	0	0	0
2725	0	3	3	4	10690564	3	0	0	0
52330	0	0	1	0	0	5	1	1	71300
51673	1	2	0	9	6788477	4	0	0	0
50906	7	0	0	0	0	4	0	0	0
53177	6	17	2	33	14798519	30	56	229	106091142
29145	18	8	11	9	1692372	0	0	0	0
6691	0	0	0	0	0	2	0	0	0
53733	1	6	1	9	136596081	4	1	3	766942

TABLE D.9

ALL SAMPLE PROJECTS RESPONSE STATISTICS (CONT. 8)

proj_id	at_noresponse	at_response	at_close	at_num_replies	at_reply_time	forum_noresponse	forum_response	forum_num_replies	forum_reply_time
17676	3	1	2	1	23082909	2	0	0	0
40468	0	0	0	0	0	2	1	1	373916
54021	0	49	2	130	8849809	2	0	0	0
56980	5	5	4	11	44440979	3	2	2	817736
54192	8	45	31	92	129061850	2	0	0	0
57796	0	80	8	182	248026849	19	61	247	44268249
23214	13	11	0	15	147181381	3	2	2	69675409
40694	2	137	7	322	602970337	2	0	0	0
59277	4	0	3	0	0	11	20	54	2749530
55870	0	1	2	1	57461	2	15	39	109378288
29600	0	1	0	2	29947322	7	3	9	1988
43805	1	1	2	2	881	0	4	10	42975837
15463	4	22	0	48	58050346	13	82	193	105530389
60358	27	17	1	20	80250730	2	0	0	0
1316	2	208	48	299	718748648	2	1	2	1240382
17916	0	0	0	0	0	2	0	0	0
56631	0	0	1	0	0	1	4	4	52058
37880	9	13	1	17	129373042	1	2	2	41477713
15142	0	0	0	0	0	8	1	1	308
47577	0	7	0	14	10539503	1	0	0	0

TABLE D.10

ALL SAMPLE PROJECTS RESPONSE STATISTICS (CONT. 9)

proj_id	at_noresponse	at_response	at_close	at_num_replies	at_reply_time	forum_noresponse	forum_response	forum_num_replies	forum_reply_time
521	0	1	0	1	13385568	2	0	0	0
59339	0	0	0	0	0	0	0	0	0
63470	8	89	6	216	852564040	2	0	0	0
54014	0	2	1	2	1910930	8	0	0	0
63121	1	5	1	6	6048512	1	12	32	9881690
17176	0	5	5	7	742	2	1	1	3733078
11522	3	0	0	0	0	2	0	0	0
54948	1	10	7	32	81793934	3	5	13	24548877
39708	0	9	0	10	38511647	1	0	0	0
42816	0	0	0	0	0	3	1	1	1497
62270	1	365	41	827	199949898	2	0	0	0
47067	0	0	5	0	0	3	3	8	158907
19383	17	159	61	266	335766554	2	0	0	0
16021	1	0	0	0	0	6	1	1	649774
66311	0	9	5	13	29816850	2	0	0	0
62103	11	13	1	28	98464935	4	7	13	30930058
1132	0	0	0	0	0	2	2	3	1320216
65881	5	2	1	2	2330296	2	0	0	0
11986	2	1	0	1	96826	2	0	0	0
50069	0	1	0	3	3660085	7	8	23	34146130

TABLE D.11

ALL SAMPLE PROJECTS RESPONSE STATISTICS (CONT. 10)

proj_id	at_noresponse	at_response	at_close	at_num_replies	at_reply_time	forum_noresponse	forum_response	forum_num_replies	forum_reply_time
67940	0	0	0	0	0	3	9	31	15400666
64331	0	0	0	0	0	2	0	0	0
47722	3	21	15	33	28498920	2	1	1	1081957
41542	0	9	0	31	34053434	17	40	97	34164035
68207	0	1	2	2	257844	16	46	131	53357782
1950	0	0	0	0	0	2	0	0	0
68502	0	7	1	11	41778367	7	6	30	28724412
8381	6	3	0	5	370973	2	2	2	123984828
68316	0	16	0	87	29402333	2	0	0	0
69146	0	3	0	7	37192122	4	0	0	0
64823	0	0	0	0	0	2	0	0	0
40604	18	151	7	262	690238265	0	0	0	0
70373	0	20	3	26	4295734	3	7	21	117504
69007	1	1	1	1	496814	4	5	22	395755
2659	2	2	0	2	1867	2	0	0	0
70090	3	0	0	0	0	2	0	0	0
25960	0	31	12	51	42882963	51	84	280	98069226
71067	1	0	4	0	0	2	0	0	0
55463	4	40	3	74	207764414	5	13	28	15806331
71174	5	5	0	11	14843695	4	6	13	1272991

TABLE D.12

ALL SAMPLE PROJECTS RESPONSE STATISTICS (CONT. 11)

proj_id	at_noresponse	at_response	at_close	at_num_replies	at_reply_time	forum_noresponse	forum_response	forum_num_replies	forum_reply_time
71616	1	22	0	28	118508721	1	0	0	0
66479	11	65	16	117	341915448	2	0	0	0
25227	23	58	4	73	634017957	2	0	0	0
70031	4	0	0	0	0	2	0	0	0
72099	8	30	0	63	188446231	2	0	0	0
49820	0	5	0	7	30559782	3	3	3	471285
71084	0	5	0	11	24392990	14	86	362	32367558
40657	0	0	0	0	0	20	66	175	349370649
7402	0	0	0	0	0	38	71	193	216800191
73406	4	10	1	13	2621135	5	6	22	147882
16520	0	0	3	0	0	1	1	1	68856220
11495	3	6	0	11	1026838	10	18	34	375561430
74668	2	0	0	0	0	2	1	1	2155450
74552	2	0	0	0	0	2	2	2	13115114
74852	4	1	8	1	592	2	0	0	0
74709	4	51	29	96	224584786	14	48	108	65768141
74573	14	23	15	32	160265461	48	106	392	47821804
74775	0	1	0	6	11192830	2	0	0	0
24819	0	1	1	2	552	3	9	22	55395272
75155	0	9	1	25	7175481	2	11	16	1749170

TABLE D.13

ALL SAMPLE PROJECTS RESPONSE STATISTICS (CONT. 12)

proj_id	at_noresponse	at_response	at_close	at_num_replies	at_reply_time	forum_noresponse	forum_response	forum_num_replies	forum_reply_time
8157	1	8	3	13	10832821	2	5	10	13595562
265	0	0	1	0	0	1	0	0	0
906	0	0	0	0	0	4	0	0	0
8956	0	25	0	35	184084729	2	0	0	0
8983	0	1	0	1	114832	2	1	1	84463
9164	0	1	0	2	2166125	2	1	5	54195
9736	0	2	0	3	5085	2	0	0	0
1737	0	3	0	3	7241300	2	0	0	0
9362	0	2	1	3	16202248	5	2	5	1107984
5322	0	1	0	2	2920	4	1	1	120390
2939	0	5	0	5	12120781	2	0	0	0
3130	19	17	0	27	30644402	23	45	117	50140036
14312	1	7	1	8	1094514	2	0	0	0
5140	0	4	1	8	11977925	4	1	2	1382865
3301	11	14	21	29	194718166	23	21	41	76437539
308	5	127	76	216	155275283	36	47	286	137216514
2238	1	1	0	1	15811	2	0	0	0
16118	3	2	6	3	1308004	2	0	0	0
25664	1	1	2	1	392	2	0	0	0
24038	0	3	9	4	97121	4	0	0	0

TABLE D.14

ALL SAMPLE PROJECTS RESPONSE STATISTICS (CONT. 13)

proj_id	at_noresponse	at_response	at_close	at_num_replies	at_reply_time	forum_noresponse	forum_response	forum_num_replies	forum_reply_time
32699	0	16	1	36	46265043	3	13	38	72484724
16757	2	29	1	39	299097846	3	0	0	0
26165	1	0	8	0	0	2	2	3	35
32880	1	1	0	1	19264	2	0	0	0
5733	4	10	1	16	21985665	6	0	0	0
726	1	0	0	0	0	3	0	0	0
39518	8	26	32	40	47209642	4	2	4	379272
28760	1	2	2	2	192889	3	0	0	0
25451	2	0	16	0	0	3	0	0	0
40205	0	24	4	53	9804293	6	9	30	404087
49087	1	1	0	1	13392917	2	1	2	4070453
9858	6	73	5	101	373127205	5	51	133	46613827
702	0	0	0	0	0	18	22	45	169314927
43989	10	6	1	9	1005991	2	1	1	68421500
12937	0	0	0	0	0	2	2	2	2633011
54559	5	46	1	90	106079002	0	0	0	0
25225	0	4	1	5	1340796	2	0	0	0
36952	0	1	0	1	313240	3	0	0	0
52647	4	22	5	40	270275640	0	0	0	0
56126	0	1	2	1	378170	9	20	56	17970459

TABLE D.15

ALL SAMPLE PROJECTS RESPONSE STATISTICS (CONT. 14)

proj_id	at_noresponse	at_response	at_close	at_num_replies	at_reply_time	forum_noresponse	forum_response	forum_num_replies	forum_reply_time
75170	0	5	9	5	739873	2	3	10	447029
75226	0	1	0	1	441195	1	0	0	0
75126	0	1	0	2	17098789	2	0	0	0
7746	1	63	21	135	195628440	8	27	96	42753342
42598	0	1	1	1	13802912	2	0	0	0
27255	0	4	0	5	76664210	26	81	308	103740328
29196	4	13	0	23	102479441	2	0	0	0
59041	3	1	0	1	2652	2	0	0	0
4220	0	1	0	1	13120169	3	2	3	34512068
67682	2	3	0	7	303909	4	9	19	21368501
66829	0	0	0	0	0	2	2	3	186590
67220	0	0	0	0	0	2	1	1	165431
56780	15	8	5	18	63185342	22	44	96	63593761
29393	1	16	2	33	4286525	2	0	0	0
34430	0	58	2	133	123489733	22	74	252	28326453
48067	2	2	0	2	85296168	2	0	0	0
31885	60	154	29	307	969877834	81	96	162	234591062
11257	2	1	0	1	1188657	5	0	0	0

APPENDIX E

ALL SAMPLE PROJECTS RESPONSE PROGRAM

This is the program we use to extract response information. This program retrieves information such as response time, the number of responses, messages without responses from SourceForge database. Results from this information are reported in [104].

```
import java.sql.*;
import java.io.*;
import java.util.regex.*;
import java.util.*;

public class response
{
    public static void main (String args[])
throws ClassNotFoundException, SQLException,  FileNotFoundException, \
IOException
     {
String driver = "org.postgresql.Driver";
String url_in = "jdbc:postgresql://zerlot.cse.nd.edu:5555/timeline";

String url_out = "jdbc:postgresql:rating";

String user = "jxu1";
String pwd = "jxu1";

String q;
ResultSet rs;

try {
```

```
     Class.forName(driver);
}catch (ClassNotFoundException e){
     System.err.println("Can't load driver" + e.getMessage());
     System.exit(1);
}

Connection conn_in = DriverManager.getConnection \
(url_in, "postgres", "postgres");

        Connection conn_out = DriverManager.getConnection\
(url_out, user, pwd);

PrintWriter out = new PrintWriter(new FileOutputStream\
("/tmp/response.txt"));

//q = "select distinct group_id from groups";
q = "select distinct proj_id from final_projects_5";
rs = response.selection(conn_out, q);
if (!rs.next()){
    System.out.println(" not found");
}else{
    do{
int proj_id = rs.getInt(1);

//# of messages without response
int at_count_noresponse = 0;
//# of messages with response and open
int at_count_response = 0;
//# of messages without response but closed
int at_count_close = 0;
int at_num_response = 0; //# of responses
//total time of first_response
long at_time_diff = 0;
int forum_count_nonresponse = 0;
//# of messages with response
int forum_count_response = 0;
//# of responses
int forum_num_response = 0;
//total time of first_response
long forum_time_diff = 0;
System.out.println(proj_id);
//System.out.println();
```

```
//# of artifact
q = "select a.artifact_id, a.open_date from \
sf0205.artifact a, sf0205.artifact_group_list b \
where  a.group_artifact_id = b.group_artifact_id  \
and b.is_public = 1 /
and b.group_id = " + proj_id;
ResultSet rs1 = response.selection(conn_in, q);
//System.out.println(q);

if (!rs1.next()){
    System.out.println(" not found");
}else{
    do{
int atid = rs1.getInt(1);
long open_date = rs1.getLong(2);
long first_response_date = 0;
q = "select count(id) \
from sf0205.artifact_message \
where artifact_id = " + atid;


ResultSet rs2 = response.selection(conn_in, q);
if (!rs2.next()){
    System.out.println(" not found");
}else{
    int count = rs2.getInt(1);
    if(count == 0){
q = "select close_date \
from sf0205.artifact \
where artifact_id = " + atid;
ResultSet rs4 =  response.selection \
(conn_in, q);
if(!rs4.next()){
    System.out.println(" not found");

}else{
    first_response_date = rs4.getLong(1);
    if(first_response_date == 0){
at_count_noresponse++;

    }else{
```

```
at_count_close++;
    }
}

    }else{
at_count_response++;
at_num_response = at_num_response + count;

q = "select min(adddate) \
from sf0205.artifact_message \
where artifact_id = " + atid  ;
ResultSet rs3 =  response.selection \
(conn_in, q);
if (!rs3.next()){
    System.out.println(" not found");
}else{

    first_response_date = rs3.getLong(1);
}

    }
}
    }while(rs1.next());
}

q = "select count(a.thread_id)\
 from sf0205.forum_threadinfo a, sf0205.forum_group_list b \
 where a.group_forum_id = b.group_forum_id \
 and b.is_public = 1 \
     and a.num_replies = 0 \
 and b.group_id = " + proj_id;

rs1 = response.selection(conn_in, q);
if(!rs1.next()){
    System.out.println(" not found");
}else{
    forum_count_nonresponse = rs1.getInt(1);
}

q = "select a.thread_id, a.num_replies, a.msg_id from \
sf0205.forum_threadinfo a, sf0205.forum_group_list b \
where a.group_forum_id = b.group_forum_id \
```

```
and b.is_public = 1 \
and a.num_replies != 0 \
and b.group_id = " + proj_id;
rs1 = response.selection(conn_in, q);
if(!rs1.next()){
    System.out.println(" not found");
}else{
    do{
int open_date = 0;
forum_count_response ++;
int thread_id = rs1.getInt(1);
int num_replies = rs1.getInt(2);
int msg_id = rs1.getInt(3);
forum_num_response = forum_num_response + num_replies;
q = "select date from sf0205.forum \
where msg_id = " + msg_id;
ResultSet rs2 =  response.selection(conn_in, q);
if(!rs2.next()){
    System.out.println(" not found");
}else{
    open_date = rs2.getInt(1);
}
q = "select min(date) from sf0205.forum \
where is_followup_to = " + msg_id + \
" and thread_id = " + thread_id;
rs2 =  response.selection(conn_in, q);
if(!rs2.next()){
    System.out.println(" not found");
}else{
    int first_response_date = rs2.getInt(1);
    forum_time_diff += first_response_date - open_date;
}
    }while(rs1.next());

}


out.println(proj_id + " " + at_count_noresponse + " " \
+ at_count_response+" " + at_count_close + " " + \
at_num_response + " " + at_time_diff + " "+ \
forum_count_nonresponse + " " + forum_count_response+\
" " + forum_num_response + " " + forum_time_diff);
```

```
    }while(rs.next());
}
conn_in.close();
conn_out.close();
out.close();
    }
    private static ResultSet selection \
(Connection conn, String response) throws SQLException{
PreparedStatement stmt = conn.prepareStatement (response);
stmt.execute();
ResultSet rset = stmt.getResultSet();
return rset;
     }

     private static  void execute  \
(Connection conn, String response) throws SQLException{
PreparedStatement stmt = conn.prepareStatement (response);
stmt.execute();
      }
}
```

APPENDIX F

ALL TABLES USED FROM SOURCEFORGE DATABASE

The following are all tables used in this dissertation. They are from SourceForge data dump.

Table: artifact

Column	Type	Null?
artifact_id	integer	not null
group_artifact_id	integer	not null
status_id	integer	not null
category_id	integer	not null
artifact_group_id	integer	not null
resolution_id	integer	not null
priority	integer	not null
submitted_by	integer	not null
assigned_to	integer	not null
open_date	integer	not null
close_date	integer	not null
summary	text	not null
details	text	not null
closed_by	integer	

artifact_category

Column	Type	Null?
id	integer	not null
group_artifact_id	integer	not null
category_name	text	not null
auto_assign_to	integer	not null

```
Table artifact_counts_agg
      Column       |  Type   | Null?
-------------------+---------+----------
 group_artifact_id | integer | not null
 count             | integer | not null
 open_count        | integer |

Table artifact_file
    Column    |  Type   | Null?
--------------+---------+----------
 id           | integer | not null
 artifact_id  | integer | not null
 description  | text    | not null
 bin_data     | text    | not null
 filename     | text    | not null
 filesize     | integer | not null
 filetype     | text    | not null
 adddate      | integer | not null
 submitted_by | integer | not null

Table artifact_group
      Column       |  Type   | Null?
-------------------+---------+----------
 id                | integer | not null
 group_artifact_id | integer | not null
 group_name        | text    | not null

Table artifact_group_list
       Column        |  Type   | Null?
---------------------+---------+----------
 group_artifact_id   | integer | not null
 group_id            | integer | not null
 name                | text    |
 description         | text    |
 is_public           | integer | not null
 allow_anon          | integer | not null
 email_all_updates   | integer | not null
 due_period          | integer | not null
 use_resolution      | integer | not null
 submit_instructions | text    |
 browse_instructions | text    |
 datatype            | integer | not null
```

```
 status_timeout      | integer |
 due_period_initial  | integer | not null
 due_period_update   | integer | not null

Table SCHEMA.artifact_message
    Column    |  Type   | Null?
--------------+---------+----------
 id           | integer | not null
 artifact_id  | integer | not null
 submitted_by | integer | not null
 adddate      | integer | not null
 body         | text    | not null

Table forum
      Column      |  Type   | Null?
------------------+---------+----------
 msg_id           | integer | not null
 group_forum_id   | integer | not null
 posted_by        | integer | not null
 subject          | text    | not null
 body             | text    | not null
 date             | integer | not null
 is_followup_to   | integer | not null
 thread_id        | integer | not null
 has_followups    | integer |
 most_recent_date | integer | not null
 is_deleted       | integer |

Table forum_agg_msg_count
     Column     |  Type   | Null?
----------------+---------+----------
 group_forum_id | integer | not null
 count          | integer | not null

Table forum_group_list
     Column      |  Type   | Null?
-----------------+---------+----------
 group_forum_id  | integer | not null
 group_id        | integer | not null
 forum_name      | text    | not null
 is_public       | integer | not null
 description     | text    |
```

```
 allow_anonymous | integer | not null

Table forum_threadinfo
         Column          |  Type   | Null?
-------------------------+---------+----------
 threadinfo_id           | integer | not null
 thread_id               | integer | not null
 thread_topic            | text    | not null
 thread_starter_user_id  | integer | not null
 num_replies             | integer |
 most_recent_post_date   | integer |
 group_forum_id          | integer | not null
 msg_id                  | integer | not null

Table frs_file
    Column    |         Type          | Null?
--------------+-----------------------+----------
 file_id      | integer               | not null
 filename     | text                  |
 release_id   | integer               | not null
 type_id      | integer               | not null
 processor_id | integer               | not null
 release_time | integer               | not null
 file_size    | integer               | not null
 post_date    | integer               | not null
 group_id     | integer               |
 package_id   | integer               |
 md5sum       | character varying(32) |

Table frs_package
   Column   |  Type   | Null?
------------+---------+----------
 package_id | integer | not null
 group_id   | integer | not null
 name       | text    |
 status_id  | integer | not null

Table frs_release
    Column    |  Type   | Null?
--------------+---------+----------
 release_id   | integer | not null
 package_id   | integer | not null
```

```
 name         | text    |
 notes        | text    |
 changes      | text    |
 status_id    | integer | not null
 preformatted | integer | not null
 release_date | integer | not null
 released_by  | integer | not null

Table groups
          Column           |          Type          | Null?
---------------------------+------------------------+----------
 group_id                  | integer                | not null
 group_name                | character varying(40)  |
 homepage                  | character varying(128) |
 is_public                 | integer                | not null
 status                    | character(1)           | not null
 unix_group_name           | character varying(30)  | not null
 http_domain               | character varying(80)  |
 short_description         | character varying(255) |
 license                   | character varying(16)  |
 register_time             | integer                | not null
 use_mail                  | integer                | not null
 use_forum                 | integer                | not null
 use_pm                    | integer                | not null
 use_cvs                   | integer                | not null
 use_news                  | integer                | not null
 preferred_support_type    | integer                | not null
 preferred_support_resource | text                  | not null
 type                      | integer                | not null
 use_docman                | integer                | not null
 not_open_source           | integer                | not null
 send_all_tasks            | integer                | not null
 use_pm_depend_box         | integer                | not null
 potm                      | integer                |
 donation_request          | text                   |
 donate_optin              | integer                |
 big_mirror                | integer                |
 project_submitter         | integer                |
 row_modtime               | integer                |

Table groups_historical
  Column  |  Type   | Null?
```

```
----------+---------+----------
 group_id | integer | not null
 date     | integer | not null
 added_by | integer | not null
 status   | integer | not null
 url      | text    |


Table SCHEMA.project_group_list
      Column      |  Type   | Null?
------------------+---------+----------
 group_project_id | integer | not null
 group_id         | integer | not null
 project_name     | text    | not null
 is_public        | integer | not null
 description      | text    |

Table stats_project_all
      Column      |       Type       | Null?
------------------+------------------+----------
 group_id         | integer          |
 developers       | integer          |
 group_ranking    | integer          |
 group_metric     | double precision |
 logo_showings    | integer          |
 downloads        | integer          |
 site_views       | integer          |
 subdomain_views  | integer          |
 page_views       | integer          |
 msg_posted       | integer          |
 msg_uniq_auth    | integer          |
 bugs_opened      | integer          |
 bugs_closed      | integer          |
 support_opened   | integer          |
 support_closed   | integer          |
 patches_opened   | integer          |
 patches_closed   | integer          |
 artifacts_opened | integer          |
 artifacts_closed | integer          |
 tasks_opened     | integer          |
 tasks_closed     | integer          |
 help_requests    | integer          |
```

```
 cvs_checkouts    | integer          |
 cvs_commits      | integer          |
 cvs_adds         | integer          |

Table top_group
          Column          |         Type          | Null?
--------------------------+-----------------------+----------
 group_id                 | integer               | not null
 group_name               | character varying(40) |
 downloads_all            | integer               | not null
 rank_downloads_all       | integer               | not null
 rank_downloads_all_old   | integer               | not null
 downloads_week           | integer               | not null
 rank_downloads_week      | integer               | not null
 rank_downloads_week_old  | integer               | not null
 userrank                 | integer               | not null
 rank_userrank            | integer               | not null
 rank_userrank_old        | integer               | not null
 forumposts_week          | integer               | not null
 rank_forumposts_week     | integer               | not null
 rank_forumposts_week_old | integer               | not null
 pageviews_proj           | integer               | not null
 rank_pageviews_proj      | integer               | not null
 rank_pageviews_proj_old  | integer               | not null

Table trove_agg
      Column       |          Type          | Null?
-------------------+------------------------+----------
 trove_cat_id      | integer                |
 group_id          | integer                |
 group_name        | character varying(40)  |
 unix_group_name   | character varying(30)  |
 status            | character(1)           |
 register_time     | integer                |
 short_description | character varying(255) |
 donate_optin      | integer                |
 percentile        | double precision       |
 ranking           | integer                |

Table trove_agg_counts
    Column    |  Type   | Null?
--------------+---------+----------
```

```
 trove_cat_id | integer | not null
 group_count  | integer |
```

Newly added, Aug. 2005 Table "trove_agg_minix"

Table trove_cat

Column	Type	Null?
trove_cat_id	integer	not null
version	integer	not null
parent	integer	not null
root_parent	integer	not null
shortname	character varying(80)	
fullname	character varying(80)	
description	character varying(255)	
fullpath	text	not null
fullpath_ids	text	
parent_only	integer	not null

Table trove_group_link

Column	Type	Null?
trove_group_id	integer	not null
trove_cat_id	integer	not null
trove_cat_version	integer	not null
group_id	integer	not null
trove_cat_root	integer	not null

Table user_group

Column	Type	Null?
user_group_id	integer	not null
user_id	integer	not null
group_id	integer	not null
admin_flags	character(16)	not null
forum_flags	integer	not null
project_flags	integer	not null
doc_flags	integer	not null
member_role	integer	not null
release_flags	integer	not null
artifact_flags	integer	
added_by	integer	not null

```
 grantcvs       | integer       | not null
 grantshell     | integer       | not null
 row_modtime    | integer       |

Table users
        Column         |          Type          | Null?
-----------------------+------------------------+----------
 user_id               | integer                | not null
 user_name             | text                   | not null
 realname              | character varying(32)  | not null
 status                | character(1)           | not null
 unix_uid              | integer                |
 add_date              | integer                | not null
 people_resume         | text                   | not null
 timezone              | character varying(64)  |
 language              | integer                | not null
 cf_uid                | integer                |
 stay_anon             | integer                |
 donation_request      | text                   |
 donate_optin          | integer                |
 last_sitestatus_view  | integer                |
 row_modtime           | integer                |
```

BIBLIOGRAPHY

1. R. Albert and A.-L. Barabasi. Statistical mechanics of complex networks. *Review of Modern Physics*, v. 74:47–97, 2002.

2. R. Albert, H. Jeong, and A. L. Barabasi. Diameter of the world wide web. *Nature*, v. 401:130–131, 1999.

3. M. J. Ashworth and M. A. Louie. Alignment of the garbage can and NK fitness models: A virtual experiment in the simulation of organizations. http://www.casos.ece.cmu.edu/casos_working_paper/GCandNK_Alignment_abstract.html, 2002.

4. R. Axelrod. Advancing the art of simulation in the social sciences. *In Simulating Social Phenomena*, pages 21–40, 1997.

5. R. Axtell, R. Axelrod, J. Epstein, and M. Cohen. Aligning simulation models: A case study and results. *Computational and Mathematical Organization Theory*, 1(2):123–141, 1996.

6. J. Balthrop, S. Forrest, M. Newman, and M. Williamson. Technological networks and the spread of computer viruses. *Science*, v. 304:527–529, 2004.

7. A. L. Barabasi. *Linked: The New Science of Networks*. Perseus, Boston, 2002.

8. A. L. Barabasi and R Albert. Emergence of scaling in random networks. *Science*, v. 286:509–512, 1999.

9. A. L. Barabasi, R. Albert, and H. Jeong. Scale-free characteristics of random networks: The topology of the world wide web. *Physics A*, pages 69–77, 2000.

10. A. L. Barabasi, H. Jeong, Z. Neda, E. Ravasz, A. Schubert, and T. Viscek. Evolution of the social network of scientific collaborations. *Physics A*, 311:590–614, 2001.

11. Jurgen Bitzer. Intrinsic motivation in open source software development. http://opensource.mit.edu/papers/bitzerschrettlschroder.pdf, 2004.

12. D. Bollier. The power of openness: Why citizens, education, government and business should care about the coming revolution in open source code software. http://eon.law.harvard.edy/opencode/h20, 1999.

13. R.S Burt. *Structural holes: The social structure of competition*. Harvard University Press, Cambridge, MA, 1992.

14. R. Burton. *Simulating Organizations: Computational Models of Institutions and Groups*, chapter Aligning Simulation Models: A Case Study and Results. AAAI/MIT Press, Cambridge, Massachusetts, 1998.

15. S. Christley. Understanding the open source software community. In *3rd Lake Arrowhead Conference on Human Complex Systems*, Lake Arrowhead, CA, 2005.

16. S. Christley and G. Madey. Analysis of activity in the open source software development community. In *the 40th Annual Hawaii International Conference on System Sciences (CD-ROM). Computer Society Press.*

17. S. Christley and G. Madey. Global and temporal analysis of social positions at sourceforge.net. In *The Third International Conference on Open Source Systems (OSS 2007), IFIP WG 2.13*, Limerick, Ireland, 2007.

18. S. Christley, J. Xu, Gao Y.Q., and G. Madey. Public goods theory of the open source development community. In *Agent 2004*, Chicago, IL, 2004.

19. A. Clauset and M. Newman. Finding community structure in very large networks. *Physics Review*, E 70(066111), 2004.

20. N. Collier. Repast: An extensible framework for agent simulation. http://repast.sourceforge.net/projects.html.

21. N. Collier and T. R. Howe. Repast 2.0: Major changes and new features. In *Seventh Annual Swarm Researchers Conference (Swarm2003)*, University of Notre Dame, IN, 2003.

22. F. Crimmins. Web crawler review. http://dev.funnelback.com/crawler-review.html, 2001.

23. K. Crowston, H. Annabi, and J. Howison. Defining open source software project success. In *Proc. of International Conference on Information Systems (ICIS)*, Seattle, Washington, 2003.

24. K Crowston and B. Scozzi. Open source software projects as virtual organizations: Competency rallying for software development. *IEE Proceedings Software*, v. 149(1):3–17, 2002.

25. K. Crowston, B. Scozzi, and S. Buonocore. An exploratory study of open source software development team structure. http://floss.syr.edu/tiki-index.php, 2002.

26. Jean-Michel Dalle, P. A. David, Rishab A. Ghosh, and W. E. Steinmueller. Advancing economic research on the free and open source software mode of production. Industrial Organization 0502007, EconWPA, February 2005. available at http://ideas.repec.org/p/wpa/wuwpio/0502007.html.

27. SourceForge.net Research Data. http://www.nd.edu/~oss/Data/data.html.

28. L. Eikvil. Information extraction from world wide web - a survey. Technical Report 945, Norweigan Computing Center, 1999.

29. J. Feller and B. Fitzgerald. *Understanding open source software development*. Addison-Wesley, London, UK, 2002.

30. L Freeman. A set of measures of centrality based upon betweeness. *Sociometry*, 40, 1977.

31. J.J. Gabarro. *Intellectual Teamwork: Social and Tchnological Foundations of Cooperative Work*, chapter The development of working relationships. Edited by Galegher J. , Kraut R.E. and Egido C., pages 79–110. Lawrence Erlbaum Associates, 1990.

32. Y. Gao. Topology and evolution of the open source software community. Master's thesis, Computer Science and Engineering Department, University of Notre Dame, Notre Dame, IN, 2003.

33. Y. Gao. Topology and evolution of the open source software community,. In *Seventh Annual Swarm Researchers Conference (Swarm2003)*, Notre Dame, IN, 2003.

34. Y. Gao. *Computational Discovery in Evolving Complex Networks*. PhD thesis, Computer Science and Engineering Department, University of Notre Dame, Notre Dame, IN, 2007.

35. Y. Gao, M. Antwerp, S. Christley, and G. Madey. A research collaboratory for open source software research. In *the 29th International Conference on Software Enginering + Workshops (ICSE-ICSE Workshops 2007), International Workshop on Emerging Trends in FLOSS Research and Development (FLOSS 2007)*, Minneapolis, MN, 2007.

36. Y. Gao, V. Freeh, and G. Madey. Analysis and modeling of the open source software community,. In *North American Association for Computational Social and Organizational Science (NAACSOS 2003)*, Pittsburgh, PA,, 2003.

37. Y. Gao, V. Freeh, and G. Madey. Conceptual framework for agent-based modeling,. In *North American Association for Computational Social and Organizational Science (NAACSOS 2003)*, Pittsburgh, PA,, 2003.

38. Y. Gao and G. Madey. Network analysis of the sourceforge.net community. In *The Third International Conference on Open Source Systems (OSS 2007), IFIP WG 2.13*, Limerick, Ireland, 2007.

39. Y. Gao and G. Madey. Towards understanding: A study of the sourceforge.net community using modeling and simulation. In *the 40th Annual Hawaii International Conference on System Sciences (CD-ROM). Computer Society Press*, Hawaii, 2007.

40. Y. Q. Gao, V. Freeh, and G. Madey. Analysis and modeling of the open source software community. In *North American Association for Computational Social and Organizational Science (NAACSOS 2003)*, Pittsburgh, PA, 2003.

41. M. Girvan and M. Newman. Community structure in social and biological networks. In *Natl. Acad. Sci. USA*, 2002.

42. M. W. Godfrey and Q. Tu. Evolution in open source software: A case study. In *ICSM*, pages 131–142, 2000.

43. M.S. Granovetter. The strength of weak ties. *American Journal of Sociology*, v. 78:1360–1380, 1983.

44. H. L. Grob, F. Bensberg, and F. Kaderali. Controlling open source intermediaries - a web log mining approach. In *Proceedings of the 26th Int. Conf. Information Technology Interfaces ITI 2004*, Sagreb, Kroatien, 2004.

45. R. Gulati and M. Gargiulo. Where do interorganizational networks come from? *American Journal of Sociology*, v. 104:1439–1493, 1999.

46. J.R. Hackman. *The design of work teams*, pages 315–342. Prentice-Hall, 1987.

47. A. Hars and S. Ou. Working for free? – motivations for participating in open source projects. In *Proceedings 34th HICSS Conference*, Maui, 2001.

48. C. Haythornthwaite. Tie strength and the impact of new media. In *Proceedings of the 34th Hawaii International Conference on System Sciences*, Los Alamitos, CA, 2001.

49. A. Hemetsberger. When consumers produce on the internet: The relationship between cognitive-affective, socially-based, and behavioral involvement of prosumers. http://opensource.mit.edu/papers/hemetsberger1.pdf, 2004.

50. G. Hertel, S. Niedner, and S. Herrmann. Motivation of software developers in open source projects: An internet-based survey of contributors to the linux kernel, 2003.

51. E.C. Hippel. Open source shows the way: Innovation by and for users – no manufacturer required! In *Sloan Management Review*, Boston, MA, 2001.

52. Apache Homepage. http://http.apache.org.

53. Free/Open Source Software Development Phenomenon Homepage. http://www.nd.edu/ oss.

54. Freshmeat Homepage. http://freshmeat.org.

55. Linux Homepage. http://www.linux.org.

56. Oracle JDeveloper Homepage. http://www.oracle.com/technology/products/jdev/index.html.

57. Perl Homepage. http://www.perl.org.

58. Savannah Homepage. http://savannah.gnu.org.

59. Sourceforge homepage. http://www.sourceforge.net, 1999.

60. Swarm Homepage. *http://www.swarm.org*.

61. VA Software homepage. http://www.vasoftware.net, 2005.

62. J. Howison, M. S. Conklin, and K. Crowston. Ossmole: A collaborative repository for floss research data and analysiss. In *Proceedings of 1st International Conference on Open Source Software*, Genova, Italy, 2005.

63. J. Howison and K. Crowston. The perils and pitfalls of mining sourceforge. In *Proceedings of Mining Software Repositories Workshop, Internat ional Conference on Software Enginnering (ICSE 2004)*, Edinburgh, Scotland, 2004.

64. A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: a review. *ACM Computing Surveys*, 31(3):264–323, 1999.

65. C. Jensen and W. Scacchi. Data mining for software process discovery in open source software development communities. In *Proc. Workshop on Mining Software Repositories*, Edinburgh, Scotland, 2004.

66. E. M. Jin, M. Girvan, and M. Newman. The structure of growing social networks. *Physics Review*, E 64(046132), 2001.

67. H. Joachim. Patterns of free revealing? balancing code sharing and protection in commercial open source development. http://opensource.mit.edu/papers/henkel2.pdf, 2004.

68. E. E. Kim. An introduction to open source communities. Technical report, Blue Oxen Associates, 2003.

69. M. Koster. The web robots pages. http://info.webcrawler.com/mak/projects/robots/robots.html, 1999.

70. L. Lopez-Fernandez, G. Robles, and J.M. Gonzalez-Barahona. Applying social network analysis to the information in CVS repositories. In *Proceedings of the First International Workshop on Mining Software Repositories (MSR 2004)*, Edinburgh, UK, 2004.

71. Waldrop M. M. *Complexity: The emerging science on the edge of order and chaos*. Simon & Schuster, New York, 1992.

72. G. Madey, V. Freeh, and R. Tynan. The open source software development phenomenon: an analysis based on social network theory. In *Americas Conference on Information Systems (AMCIS2002)*, Dallas, TX, 2002.

73. G. Madey, V. Freeh, and R. Tynan. Understanding OSS as a self-organizing process. In *The 2nd Workshop on Open Source Software Engineering at the 24th International Conference on Software Engineering (ICSE2002)*, Orlando, FL, 2002.

74. G. Madey, V. Freeh, and R. Tynan. *Modeling the F/OSS Community: A Quantitative Investigation, Free/Open Source Software Development, Edited by Koch, S.* Idea Publishing, 2004.

75. G. Madey, V. Freeh, R. Tynan, Y. Gao, and C. Hoffman. Agent-based modeling and simulation of collaborative social networks. In *Americas Conference on Information Systems (AMCIS2003)*, Tampa, FL, 2003.

76. G. Madey, V. Freeh, R. Tynan, and C. Hoffman. An analysis of open source software development using social network theory and agent-based modeling. In *The 2nd Lake Arrowhead Conference on Human Complex Systems*, Lake Arrowhead, CA, USA, 2003.

77. C. P. Massen and J. P. K. Doye. Identifying "communities" within energy landsacpes. *Physics Review*, E 71(046101), 2005.

78. J. Matlis. Scale-free networks. http://www.computerworld.com/networkingtopics/networking/story/0,10801,75539,00.html, 2002.

79. S. McConnell. Open-source methodology: Ready for prime time? *IEEE Software*, v. 16(4):6–8, 1999.

80. J.E. McGrath. *Groups, interaction and performance*. Printice-Hall, 1984.

81. S. Milgram. The small world problem. *Psychology Today*, v. 2:60–67, 1967.

82. J. Y. Moon and L. Sproull. Essence of distributed work: The case of the linux kernel. *First Monday*, v. 5(11), 2000.

83. M. Newman. Scientific collaboration networks: II. shortest paths, weighted networks, and centrality. *Physics Review*, E 64(016132), 2001.

84. M. Newman. Fast algorithm for detecting community structure in networks. *Physics Review*, E 69(066133), 2004.

85. M. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physics Review*, E 69(026113), 2004.

86. M. E. J. Newman, D. J. Watts, and S. H. Strogatz. Random graph models of social networks. In *Proc. Natl. Acad. Sci.*, pages pages 2566–2572, 2002.

87. M. North and C. Macal. The beer dock: Three and a half implementations of the beer distribution game. http://www.dis.anl.gov/msv/cas/Pubs/BeerGame.PDF.

88. SourceForge Research Data Archive: A Repository of FLOSS Research Data. http://zerlot.cse.nd.edu/mywiki/.

89. S. O'Mahony and F. Fabrizio. Hacking alone? the effect of online and offline participation on open source community leadership. http://opensource.mit.edu/papers/omahonyferraro2.pdf, 2004.

90. Oracle. *http://www.oracle.com*.

91. Pajek. Pajek homepage. http://vlado.fmf.uni-lj.si/pub/networks/pajek/, 2004.

92. W.W. Powell, D.R. White, K.W. Koput, and Smith J. Oweb. Network dynamics and field evolution: the growth of interorganizational collavoration in life sciences, 2002. Forthcoming, American Journal of Sociology.

93. S. Ramakrishnan. Php scripting: the growing popularity of footloose and fancy-free code. http://www.oracle.com/technology/oramag/webcolumns/2003/opinion/ramakrishnan_php.html, 2003.

94. E. Raymond. The catherdral and the bazaar. *First Monday, http://www.firstmonday.dk/*, v. 3, 1998.

95. Repast homepage. http://repast.sourceforge.net/.

96. G. Robles and R. Gonzalez-Barahona, J.and Ghosh. Gluetheos: Automating the retrieval and analysis of data from publicly available software repositories. In *Proceedings of the Mining Software Repositories Workshop. 26th International Conference on Software Engineering*, Edinburgh, Scotland, 2004.

97. C. Rossi and A. Bonaccorsi. Intrinsic motivations and profit-oriented firms in open source software. do firms practise what they preach? In *The first International Conference on Open Source Software*, Genova, Italy, 2005.

98. W. Scacchi. Issues and experiences in modeling open source software processes. In *Proc. 3rd. Workshop on Open Source Software Engineering, 25th. Intern. Conf. Software Engineering*, Portland, OR, 2003.

99. J. Scott. *Social network analysis: a handbook (2nd Edition)*. Sage Publications, London, 2000.

100. ServerWatch. sendmail – the industry-leading mail transfer agent for unix platforms. http://www.serverwatch.com/stypes/servers/article.php/16059_1299201.

101. SourceForge software map. http://www.sourceforge.net/softwaremap, 1999.

102. Netcraft survey. http://news.netcraft.com/archives/2006/04/26/apache_now_the_leader_ in_ssl_servers.html, 2006.

103. J. R. Tyler, D. M. Wilkinson, and B. A. Huberman. Email as spectroscopy: automated discovery of community structure within organizations, 2003.

104. R. Tynan, G. Madey, S. Christley, V. Freeh, and J. Xu. Task characteristics, communication, and outcome in open source software development. Under review.

105. B. Uzzi. The sources and consequences of embeddedness for the economic performance of organizations: The network effect. *American Sociological Review*, v. 61:674–698, 1996.

106. E. von Hippel. *Democratizing innovation*. MIT Press, 2005.

107. E. von Hippel and G. Krogh. Open source software and the "private-collective" innovation model: issues for organization science. *Organization Science*, v. 14(2):209–223, 2003.

108. S. Wasserman and K. Faust. *Social network analysis: methods and applications*. Structural Analysis in the Social Sciences. Cambridge University Press, Cambridge, UK, 1994.

109. D.J. Watts and S.H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, v. 393:440–442, 1998.

110. D. Weiss. A large crawl and quantitative analysis of open source projects hosted on sourceforge. Research report ra-001/05, Institute of Computing Science, Poznań University of Technology, Poland, 2005.

111. B. Wellman. *Social structures: A network approach*, chapter Structural analysis: From method and metaphor to theory and substance, Edited by Wellman B. and Berkowitz S.D., pages 19–61. Cambridge University Press, 1988.

112. H.C. White. *Markets from networks: socioeconomic models of production*. Princeton University Press, Princeton, NJ, 2002.

113. D. Wilkinson and B. Huberman. A method for finding communities of related genes. In *Natl. Acad. Sci. USA*, 2004.

114. J. Xu, S. Christley, and G. Madey. The open source software community structure. In *NAACSOS 2005*, Notre Dame, IN, 2005.

115. J. Xu, S. Christley, and G. Madey. *The Economics of Open Source Software Development*, chapter Application of Social Network Analysis to the Study of Open Source Software. Elsevier Press, 2006.

116. J. Xu, S. Gao, Y.Q. andChristley, and G. Madey. A topological analysis of the open source software development community. In *The 38th Hawaii International Conference on Systems Science (HICSS-38)*, Hawaii, 2005.

117. J. Xu, Y. Gao, J. Goett, and G. Madey. A multi-model docking experiment of dynamic social network simulations. In *Agents 2003*, Chicago, IL, 2003.

118. J. Xu, Y.Q. Gao, and G. Madey. A docking experiment: Swarm and repast for social network modeling. In *Seventh Annual Swarm Researchers Meeting (Swarm2003)*, Notre Dame, IN, 2003.

119. J. Xu, Y. Huang, and G. Madey. A research support system framework for web data mining. In *Workshop on Applications, Products and Services of Web-based Support Systems at the Joint International Conference on Web Intelligence (2003 IEEE/WIC) and Intelligent Agent Technology*, Halifax, Canada, 2003.

120. J. Xu and G. Madey. Exploration of the open source software community. In *NAACSOS 2004*, Pittsburgh, PA, 2004.

121. N. Xu. An exploratory study of open source software based on public project archives. Master's thesis, the John Molson School of Business, Concordia University, Canada, 2003.

This document was prepared & typeset with LaTeX 2ε*, and formatted with* NDdiss2ε *classfile (v3.0[2005/07/27]) provided by Sameer Vijay.*