COMPUTATIONAL DISCOVERY IN EVOLVING COMPLEX NETWORKS

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

by

Yongqin Gao, M.S.

Greg Madey, Director

Graduate Program in Computer Science and Engineering

Notre Dame, Indiana

February 2007

COMPUTATIONAL DISCOVERY IN EVOLVING COMPLEX NETWORKS

Abstract

by

Yongqin Gao

The field of study in evolving complex networks has more and more researchers working using various methods. We designed and developed a computational discovery methodology to study these evolving complex networks. This methodology is a cyclic procedure involving four different processes: data mining, network analysis, computer simulation and collaboration. The data mining process is responsible for discovering potential associations and patterns. The network analysis process is responsible for assessing the discovery found in the data mining process and analyzing the network measures in the network. The computer simulation process is responsible for generating a validated model to simulate the evolution of the complex network based on the discoveries and measures used in the previous processes. Finally, the collaboration process is responsible for designing and maintaining a research collaboratory to host our research and support any possible similar research.

To demonstrate the methodology, we applied this methodology in the study of Open Source Software movement, in particular, a study of the SourceForge.net development community. Through applying the methodology, we generated a distribution-based predictor to predict the "popularity" of a project, had more insights about the structure and evolution of the SourceForge.net community network, generated a validated model to simulate the evolution of the community, and finally implemented

and maintained a research collaboratory for the Open Source Software related research.

Dedicated to my beloved wife,

Ye Sun

and my dear parents,

Chuanshan Gao and Xiuzheng Wang,

from whom I have received enormous

support, encouragement, and love.

CONTENTS

FIGURES

TABLES

ACKNOWLEDGMENTS

I would like to express my great appreciation and gratitude to my advisor, Dr. Greg Madey, for his patient guidance and constant encouragement throughout this research. All the suggestions, technically and nontechnically, will be valuable assets to me in the future. Without his supervision, the dissertation would have been impossible.

I am grateful to Dr. Flynn, Dr. Striegel and Dr. Wood for serving on my defense committee and for their many insightful suggestions.

I am also grateful to my colleagues, Scott Christley, Yingping Huang, Tim Schoenharl, Matt Van Antwerp, Ryan Kennedy, Alec Pawling and Jin Xu for helpful discussions.

I also acknowledge the assistance of Jeff Bates, Vice President of OSTG Inc., and Jay Seirmarco, General Manager of SourceForge.net, and the contributions of University of Notre Dame undergraduate students Chris Hoffman, Carlos Siu and Nadir Kiyanclar for their assistance with data collection for this project.

Finally, this research was partially supported by the US National Science Foundation, CISE/IIS-Digital Society & Technology, under Grant No. 0222829. The project web site can be found at: http://www.nd.edu/˜oss.

CHAPTER 1

INTRODUCTION

1.1 Evolving Complex Networks

Network research is a subject frequently found within applied mathematics and physics, with the same general subject matter as graph theory. Network research itself is the study of graphs as a representation of symmetric relations or, more generally, of asymmetric relations between discrete objects [85]. Network research can improve our understanding of many real systems such as communication networks and social networks. Many well-studied networks have apparent generating principles, which means the topology of the network is determined by a given generating principle, such as the physic network of a LAN (star shape or bus shape). These kinds of networks are called “simple networks”. However, there are large-scale networks to which we cannot attribute apparent generating principles, such as the Internet [73], the gene regulatory networks and the social networks. These networks have a mixture of randomness and structure and are called “complex networks”. Usually, these complex networks are decentralized and self-organized. The control and order in this kind of networks is emergent rather than predetermined.

Evolution is another complication of these networks. Many of the real networks are evolving networks, and the structure of the network is changing continuously with respect to the time. This complication made these networks even more difficult to study. However, the knowledge of the principles of the evolution and structure of networks are of vital practical importance, like efficient indexing and searching of the Internet [97]; stability analysis for biological and communication networks [57]; epidemic threshold studies for social science [87] and networking influence in software engineering practice [18].

Because of the potential importance of the study of evolving complex network, there is much research related to the evolving complex networks. Complex network research began with the foundation of random network theory by Erdös and Rényi in 1959 [38]. The construction and evolution of a random network is determined by each individual node deciding its own connection based on the local knowledge it has. The topology of this kind of decentralized and self-organized network is unclear. Several important studies [82, 83, 84, 17] on complex networks have been conducted and provided methods of analyzing and understanding the underlying mechanisms of their topology, especially the Random Network Theory (proposed by Erdös and Rényi) [38], the Small World Networks (proposed by Watts and Strongatz) [101] and the Scale Free Networks (proposed by Albert and Barabási) [7]. In this dissertation, we applied techniques of computational discovery to study the evolving complex networks.

1.2 Computational Discovery

The study of computational scientific discovery emerged from the view that science is a problem-solving activity, that heuristics from problem solving can be applied to the study of scientific discovery in either historical or contemporary cases, and that methods in artificial intelligence provide techniques for building computational systems. Much of the current work in computational discovery is occurring within applications to particular sciences, especially bioinformation science [34, 95, 16, 64]. There are gradients from computational discoveries that are not based on AI methods, to computational discoveries that are based on AI methods, to methods with a “cognitive flavor.”

Becuase of the inherent computational complexity of the evolving complex network, we used computational discovery methodology to study the evolving complex networks, such as the community networks of the Open Source Software community.

1.3 Open Source Software Research

In this dissertation, we proposed a computational discovery methodology to study the evolving complex networks. Meanwhile, we also applied the methodology on a real, evolving complex network. The case study we studied in this dissertation is the community network in the Open Source Software (OSS) developer community [25, 98]. By applying the method on the OSS community network, we were able to get more insights about the OSS community. In this section, a brief introduction about the research on the Open Source Software development phenomenon will be provided.

Software is central to the functioning of modern computer-based society. The OSS (Open Source Software[1]) phenomenon is a novel, widely growing approach to develop both applications and infrastructure software. It exhibits many counter-intuitive properties and is not well understood [37]. Open Source Software development, despite often consisting of volunteers dispersed worldwide, now competes with commercial software firms.

In 1984, Richard Stallman founded the Free Software Foundation[2], the first official organization dedicated to OSS. OSS should not be confused, however, with “freeware,” “shareware,” “use-restricted” software or “royalty-free” software. Ta-

[1] “Open source” is a certification mark owned by the Open Source Initiative (http://www.opensource.org).

[2] http://www.fsf.org/fsf/fsf.html.

ble 1.1 [45][3] summarizes the differences between these various categories of software.

The purpose of Open Source Software was not to ensure distributing software to the end user without cost, but to ensure that the end user could use the software freely (no cost, no limitation, source code available and modifiable). Essentially, OSS exists in countless varieties today, each with its own unique history [36].

Linux, perhaps the best-known Open Source Software, began modestly in 1991, seven years after the founding of the Free Software Foundation. A recent survey was carried out by the IDC[4]. It shows robust growth in revenue and market share of the operating system and subsystems market for 2003-2005 period. "Although Linux server only accounts for 1.4% of the overall operating systems and subsystems in 2005," said Al Gillenn, research vice president, System Software at IDC, "it continues to show robust growth and demonstrates that Linux servers are taking on important roles in IT customers' computing infrastructure" [24].

Another notable OSS product is the Apache web-server, begun in February 1995. As of November 2006, Apache had achieved 60.32% of the web-server market, while market share of Microsoft has accounted only for 31.04% of the market[5] as shown in Figure 1.1.

Meanwhile, many OSS hosting sites emerged to support OSS development, such as Savannah[6], Freshmeat[7], SourceForge[8] and so on. SourceForge.net, as one of the largest and most famous OSS hosting web sites, offers features like bug tracking, project management, forum service, mailing list distribution, CVS and much more.

[3]This is adapted from "The Halloween Documents," which refer to a set of confidential internal Microsoft memos, leaked to the OSS community at the end of October 1998, which discuss the concerns of Microsoft at the threat posed by OSS.

[4]IDC is a premier global market intelligence and advisory firm in the information technology and telecommunications industries.

[5]http://www.netcraft.com/survey/.

[6]http://savannah.gnu.org.

[7]http://freshmeat.net.

[8]http://sourceforge.net.

TABLE 1.1

SOFTWARE LICENSING TAXONOMY

Software Type	Zero Price	Redistributable	Unlimited Users and Usage	Source Code Available	Source Code Modifiable
Commercial (e.g., typical Microsoft products)					
Trial Software (e.g., time-bounded evaluation products)	X	X			
User-Restricted (e.g., Netscape Navigator, freely available and redistributable only to non-profit-making entities)	X	X			
Shareware (e.g., WinZip, which has a license that eventually mandates purchase)	X	X			
Freeware (e.g., Leap Frog, released in binary form only and in public domain)	X	X	X		
Royalty-free binaries (e.g., Microsoft's Internet Explorer and NetMeeting, distributed in binary form only)	X	X	X		
Royalty-free libraries (e.g., class libraries)	X	X	X	X	
Open Source (e.g., Linux, Apache)	X	X	X	X	X

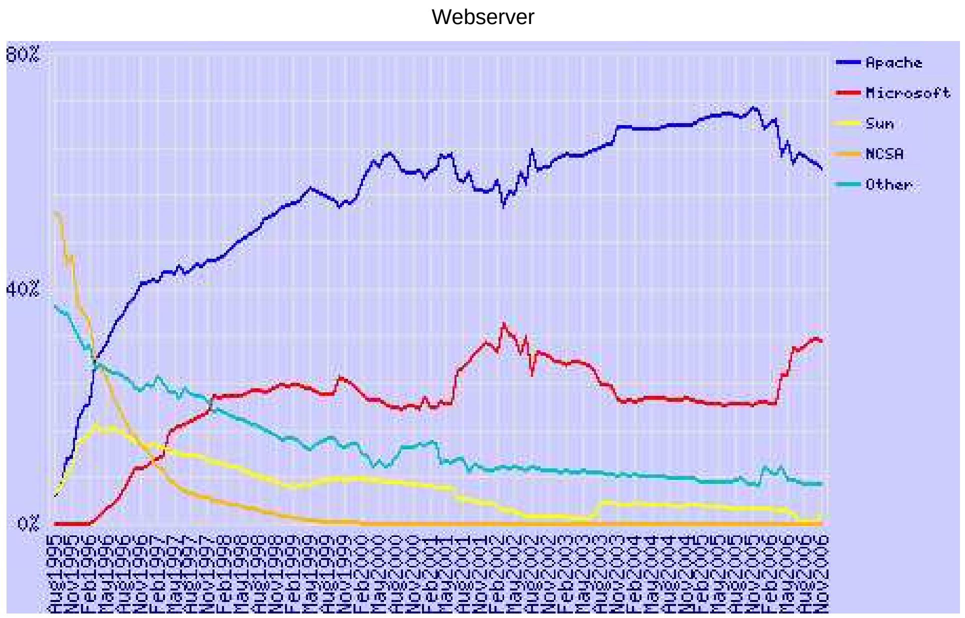


Figure 1.1. Web Server Market Share Survey about Apache (survey results by NetCraft.com)

As of November 2006, SourceForge hosted 1,439,773 registered users and 134,751 projects.

With the great and growing market share of several successful Open Source Software and fast expansion of the OSS communities, understanding the Open Source Software movement came into the focus of many researchers. For computer science researchers, understanding the Open Source Software movement can help us not only gain better understanding of evolving complex networks, but also help improve the current software engineering practices in quality, productivity and security.

Our research about Open Source Software movement focuses on the influence of the network structure evolution to the whole community and, thus, the individual entities (developer and project) in the community.

1.4 Our Methodology and Contribution

We proposed a methodology to study the evolving complex networks on a cyclic procedure: data mining was used to uncover potential discovery, then network analysis was used to assess the discovery from data mining and possibly find more characteristics of the network structure and evolution. After that, modeling and simulation were used to verify and validate the discovery from previous steps, and in this step, we generated a "fit" model to simulate the evolution of the complex networks. Finally, a research collaboratory was designed and implemented to provide the research data and some of the computational toolkits to the research community. Eventually, by sharing the research in the collaboratory, further research could be initialized. The methodology is displayed in Figure 1.2. During the study, we also applied this methodology in a real-world problem – Open Source Software research and had multiple discoveries during the procedure.

We will discuss the four major processes and our contributions in the procedure one by one.

Data mining is the first process. Data mining, also called knowledge discovery in databases, is the process of automatically searching large volumes of data for patterns, such as association rules, and discovery, such as classification and clustering. In our iterative methodology, data mining is used to discover interesting patterns about the evolving complex network. In addition to applying traditional data mining methods, we used a novel data mining method based on hypothesis tests to do the data mining. By engaging the data mining process in the case study, we found the most important features to define the popularity of a project, and a novel activity-distribution-based method to predict the "popularity" of a project.

Network analysis is the second process. This is the analysis of networks through statistical methods on selected network measures. In our iterative methodology,

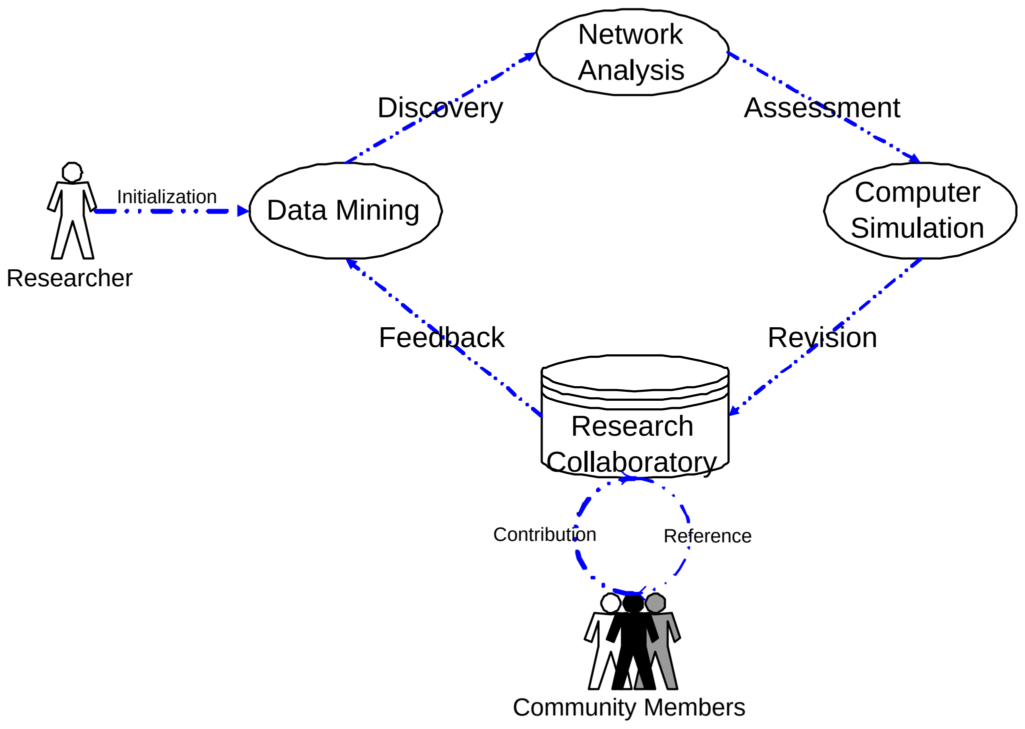


Figure 1.2. Our Proposed Computational Discovery Methodology for Studying the Evolving Complex Networks

network analysis was used to assess the discovery we uncovered in the data mining process and to discover more information about the network structure and its evolution. We introduced new sets of network analyses (centrality analysis and path analysis) into the network analysis, comparing them to the master thesis, and also used statistical hypothesis tests to verify the discovery in the network analysis. In the case study, we applied the network analysis on the more complete data set we got from SourceForge.net and gained more insights about the network structure and evolutions of the designated OSS community network and individuals in the network.

Modeling and simulation are the third process. Simulation is an imitation of some real process (for example, the evolution of complex networks). Normally, simulation entails certain key characteristics or behaviors of a selected complex network. In our iterative methodology, simulation is used to revise the results from previous steps and develop a model to imitate the evolution of the complex network. We introduced three new models in the simulation iterations, comparing them to the master thesis. By applying these new models in our case study, we generated a "fit" model to simulate the community and possibly predict the future of the SourceForge.net community by extending the simulations.

Designing and maintaining the research collaboratory is the last process. A collaboratory is a virtual community, where all the researchers can perform their research, interact with colleagues, access instruments, share data and computational resource, and access information in the repository, without regard to their physical location. In our iterative methodology, the research collaboratory is used to share the information and possible computational resources among researchers and possibly invoke further research from current research. We designed, implemented and maintained the research collaboratory, including the back-end database, the application layer and the web interface, for the Open Source Software research, based on the accumulated data sets downloaded from SourceForge.net. The collaboratory has been up and running for over a year with good utilization (details about the utilization statistics can be found in Appendix C.2).

Every process in the methodology is a stand-alone research topic in computer science research. Grouping them altogether with new methods or new algorithms in every process can help us conduct computational discovery methodology to study the evolving complex networks. More details about every process will be discussed, respectively, in the following chapters.

1.5 Layout

The remainder of the dissertation is structured as follows: Chapter 2 explains the case study and the state-of-the-art related research, focusing especially on their limitations. The following four chapters discuss the four major processes in the proposed computation discovery methodology for studying the evolving complex networks. Chapter 3 focuses on the first process – data mining. In that chapter, we discuss the new data mining method we applied on the SourceForge.net data set and the discovery by using data mining on the SourceForge.net data set. Chapter 4 discusses the second process – network statistics. In that chapter, we provide a more completed network analysis and apply it on the SourceForge.net data set. Chapter 5 delves into the third process – simulation iteration. In that chapter, we discuss the framework for simulation iterations and our modeling and simulation experiments on the SourceForge.net community. Chapter 6 explains the last process – research collaboratory. Design and implementation of our collaboratory about Open Source Software research are explained in this chapter. Finally, we give a summary and conclusion of our work in Chapter 7.

CHAPTER 2

CASE STUDY AND RELATED WORK

In order to better explain the computational discovery methodology we proposed, we applied the methodology on a real-world problem – Open Source Software research – and used this problem as the case study for the dissertation. In this chapter, we will explain the case study first, then discuss the related work of our research.

2.1 Case Study

The case study is the Open Source Software research. In particular, the Open Source Software community we inspected is the SourceForge.net development community. We will introduce the SourceForge.net first, then we will explain the setup of our case study.

2.1.1 About SourceForge.net

The goal of the case study is to understand the Open Source Software movement by studying the Open Source Software community, especially the structure and evolution of the community networks. SourceForge.net, one of the largest Open Source Software developer communities (as of November 2006, SourceForge.net hosted 134,751 projects and over 1,439,773 registered users with a centralized resource), is the designated Open Source Software community in our study, since we have monthly database dumps from them for over two years. In the study, we achieved the goal of the case study by inspecting the available information in the database dumps and the information about the networks constructed from the information stored in the database dumps.

SourceForge.net uses the back-end database to store the data supporting its web site and log the various activities happening in the community. There is ample information about the SourceForge.net community existing in the database dumps. We are most interested in the following information:

- user activities such as feature requests, bug reports, tasks and so on;
- project activities such as project initialization, project change and so on;
- project statistics such as monthly pageview, downloads, releases and so on;
- the community statistics such as project count, user count, project ratings and so on.

That information can provide us insights about the whole community and individuals (users or projects) in the community, which will help us understand how the SourceForge.net community works.

Another important piece of information stored in the database is its history, since we have collected the monthly database dumps over years and we saved all the database images for every month. With these images stored together, we can reconstruct the evolution history of the community and every individual entities in the community. This information related to community or individual evolution will be used in our study, too.

In addition to the information stored directly in the database, we can also collect information about the community networks by constructing the networks based on the information stored in the database. The Open Source Software community can be described as a collaboration network. In these networks, the nodes are developers and projects. There is a link between a developer and a project if the developer joins the project. This network is similar to some well-studied collaboration networks (movie actor and scientist collaboration network) because of common features, such as the bipartite property, small-world phenomenon and high-clustering coefficient. But the SourceForge.net community network also differs from these traditional networks. In those two traditional collaboration networks, the edge (link, relationship) is persistent, which means it will never be removed from the network after it has been added. This is not the case in the Open Source Software community, where the edge can be removed if the developer (node) quits the project (node). Thus, the collaboration network in OSS is based on active relationships instead of once-present relationships found in many collaboration networks. The detachment property is one way that makes the community network for SourceForge.net different.

Our case study about the SourceForge.net development community is based on these three types of information. Study of this community (SourceForge.net) can let us understand more about the Open Source Software movement. By obtaining more insights and, we hope, discovering the mechanisms producing the topology and evolution of the individual project and the OSS collaboration network, we may predict the future of the OSS through modeling and simulation. Furthermore, by studying the OSS community networks, we are able to find a method to study similar evolving complex networks and, possibly, get insights about similar evolving networks like the Internet and other social networks.

2.1.2 Case Study Setup

Before the discussion about the processes, we will explain the details of the dataset, hardware and software environment of the research.

First of all, we will explain the dataset (database dumps) we have. Every month, we downloaded the database dump that SourceForge.net provided. The database dump contains the most recent image of the database at SourceForge.net. For every single database dump, there is a full image of the database. There are over 100 relations (tables) in the data dumps provided to us. SourceForge.net cleanses the data by removing personal information and striping out all OSTG-specific and site functionality-specific information. We built a data warehouse comprised of these monthly dumps, with each stored in a separate schema. Thus, each monthly dump is a snapshot of the status of all the SourceForge.net projects at that point in time. And by storing these database dumps together without overwriting each other in our collaboratory, we have a complete evolution history of the SourceForge.net community. As of November 2006, the data warehouse was almost 460 GBytes in size, and is growing at about 25 GBytes per month. In every database dump, there are three major types of tables:

- The tables storing the data used to support the SourceForge.net web site; for example, the tables storing the information about user or group, etc.
- The tables storing the statistics of the whole community, including daily page access, downloads, etc.
- The tables storing the history information of the other tables.

We hosted this collaboratory using all open source softwares. Linux and Apache are used as the operating system and the web server, PostgreSQL, is used as the database server. The computers are all Intel-based computers. Details about the software and hardware environment can be found in Chapter 5.

2.2 Literature Research

Our study involves several different domains in computer science. We will discuss the related works and our contributions, respectively.

2.2.1 Open Source Software Research

There are a variety of fields related to Open Source Software research, including economics, law, social-psychology, anthropology and computer science. Although their interests may be different from our research interests, their methods or results may be helpful in our research. We will briefly introduce and summarize this research in this section.

We believe that the reason for the success of the Open Source Software is due to the complex structure and evolving properties of the community. The following researches in social and organizational science also support this belief.

Alessandra and Alessandra [11] provided a sufficiently comprehensive account of the contributions from current research to draw some general conclusions on the state of understanding of the Open Source Software phenomenon and identified directions for future research. Their report suggested that what is puzzling about Open Source Software is not so much the fact that people freely contribute to a good they make available to all but, rather, the complexity of its structure and its ability to organizationally evolve over time.

Jens [61] presented results of modeling the Open Source Software production process as a contest network, where similar vertices compete with each other and the winner gets some rewards. They suggested several possible individual motivations in the network: the active developers gain a reputation, and investors searching for highly talented applicants profit from the selection mechanism of the Open Source Software production process and finance it to receive inside information.

Moreno and Faldani [81] used complex adaptive system theory to understand and analyze the Open Source Software community. Their paper concluded that the Open Source Software community is distinctive because it is neither controlled by a central authority that defines strategy and organization nor totally chaotic, and it should be placed at a middle position between a planned community and a chaotic one. Their paper presented a description of the main characteristics of the functioning of the Open Source Software community regarding its organizational structure and development process. The concept of complex adaptive system was then introduced to simulate the OSS community. Using complex adaptive system theory, they interpreted the characteristics of the Open Source community.

There are also active researches in information science about OSS. Their focus is on the success of the information systems, often referencing the collaboration networks in the OSS community.

Information systems' success is one of the most widely used dependent variables in information systems research. Kevin, Hala and James [68] identified a range of measures that can be used to assess the success of Open Source Software projects. They identified measures based on a review of the literature, a consideration of the Open Source Software development process and an analysis of the opinions of Open Source Software developers. For each measure, they also provided examples of how they might be used in a study of Open Source Software development.

Software engineering is another research field related to OSS research. The following research focused on the study of the continuous design/redesign property of the OSS.

Open Source Software is widely used in supporting critical applications and infrastructure, including the Internet and World Wide Web themselves. The deep engagement of users and developers, coupled with the openness of systems, lead to community-based system design and re-design activities that are continuous. Gasser and Scacchi [52] presented their research into continuous, open, community-based design practices in OSS. They discussed the importance of community knowledge to software development and the construction procedure of community knowledge by studying their qualitative empirical studies of large repositories of problem-report data, primarily from the Mozilla project.

Scacchi [93] concluded that OSS projects rely on electronic communication media, virtual project management and version management mechanisms to coordinate globally dispersed software development efforts. He observed that OSS projects co-evolve with their development communities that reinvent and transfer software technologies as part of their community and project team-building process.

These researches all confirmed the importance of complex structures and evolving properties in the OSS community. The limitations of their research are that their studies are all based on non-automatic methods (e.g., statistical analysis) only. Since the data in the OSS community is huge, incomplete and noisy, non-automatic methods will become intractable to apply unaided. So we propose to develop and apply a complete cyclic methodology, including automatic and non-automatic processes, to help analyze and understand the OSS movement.

2.2.2 Data Mining

Data mining is a new and promising method recently used to study the OSS-related research.

Chawla, Arunasalam and Davis [30] reported their results of mining data acquired from SourceForge.net, the largest Open Source Software hosting web site. In the process, they introduced Association Rules Network (ARN), a (hyper-)graphical model to represent a special class of association rules. Using ARNs, they discovered important relationships between the attributes of successful Open Source Software projects. They verified and validated those relationships using factor analysis, a classical statistical technique related to Singular Value Decomposition (SVD). This paper focused on the application of association rules method in the research of OSS.

Jensen and Scacchi [62] combined techniques from text analysis, link analysis and of repository usage and update patterns to discover software processes from OSS development web repositories. They believed that this discovery could help with the understanding of the process techniques that have led to their success. However, they used only classic feature-based data mining methods in the research, which is inadequate in OSS research because of the lack of evolution information in the OSS community.

As a data analysis technique, data mining was also used in social network research. The following are two examples of this kind of research.

Jensen and Neville [63] proposed the cross-disciplinary efforts and joint research efforts in machine learning, data mining and social network analysis. They argued that older data mining algorithms were developed to analyze propositional data, which are individual records that are assumed to be statistically independent to each other. But recent data mining algorithms focused on relational data where the relations among entities are central. Network data is typically relational data. So they concluded that such joint research in data mining and social network analysis would be very useful, especially in relational data.

Kempe, Kleinberg and Tardos [67] studied the models for the processes by which ideas and influence propagate through a social network using data mining techniques. They used clustering techniques in data mining to aid their algorithm to discover the most influential nodes in the social network.

In these studies, only feature-based data mining techniques are involved. We will study not only the feature-based data mining techniques, but also a novel data-mining method based on statistical hypothesis testing in our study.

2.2.3 Network Analysis

Topology analysis is a method that can be used to understand the evolving complex networks [78, 2, 40]. It can also be used to understand the OSS phenomenon. This study also tried to understand the OSS phenomenon by studying the community as a collaboration network where every user and project can be a single node in the network.

Madey and Gao [48] analyzed the empirical data they collected from SourceForge to obtain statistics and topological information of the Open Source Software developer collaboration network. They extracted the parameters of the evolution by inspecting the network over time. They also generated a model that depicts the evolution of this collaboration network. Degree distribution, diameter and clustering coefficient are frequent attributes used to describe a network and have been used since the beginning of small world network research [12]. They also used these attributes to characterize the empirical data they collected from SourceForge, while other research tended to look at the network as a single snapshot in its evolution, which means they all based their observations on network, without respect to time. They were able to inspect the network with consideration of time, using the empirical data collected over more than two years.

Xu, Madey and Gao [104] presented the results of docking [26] a Repast [58] simulation and a Java/Swarm [56] simulation of four social network models of the Open Source Software community. They collected data about the SourceForge Open Source Software developer site for more than two years. Developer membership in projects was used to model the social network of developers. Social networks based on random graphs, preferential attachment, preference attachment with constant fitness, and preferential attachment with dynamic fitness were modeled and compared to collected data. They described how properties of social networks such as degree distribution, diameter and clustering coefficient were used to dock Repast and Swarm simulations of four social network models. The simulations grew "artificial societies" representing the SourceForge developer/project community. As a byproduct of the docking experiment, they provided observations on the advantages and disadvantages of the two toolkits for modeling such systems.

Some previous analyses studied the OSS community based on the global topology of the collaboration network. This method was not capable of revealing behaviors of a single object such as a user or a project. Our study extended the understanding of OSS to the study of individual behaviors and introduced a new measure set in the study of the OSS community.

2.2.4 Computer Simulation

Computer simulation, which integrates scientific visualization, mathematical modeling, knowledge-based design aids, intelligent user interface and many other techniques for creating platforms for trying out new ideas and doing research, was first developed in 1953 (by Enrico Fermi and his colleagues) and has gained increasing attention ever since.

Modeling is always one of the most important parts of a simulation. Mathematical methods (ODE, PDE, statistical approaches) are traditional modeling methods, but they are not sufficient enough in increasingly complex system simulations. These methods can describe macroscopic properties of a system, but fail to explain the origin of those properties. These methods cannot be easily extrapolated into situations where the assumptions behind the equations no longer hold. Also, these methods are inadequate in handling discontinuous systems and heterogeneity in populations [53].

Agent-based modeling can complement and enhance these traditional approaches. An agent is a self-contained entity with internal behavior and attributes. The typical components of an agent include internal data representations and methods of modifying their internal data and environment.

Axelrod has done a good deal of work in simulation of social networks [14, 91]. Part of his research [15] included the replications of eight popular simulation models using Swarm (a simulation toolkit that will be discussed later in this chapter) and comparing the results to the original. These models include Conway's game of life [89], Schelling's tipping model [94], Axelrod's evolution of prisoners' dilemma strategies [13], March's organizational code [77] and Riolo's prisoners' dilemma tag model [92]. These models are selected for their simplicity, diversity and reasonable run times. As a result, Axelrod concluded, "In most cases, the results were so close that we probably attained distributional equivalence for all the models." In his paper, he also discussed the docking of simulation. (Docking is an alignment process and experiment for verifying simulations. detailed definition can be found in [15].) We also did the docking for our simulation, which is discussed in my master thesis [45].

Prietula, Carley and Gasser [90] discussed the importance of computational modeling and simulation in organizational design, analysis and reengineering, especially in large-scale transnational organizations. They also discussed the existing methods in modeling and simulation of social networks such as organizations.

Previous research involving the simulation of complex networks is based on snapshots [7, 101, 86], which focused on the topology of the network at a given point in time. Since we collected data over a multiple-year period, we were able to look into the evolution of the network. This helped us understand the evolution of the network topology, the development patterns of any single object in the network and the impact of the interaction among objects on the evolution of the overall network system. Preliminary reports on this research were published in proceedings of

NAACSOS 2003 [48, 49].

2.2.5 Research Collaboratory

Problems of geographic separation are especially present in large research projects. The time and cost for traveling, the difficulties in keeping contact with other scientists, the control of experimental apparatus, the distribution of information, and the large number of participants in a research project are just a few of the issues scientists are faced with.

Therefore, collaboratories have been put into operation in response to these concerns and restrictions. However, the development and implementation prove to be not so inexpensive. From 1992 to 2000, financial budgets for scientific research and development of collaboratories ranged from US$447,000 to US$10,890,000 and the total use ranged from 17 to 215 users per collaboratory [96]. Particularly higher costs occurred when software packages were not available for purchase and direct integration into the collaboratory, or when requirements and expectations were not met.

Chin and Lansing [65] state that the research and development of scientific collaboratories had, thus far, a tool-centric approach. The main goal was to provide tools for shared access and manipulation of specific software systems or scientific instruments. Such an emphasis on tools was necessary in the early development years of scientific collaboratories because of the lack of basic collaboration tools (e.g., text chat, synchronous audio or videoconferencing) to support rudimentary levels of communication and interaction. Today, however, such tools are available in off-the-shelf software packages such as Microsoft NetMeeting, IBM Lotus Sametime, Mbone Videoconferencing [65]. Therefore, the design of collaboratories may now move beyond developing general communication mechanisms to evaluating and supporting the very nature of collaboration in the scientific context [65].

Because of the gigantic volume of information involved in the bioinformation and genome researches, research collaboratories supporting these researches are the most developed research collaboratories. There are many successful research collaboratories for biotechnology researches. NCBI [1] is one of them. Established in 1988 as a national resource for molecular biology information, NCBI hosts databases and other computational resources like BLAST to support research of analyzing genome data and disseminates biomedical information. Other successful collaboratories include FlyBase [2], VectorBase [3], EMBL [4] and GenomeNet [5].

Our collaboratory is one of the pioneers in supporting software-developing research. This kind of collaboratory is different from the collaboratory supporting bioinformation research in the following aspects:

- Data repository: Unlike the genome information, there is a lot of temporal information (e.g., monthly group statistics) in the software-developing research. So we need to design a schema to better facilitate temporal-related query in the database.
- Computational tools: The major tasks in the repositories of bioinformation are about browsing the sequences and searching for special patterns in the sequences. So powerful indexing, searching and presenting tools are most important. For our repositories, we are more interested in discovering potential rules or patterns underneath the data set, so powerful data mining or knowledge discovery tools are most important.

[1]http://www.ncbi.nlm.nih.gov.
[2]http://flybase.bio.indiana.edu.
[3]http://www.vectorbase.org.
[4]http://www.ebi.ac.uk/embl.
[5]http://www.genome.jp.

CHAPTER 3

DATA MINING

Computational discovery usually involves large amounts of data and fast data accumulation. To uncover potential patterns, relationships or correlations in these data, we need support from automatic methods like data mining [1].

Data mining can be defined as “a nontrivial extraction of implicit, previously unknown, and potentially useful information from data” [44]. Data mining involves the process of analyzing data, sorting through large amounts of data, and finding interesting patterns or relationships; e.g., picking out association rules from a large data set. A simple example of data mining is the recommendation system used by Amazon.com. Basically, the system tracks every pageview and every purchase of the users, generates shopping pattern for every user and clusters users based on their shopping patterns. Then the system gives recommendations based on the shopping patterns of similar users. For example, if a user bought a whole set of Harry Potter books, the system will recommend Narnia to the user next time, since other users sharing the similar shopping patterns bought Narnia, too. In this case, data mining is used to discovery previously unknown information about the user. Another widely used (though hypothetical) example is that of a very large North American chain of supermarkets. Through intensive analysis of the transactions and the goods bought over a period of time, analysts found that beer and diapers

[1]Materials in this chapter are partially reported in [51, 32].

were often bought together. Though explaining this interrelation might be difficult, taking advantage of it, on the other hand, should not be hard (e.g., placing the high-profit diapers next to the high-profit beers). This technique is often referred to as Market Basket Analysis.

The study of evolving complex networks often involves sorting through large amounts of data and picking out pieces of relative associations or interesting patterns, so we used data mining in the computational discovery of evolving complex networks.

3.1 Data Mining Process

Usually, preprocessing the data is needed before mining the data. Since data mining processes large amounts of data, unrelated, corrupted or noisy data may cause skewed or even error results. Preprocessing the data will guarantee that the dataset is unambiguous, correct, and complete. The data preprocessing can be further divided into data preparation and data purging, which will be discussed later, respectively.

After that, we will discuss feature selection. Feature selection is another necessary step before mining to improve the efficiency and reduce the complexity of the mining procedure.

And finally, we will apply the data mining algorithms on the data set. The procedures of data mining are illustrated in Figure 3.1 , which includes data preparation, data purging, feature selection and algorithm application.

3.1.1 Data Preparation

A good set of data is a prerequisite for producing effective data mining of any kind. However, since different research focuses on different aspects of the data set, even different data sets, data preparation is used to prepare the raw data set

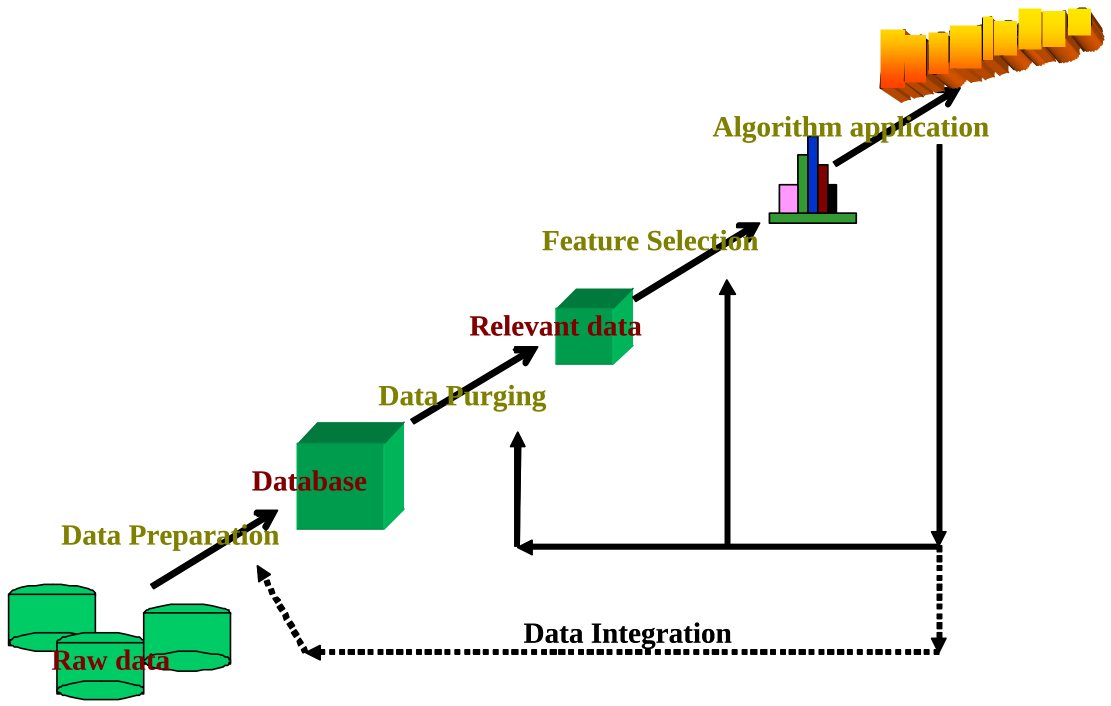


Figure 3.1. Data Mining Procedures

according to the specific research. Data preparation can be accomplished in three steps:

1. *Data discovery* consists of discovering and actually locating the data to be used.
2. *Data characterization* describes the data in ways useful to the user.
3. *Data set assembly* builds a personalized representation for the incoming data so that it can be actually mined correctly and efficiently.

Data discovery is normally a manual procedure, identifying the topic needed to be studied and then locating the related data. The original data may be a transaction processing system fed by an ATM machine or a POS terminal in a store. It may be some other record-capturing transactions or events. For many reasons, such as legal issues, data format, connectivity and so on, the raw data might not be readily accessible. So after locating the raw data, we also need to deal with some data access issues before we can proceed to next step in data preparation.

The next step is data characterization. Most raw data used in data mining is stored in some structure or system to facilitate special usage, such as for transaction systems or activity logs. Unfortunately, these structures or systems are not designed to facilitate data mining tasks, so normally, restructuring, filtering and aggregating are needed to prepare the data set to support data mining. Data characterization is used to fulfill this task.

Most data mining algorithms always require the data set to be assembled in the form of a single "flat table," which is represented by row and column. In the "flat table," the row represents the records and the column represents features for the records. This structure is different from the multiple-level structure of a database, a data warehouse or a file system hierarchy where, normally, the raw data is stored. The major task of the assembly step is to transform the raw data set, the data set generated in data characterization and possible external data source into the simple “flat table” for the data mining. These transformations may include feature extraction, explanatory structure generation, data enhancement and data enrichment. By these transformations, we thus generate the “flat table” required by the data mining.

Most of the tasks in data preparation require domain knowledge. With proper domain knowledge, we can extract related features and even invent more useful features, which will benefit the data mining. It is difficult to fully automate the data preparation, but we can use good methodology to guide the preparation to achieve a more accurate and complete data set. We used a semi-automatic method – schema-driven data preparation – which uses the data schema to guide the data preparation. By describing the problem we want to solve or the topic we want to study, we can get the original schema for the “flat table.” And then by repeatedly comparing the schema with the schemas of the raw data, we can define the final schema for the “flat table.” And using the schema and the schemas for the raw data, we can do the data preparation accordingly.

The focus of data preparation is entirely on how to get the data and to preprocess the data to fit the purpose of data mining.

3.1.2 Data Purging

Inconsistent or polluted data can defeat any techniques until the inconsistency or pollution is discovered and corrected. Data purging is trying to discover and correct these two problems before we apply the data mining algorithm on the data set.

The first problem is data inconsistency. This problem is due to the fact that different things may be represented by the same name in different data sections/tables, and even the same thing may be represented by different names in different sections/tables. The perspective with which a data section/table of variables is built has a huge effect on what is intended by the labels attached to the data. Each data section/table is built for a specific purpose, and mostly it is different from the purposes of other data sections/tables. Variable content, however labeled, is defined by the purpose of the data section/table of which it is a part. The clearest illustration of this type of inconsistency comes from considering the definition of a user from the perspective of different data sections/tables. To a software development community, a user is anyone who has made a contribution to the community, then the user will have a record of activity in the activity table. On the other hand, the personnel table regards a user as anyone who has a user ID. However, there are anonymous users, who do not have a user ID, but who have a record in the activity table. So these two tables will give different answers for the query such as “How many users are there?”

Another problem is data pollution. Data pollution can occur for a variety of reasons. One of the most common is when a user attempts to expand a system beyond its originally intended functionality. This problem is especially significant for evolving systems, because we need to update the description of data storage frequently to accommodate new data or features during system evolution. As long as the system is evolving, we need to consistently update the data storage and the system functionality, which will generate a lot of inaccurate or misleading information. Another source can be data copying. Data format can be incorrectly specified and the content from one field may be accidentally transposed into another. One such case involved a file specified as a comma-delimited file. Unfortunately, if one of the fields occasionally contained commas (the delimiter), the field would be imported into offset fields to introduce errors into the data. Since only a few of the fields might contain embedded commas, visual inspection of parts of many thousands of records can reveal no problem. Tracking down this kind of problem took considerable time and effort. Human error is another source of data pollution. While data fields are often optimistically included to capture what could be very valuable information, they can be blank, incomplete, or just plain inaccurate.

Also, for data sets from an evolving system, not only the content of the data set is changing from time to time, but also the schema of the data set. Without data purging, there will be a lot of inconsistency and pollution in the data set. The actual process of data purging involves removing typos, or validating and correcting values against a known list of entities. This process will also be mainly done manually and may be different for different data mining tasks.

3.1.3 Feature Selection

As we discussed before, a dataset in a "flat table" can be measured in two dimensions, the number of features F and the number of records R. Both F and R can be enormously large. This enormity may cause serious problems to many data mining systems.

Feature selection is one of the long-existing methods that deal with this problem. Its objective is to select a minimal subset of features according to some reasonable criteria so that the original task can be achieved equally well, if not better. By choosing a minimal subset of features, irrelevant and redundant features can be removed and, in a sense, the data set is a better representative of the whole data set. If necessary, the reduction of F can also give rise to the reduction of R by eliminating duplicates, and normally, simpler data can lead to more concise results and their better comprehensibility.

The problems of feature selection can be examined in many perspectives. The four major ones are:

1. How should we search for the "best" features?

2. What should be used to determine best features, or what should be the criteria for evaluation?
3. How should new features be generated for selection, adding or deleting one feature to the existing subset or changing a subset of features? (That is, feature generation is conducted sequentially or in parallel.)
4. How do applications determine feature selection? Applications have different requirements in terms of time, results, etc.

Feature selection is considered as a problem of searching for an optimal subset, which can usually be dealt with by some traversal algorithm. Common feature selection algorithms can also be reviewed in the way that features are evaluated, mainly univariate and multivariate evaluation. In the course of feature selection, univariate evaluation considers adding the best feature among the unchosen features to the set of chosen features, or deleting the least important features from the set of chosen features; multivariate evaluation considers a subset of features instead of a single feature in searching for the best subset. A feature selection component can be used as a preprocessor or a performance booster for a data mining process.

3.1.4 Algorithm Application

The final step for data mining processing is the actual application of the data mining algorithm. Different data mining algorithms can be used to deal with different data mining requests. Normally, association [4], clustering [60] and classification [41] are the most popular data mining tasks.

3.2 Case Study

We have been collecting monthly database dumps from SourceForge.net since February 2005, which includes information about users, projects and various activities in the community. In the rest of this section, we will discuss the data mining process we applied on the SourceForge.net data set. The goal of the data mining procedure in our study is to define the "popularity" of a project, identify the

TABLE 3.1

POPULARITY FEATURES

Feature	Description
Developers	Number of core developers in the group
Downloads	Number of downloads
Site_views	Number of views of the website for the group
Subdomain_views	Number of views of the subdomain for the group
Page_views	Number of views of pages for the group

most important feature(s) for a project and, we hope, find a good prediction of the "popularity" of a project.

3.2.1 Data Preparation

Data preparation is the first step in data mining. The major tasks in this step are to identify and collect features we want to inspect. These features include statistics describing the popularity of a project and statistics describing the user activities in a project. We also introduced new features that can describe the network characteristics of the community networks. The following three sets of features were inspected in our study.

The first set of features describe the popularity of a project, which are listed in Table 3.1. These features are the statistics stored in table *stats_project_all* in every schema (each schema is a monthly database dump from SourceForge.net).

The second set of features describe the activities that occurred in a project, which are listed in Table 3.2. There are a total of 21 possible activities in a project; the original raw data about these features are stored in the tables: *artifact*, *forum*, *project_history*, *frs_release*, *doc_data* and *people_job*.

The third set of features is introduced to describe the network characteristics of a project in the project network, which are listed in Table 3.3. These features can be generated from the project network constructed from the original table "user_project".

These features are the ones used in the data mining procedure. We used different unsupervised data mining methods to study the SourceForge.net community. Unsupervised data mining is a method to find a categorization fitting the observations. It is distinguished from supervised learning by the fact that there are no predefined output categories. The data set used in unsupervised data mining has only input features. To evaluate the result of unsupervised data mining, usually we will generate some evaluation criteria. In this study, we used the first set of features to define the criteria used in evaluation and used the second and third sets of features as the input features in the data mining.

After identifying the features, we collected these features. The first set of features was collected by direct SQL queries in the repository database. Until September 2006, there are 17 continuous schemas existing in our database. Every schema stands for a single monthly snapshot from SourceForge.net database since February 2005 and table *stats_project_all* is in all the schemas, each with a size of over 120,000 records.

The second set of features describe the activities in the project. We calculated the summary of all activities in the project monthly. Since there are different kinds of users existing in the community and the same activity taken by different kinds of users has a different influence on the project, we calculated the activity summary as a weighted summary, according to the user involved in the activity. We categorized the users into five categories: administrator, core developer, co-developer, active

TABLE 3.2

ACTIVITY FEATURES

Feature	Description
File_releases	Number of file releases for the group
New_message	Number of the new messages posted in the group forum
Followup_message	Number of the followup messages posted in the group forum
Bug_submitted	Number of bugs submitted for development
Bug_assigned	Number of bugs assigned to developer
Support_submitted	Number of support requests submitted
Support_assigned	Number of support requests assigned
Feature_submitted	Number of feature requests submitted
Feature_assigned	Number of feature requests assigned
Patches_submitted	Number of patches submitted
Patches_assigned	Number of patches assigned
Artifacts_submitted	Number of other artifacts submitted
Artifacts_assigned	Number of other artifacts assigned
Todo_submitted	Number of todo items submitted
Todo_assigned	Number of todo items assigned
Tasks_generated	Number of project tasks generated
Tasks_assigned	Number of project tasks assigned
Tasks_modified	Number of project tasks modified
Project_modified	Number of project modifications
Document_created	Number of documents created
Job_assigned	Number of jobs assigned

TABLE 3.3

NETWORK FEATURES

Feature	Description
1st_degree	First order of degree of a project
2nd_degree	Second order of degree of a project
Betweenness	Betweenness of a project in the project network
Closeness	Closeness of a project in the project network

user and lurker. These categories are explained as:

- *Administrator* is the category of users with administration privileges on the project. These users can be identified by checking the *admin_flags* field in the *user_group* table.
- *Core developer* is the category of users with CVS privileges, but without administration privileges on the project. These users can be identified by checking both the *admin_flags* field and the *grantcvs* field in the *user_group* table.
- *Co-developer* is the category of users contributing to the project, but not administrators and core developers. All the co-developers have assigned tasks, including bugs, features, patches, supports, tasks, todos, jobs and other artifacts. They can be identified by querying the joint table of all activity tables, like *artifact*, *forum*, *people_job*, *file_releases*, *project_tasks* and *doc_data*.
- *Active user* is the category of users having activities on the project, but not belonging to the first three categories. The most popular activities of these users are *new_forum_message*, *followup_forum_message*, *bug_submitted*, *support_submitted*, *artifacts_submitted* and *feature_submitted*. They can also be identified by querying the joint table we generated for identifying co-developers.
- *Lurker* is the category of users solely registering on the project without taking any activities during the inspected time frame. Majority of users in the SourceForge.net belong to this category. They can be identified by comparing the joint table we generated before with the *user_project* table, which records all the registered user in the community.

We also illustrated the flow chart of the user categorizing procedure in Figure 3.2. By using this procedure, we are able to categorize the users into these five categories.

We only inspected users existing in both *user_project* table and *user_project_act* table. This means that only users having at least one activity will be considered. Our database schemas are month by month; every month there are a certain number of users who do not have any activity. These users can be administrator, core developer, co-developer or active user (Lurkers do not have any activity, by definition). Since our data mining is focused on user activities, introducing these users without any activities will only introduce noise into the data mining. Moreover, as we will explain in the next chapter, in any month, no more than 15% of the users have activities, so removing these users without any activities can significantly improve the data mining performance. Thus, by removing the users without any activities, we can both increase the accuracy and reduce the complexity of the data mining procedure.

Figure 3.3 shows the active developer role distribution in the data set of June 2006. Since role is only unique given both user and project, the role distribution is based on every user-project pair instead of user only. For example, if user u_i is an administrator in project p_i and an active user in project p_j, u_i will be counted twice in the distribution. "Lurker" is not included in the distribution, although it is the largest part of the developer community. The relation between the lurker and the whole developer community is described in additional details in the next chapter.

After determining the user categories, we used these categories to weight the activities. Certain activities, like $file\_release$ and $project\_modified$, can only be taken by an administrator, so we will not weight these activities. For the other activities, we will assign the weight according to the user category of the user taking the activity. Weights are used to differentiate the contribution of the activities by different user categories. We assigned weights for administrator, core developer, co-developer and active user as 4, 3, 2, 1. So given the activity as act_i, the weight as

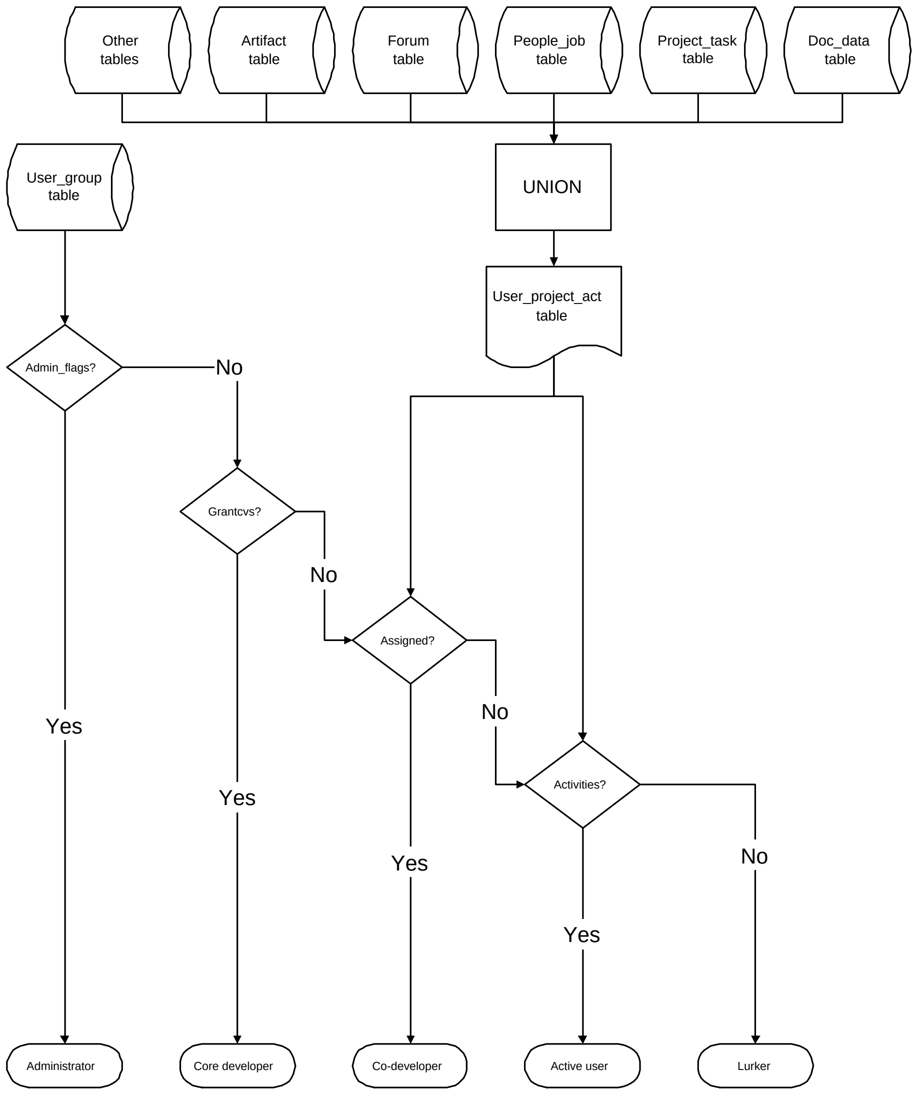


Figure 3.2. User Categorization Procedure

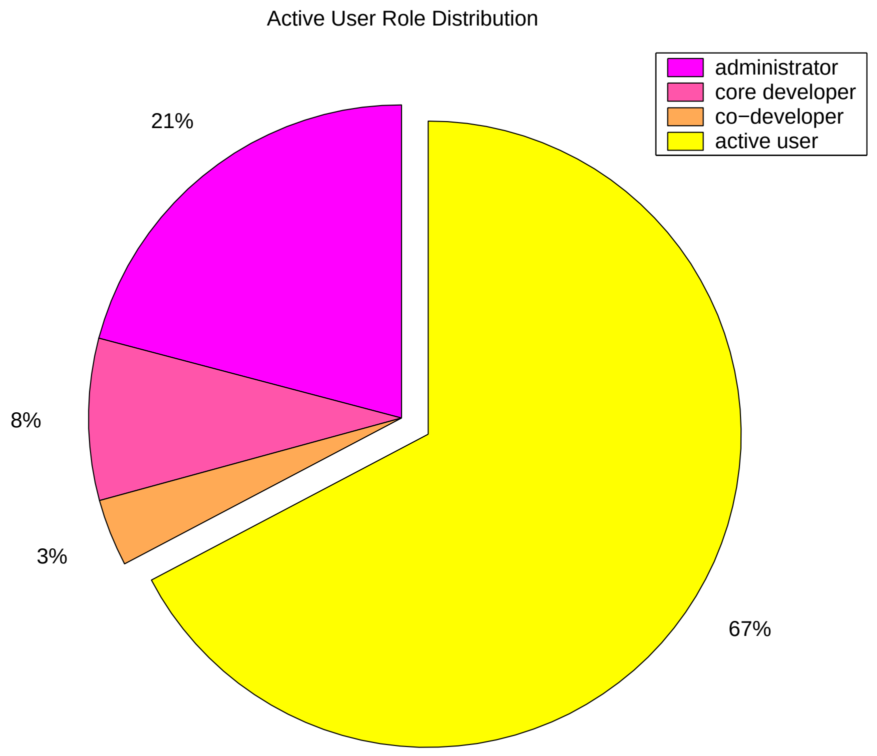


Figure 3.3. Active Developer Role Distribution

ω_i, the corresponding activity feature (every activity feature represents an activity category listed in 3.2), AF for a project can be calculated as

$$AF = \sum_{i=1}^{n} \omega_i \times act_i \tag{3.1}$$

Thus, we can collect the second set of features – activity features – by applying this equation on every feature category.

We also introduced some novel features into the data mining, which describe the network characteristics of community networks.

The SourceForge.net community includes two major entities. One entity is the developer or user in the community. The other entity is the project existing in the OSS development community. Using different prospectives, the community can be described as three different networks.

1. *Bipartite network*: The OSS community can be presented as a graph $G(V, E)$, where $V = V_p + V_u$, $V_p = \{v_p | v_p\ is\ a\ project\ in\ the\ community\}$ and $V_u = \{v_u | v_u\ is\ a\ user\ in\ the\ community\}$;
$E = \{e | e\ is\ an\ edge\ between\ v_p\ and\ v_u,\ when\ user\ v_u\ participates\ in\ project\ v_p\}$.
2. *User network*: The OSS community can also be presented as a graph $G(V, E, W)$, where
$V = \{v | v\ is\ a\ user\ in\ the\ community\}$,
$E = \{e | e\ is\ an\ edge\ of\ (v_1, v_2),$
$when\ v_1\ and\ v_2\ participate\ in\ one\ or\ more\ common\ project\}$
3. *Project network*: The OSS community can also be presented as the duel network of part (2).

The project network and user network can be transformed from the bipartite network as shown in Figure 3.4. More detailed explanation of these networks and transformations can be found in the next chapter.

The last set of features are the network features. These features are all based on the project network, since our focus is the networking characteristics of the projects. The features we introduced are focused on the network measures of the project in the project network, including first and second degree, betweenness and closeness.

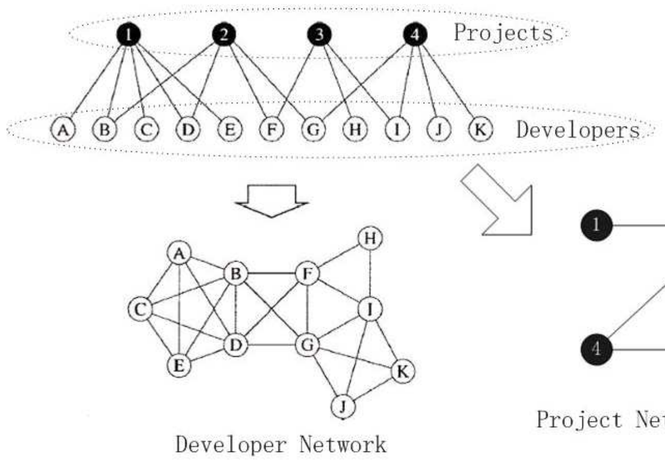


Figure 3.4. Network Transformation (adapted from Newman, Strogatz and Watts, 2001)

First degree FD_i is the number of projects that are only one link away from project i in the project network, and second degree SD_i is the number of projects that are exactly two links away from project i in the project network. Note that FD_i and SD_i are mutually exclusive by definition. Definitions of betweenness and closeness can be found in Chapter 4.

We collected this set of network features by inspecting the project network constructed from the original "user_group" table in every schema.

3.2.2 Data Purging

As we described in the previous section, we need to clean the data to improve the accuracy and efficiency of the data mining procedure.

There are more than 17,000 active projects existing in the SourceForge.net community, according to the our database dump of September 2006. Some of the projects are newly generated, some of the projects are very active and some of the projects have limited activities. Meanwhile there are some projects that do not have a single activity for a certain month, and we call these "inactive project" of the month. Since project activity is the major focus we are inspecting in the projects, we will remove those inactive projects. The numbers of inactive projects are different from month to month. However, in the dataset we have, the majority of the projects are active; for example, only 594 out of 108,947 projects were inactive in December 2005.

SourceForge.net categorized the projects into 19 categories and the populations of these categories vary considerably (for example, category "Internet" has 24520 projects while category "Sociology" has just 390 projects). By removing the inactive projects, we may introduce bias into the dataset by changing the proportionate category allocation of the projects, which may introduce error into the data mining. In order to avoid this, we used stratified sampling to sample the projects, according to the original proportionate category allocation.

By removing the "inactive projects" and stratified sampling, we cleaned the data set for the next step in the data mining procedure.

3.2.3 Feature selection

As we explained in the previous section, we will use three sets of features in the data mining procedure. According to the usage in the data mining procedure, we group them into two groups. The first group is the first set of features. These features are the popularity features, which are used to define the target categories of the project – "popular" or "unpopular." This group of features includes five features. The second group is the second and the third sets of features. These features are the input features in the data mining procedure. This group of features includes 25 features from both the second and the third sets of features. We will apply feature selection on both groups, respectively.

The algorithm we used in the feature selection is the NMF (Non-Negative Matrix Factorization) method [71, 70]. NMF is a method used in multivariate analysis and linear algebra where a matrix X, is factorized into (usually) two matrices W and H

$$nmf(X) \rightarrow WH \tag{3.2}$$

Factorization of matrices is generally non-unique, and a number of different methods of doing so have been developed (e.g., principal component analysis and singular value decomposition) by incorporating different constraints. Non-negative matrix factorization differs from these methods in that it enforces the constraint that all three matrices must be non-negative; i.e., all elements must be equal to or greater than zero. This constraint is complied by many real-world problems like the SourceForge.net community, where negative values for a feature are not possible. By selecting the number of columns of W and the number of rows of H in NMF, the product WH will become an approximation to X. This method is used in data mining to reduce the dimension of the data set (removing dependent or insignificant features).

In our case study, matrix $X(n, m)$ is the original dataset, where the rows represent the projects (n projects) and the columns represent the features (m features). By factorizing into $W(n, l)$ and $H(l, m)$, where l is the number of independent and significant features, the original dataset can be approximated as the product of WH. Matrix W is the dataset with a reduced feature set and matrix H is the weight matrix. Every row (project) in X can be approximated by the product of the corresponding row in W and the corresponding column in H. By using NMF, we can reduce the feature size from m to l.

By using NMF feature selection, we reduce the size of the feature set from 25 to 11, which significantly improves the program performance without sacrificing the

TABLE 3.4

SELECTED FEATURES BY USING THE FEATURE SELECTION. ACTIVITY FEATURES AND NETWORK FEATURES ARE THE MOST SIGNIFICANT FEATURES.

Feature
file_releases
support_submitted
feature_submitted
followup_message
tasks_assigned
tasks_submitted
tasks_modified
1st_degree
2nd_degree
Betweenness
Closeness

accuracy. The most significant attributes include file_releases, support_submitted, task-related activities and all the network features listed in Table 3.4.

3.2.4 Algorithm application

After data preparation, data purging and feature selection, we can start to apply the data mining algorithms. In the previous sections, we already found a method to identify the popular projects and found the independent and significant features sets for every project. In this section, we will use data mining algorithm to predict the popularity of a project based on the selected features in Table 3.4. Since the average active lifespan (from the creation time of the project to the first time there is no activity in the project) of a project in SourceForge.net is around six months, we inspected the projects on this lifespan, predicting the popularity of a project based on the features six months before. For example, we can inspect the feature set of March 2005 to predict the popularity of the same project in September 2005.

The first task is using the first group of features to identify popular projects and unpopular projects, which is used to evaluate the output of the data mining algorithms. Because only the minority of the project population will be popular projects, we used the outlier detection method to find the popular projects, based on the first group of features.

Outliers are the rare or atypical data that do not comply with the general behavior or model of the data. Outlier detection is widely used in fraud detection, network intrusion detection and weather prediction. Many outlier detection algorithms are byproducts of clustering [3]. For these, outliers are objects that do not belong to any cluster. We used the outlier detection algorithm using $k-d$ trees [20]. By using this algorithm, we get 1911 popular projects out of more than 160,000 total projects.

To get the prediction we wanted, we applied multiple data mining algorithms on the data set, including unsupervised feature-based data mining and unsupervised distribution-based data mining.

Unsupervised feature-based data mining [39, 79, 99], which is usually called clustering, is the first algorithm we applied. The clustering algorithm we chose is the K-means algorithm [59]. The K-means algorithm is an algorithm to cluster objects based on attributes into K partitions. It is a variant of the expectation-maximization algorithm in which the goal is to determine the K means of data generated from gaussian distributions. It assumes that the object attributes form a

TABLE 3.5

K-MEANS CLUSTERING RESULT. SINCE WE PRESET THE K AS 10, THERE ARE 10 CLUSTERS IN THE RESULT.

Cluster ID	Size
1	6201
2	98
3	64824
4	2
5	4
6	29724
7	4
8	10
9	9
10	84
total	100960

vector space. It tries to minimize total intra-cluster variance, or, the function

$$V = \sum_{i=1}^{k} \sum_{j \in S_i} |x_j - \mu_i|^2 \tag{3.3}$$

where there are k clusters S_i, $i = 1, 2, ..., k$ and μ_i is the centroid or mean point of all the points $x_j \in S_i$.

We picked this algorithm since it converges extremely quickly in practice. By giving k as 10, the result is a cluster tree and there are 10 leaf clusters, which represent the fine-grained clusters of the projects. The size of these leaf clusters are listed in Table 3.5.

From these clusters, we can summarize the representative feature sets of different project clusters and generate the rules to describe these clusters. Here are two of the rules for the two biggest clusters:

1. if file_release in [1.050, 20.040] and followup_message in [1.238, 211.590] and 1st_degree in [1.385, 9.908] then 6
2. if 2nd_degree in [20.040, 39.030] and followup_message in [1.238, 211.590] and betweenness in [2.35e-04, 2.44e-04] then 3

Where 3, 6 are cluster IDs as in Table 3.5.

To evaluate the correctness of these rules, we compared the clustering results with the clusters (popular and unpopular) we got from the previous step. We calculated the values of support and confidence for these rules. For the previous sample rules, the support for rule 1 is 0.995 and confidence is 0.58; and the support for rule 2 is 0.112 and confidence is 1. After evaluation, we found no matter which type (popular or unpopular) we assigned for those clusters, either the support or the confidence would fall below reasonable thresholds as we explained for sample rule 1 (cluster 6) and rule 2 (cluster 3). The predictions generated by unsupervised feature-based data mining are not accurate statistically.

In addition to using feature-based clustering in classic data mining, we also applied another novel algorithm [31] to clustering the data set – clustering with a statistical hypothesis test.

Unlike a distance metric that determines how close the entities are in an N-dimensional space, a statistical hypothesis test compares the entities to determine where they are similar, based on the similarity of underlying distribution of their feature set. For our analysis, we assume that each entity has an underlying distribution of the feature set; however, this distribution is not known *a priori*. Therefore, the task of determining the cluster of the entity is best done by making as few assumptions as possible on the underlying distributions for each entity, while still being able to determine if an entity does or does not have the same feature distribution as another entity. The non-parametric test we use for analyzing entities (projects) in Open Source Software is *Fisher's contingency-table test* for variables with more than two categories [43]; however, our algorithm is independent of the actual statistical test so other more appropriate non-parametric tests may be used for different data sets.

The Fisher's contingency-table test is defined as:

Given two independent samples (S_1 and S_2) of univariate measurements, the first sample with n_1 random variables and the second sample with n_2 variables, where n_1 is not necessarily equal to n_2, each random variable in each sample is placed into one of C possible categories. The null and alternative hypotheses become:

- H_0: The distributions of S_1 and S_2 do not differ.
- H_A: The distributions S_1 and S_2 differ.

In a statistical hypothesis test, a significant result is interpreted as the null hypothesis is rejected and the alternative hypothesis is accepted; while a non-significant result indicates the null hypothesis is not rejected. This leads very naturally into an algorithm for clustering that is based on unequivalence of data instead of equivalence; in particular, if the null hypothesis is rejected, then two samples cannot be from the same underlying distribution (with some level of confidence); therefore, those samples cannot be the same social position, so they are put into separate clusters. So the clustering procedure in our algorithm works like this. We do a pair-wise statistical test between every pair of entities. We separate all of the entities that are different, and we are left with a set of entities where we could not reject the null hypothesis and that set them becoming one cluster. We continue this pair-wise comparison process with the remaining unclustered entities; i.e., the entities that got rejected or excluded from a cluster, until all of the entities have been placed into a cluster. The pseudo code for the algorithm is given below.

```
While (still unclustered entities)
```

```
Put all unclustered entities into one cluster
While (some entities not yet pairwise compared)
    A = Pick entity from cluster
    For each other entity, B, in cluster not yet compared to A
        Run statistical test on A and B
        If significant result
            Remove B from cluster
```

In the worst case, if each entity gets put into its own cluster, then each entity will be pair-wise tested with each other entity, resulting in $O(n^2)$ computational complexity. Also, the confidence level in the statistical hypothesis test is set at $\alpha = 0.95$.

For our study, we found that counts for individual activities are not a good prediction for the popularity of the project in six months. By using the statistical test in clustering, we inspected the mutual relationships between these activities instead of any single activity.

By applying the algorithm, we got the cluster as in Table 3.6. After evaluating using the same method in the previous study, we can associate cluster 1 and cluster 2 to "unpopular" project in six months, and associate cluster 3 to "popular" project in six months. We called the rules used to define cluster 1 and cluster 2 as "unpopular" rules, and the rules used to define cluster 3 as "popular" rules. These rules have much better statistics than the rules generated in the previous study. The support for the "unpopular" rules is 0.964 and the confidence for the "unpopular" rules is 0.97. The support for the "popular" rules is 0.971 and the confidence for the "popular" rules is 0.916.

We demonstrated two sample activity distributions in Figure 3.5 to have an

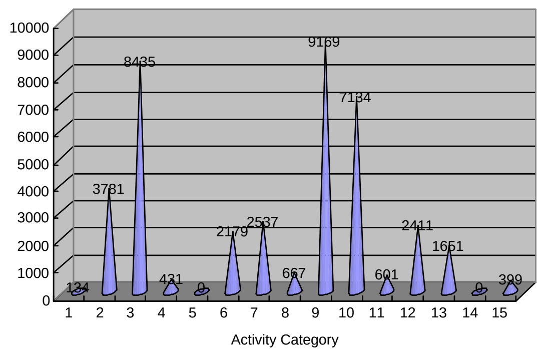
TABLE 3.6

CLUSTERING RESULT USING THE STATISTICAL HYPOTHESIS TEST BASED CLUSTERING.

Cluster ID	Size
1	89709
2	9191
3	2060
total	100960

intuitive look at the results of the clustering. There are two figures presenting activity distributions for two projects. The left figure represents a project in cluster 3 and the right figure represents a project in cluster 1. We rearranged the activity category on the X coordinate to better explain our discovery. Categories 1-3 are file releases and message posts. Categories 4-9 are activities initialized by users, and categories 10-15 are activities initialized by administrators. Although these two projects have more activities than average projects in the community, the project in the left figure has more symmetric activity distribution than the project in the right figure, if we look at the activity categories 4-15 only. This means that the activity distribution for “popular” projects is more balanced than the activity distribution for “unpopular” projects. We can understand this phenomenon intuitively, because if there are balanced activities from both user and administrator in a project, the project is more likely to be popular. Recently, Charles M. Hannum, one of the founders of NetBSD, also mentioned a similar idea in a recently published email [55].

The confidence for the “popular” rules is not as high as the confidence for the “unpopular” rules. This is because we only inspected the relationship between



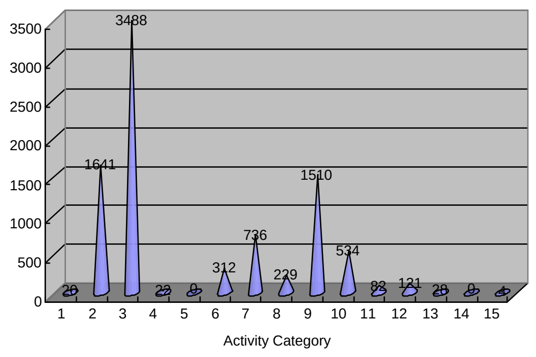


Figure 3.5. Sample Activity Distributions for Cluster 3 and 1

activities, so for certain projects with very little activity but with balanced activity distribution, those will be predicted as "popular" by our algorithm. We can improve the performance by combining the unsupervised feature-based data mining and the unsupervised distribution-based data mining.

3.3 Summary

In this chapter, we discussed the data mining part of computational discovery and its application in our case study – Open Source Software Research. First, we explained the processes needed in data mining as data preparation, data purging, feature selection and algorithm application. After that, we applied the data mining procedure in the Open Source Software Research. By applying the data mining procedure, we are able to

1. identify the "popular" projects using the features extracted from the database.
2. find the independent and significant features to predict a "popular" project.
3. predict the "popular" project using novel, unsupervised distribution-based data mining algorithm.

CHAPTER 4

NETWORK ANALYSIS

In recent research, network characteristics have received more and more attention, especially in evolving networks like the Internet, social networks and communication networks [88, 103, 72]. Analyzing these characteristics can reveal interesting information. In the previous chapter, we used data mining to discover the relationship between network characteristics and the individual project to generate predictions about the success of an individual project. In this chapter, we will use network analysis to further investigate the network characteristics in the evolution of the complex network [1].

4.1 Network analysis

Complex network-related research became popular because of the random network theory [38]. We promote network analysis in our methodology, since network analysis can help us assess the discovery from the data mining process. The analysis we used includes structure analysis, centrality analysis and path analysis. We conducted the analyses in the following manner. First, we conducted the structural analysis, including the following measures: diameter, clustering coefficient and component distribution. Then we conducted the centrality analysis, including the following measures: average degree, degree distribution, average betweenness and

[1]Materials in this chapter are partially reported in [46, 50, 48, 45].

average closeness. Finally, we conducted the path analysis on most of the previous measures.

4.1.1 Structure Analysis

The first analysis is the structure analysis [80, 27]. Structure analysis is used to inspect the macro-measures of the network structure. The measures inspected in the structure analysis describe the network structure in a global view. Study of these measures helps us understanding the influence of network structure to individual nodes in the network.

The *diameter* of a network is the maximum distance (number of hops or edges) between any pair of nodes. The diameter can also be defined as the average length of the shortest paths between any pair of nodes in the network. In our research, we are more interested in the measures that can describe the efficiency of information propagation. So the average value is more suitable for our purpose, and we used the second definition in our research. Strictly speaking, the diameter of a disconnected graph (i.e., one containing isolated components) is infinite, but it is normally defined as the maximum diameter of its sub-clusters or other approximate values. Random graphs and other complex networks all tend to have small diameters. This is the phenomenon scientists referred to as the "small world phenomenon" [100]. The smaller the diameter of a network is, the better the network is connected.

The diameter is one of the important attributes in complex network research, especially since the small world phenomenon[2] was popularized. The diameter of a network can be obtained by a modified breadth-first search of the whole network. The complexity of the algorithm is bounded by $O(MN + N^2 logN)$, where N is the number of vertices in the graph and M is the number of edges in the graph.

[2] "Six degrees of separation" is a famous claim by Ouisa, a popular character in John Guare's play (1990)

The complexity can be further approximated to $O(N^3)$. But since most complex networks are often disconnected and the actual sizes of these networks are huge and dynamic, e.g., (the Internet), it is not feasible for us to obtain the diameter by such a naive approach. Fortunately, we are still able to obtain the approximate diameter using statistical methods.

We calculated the diameter measures using similar method used in the previous study (the proof of this method can be found in Appendix A). The approximate methods can generate fairly accurate results, especially when the network size is huge ($N > 10,000$), which was verified in the previous study (we used the naive breadth-first search method to obtain the "exact" diameter of the network to verify the correctness of the approximate methods). So we will use just the approximate method in this study. More detailed explanation and discussion can be found in the master thesis [45].

The equation we used to calculate the approximate diameter D is

$$D = \frac{log(N/z_1)}{log(z_2/z_1)} + 1 \tag{4.1}$$

where N is the number of nodes in the network, z_1 is the average degree of nodes in the network, and z_2 is the average number of nodes two steps away from a given node as defined in [45].

The next measure is *clustering coefficient*. The neighborhood of a node consists of the set of nodes to which it is connected. The clustering coefficient of a node is the ratio of the number of links to the total possible number of links among the nodes in its neighborhood. The clustering coefficient of a graph is the average of the clustering coefficients of all the nodes. Recent research has found that real complex networks typically have a high clustering coefficient, which means that they exhibit a large degree of clustering [8].

The naive algorithm to get the clustering coefficient of a network requires three nested scans of the whole network. The complexity of the algorithm is high for networks of such a huge size. However, clustering coefficients of some real networks, such as the network we studied in SourceForge, can be calculated more easily from related bipartite graphs [86] by using the generating function method for bipartite graphs. More detailed explanation of this method can be found in the master thesis [45].

Using this method, the clustering coefficients of these kinds of bipartite structures result in a non-vanishing value,

$$C = \frac{1}{1 + \frac{(\mu_2 - \mu_1)(\nu_2 - \nu_1)^2}{\mu_1 \nu_1 (2\nu_1 - 3\nu_2 + \nu_3)}} \tag{4.2}$$

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

The result of equation 4.2 has been tested in several collaboration networks [86]. We also verified the approximate clustering coefficient method by measuring values that we had calculated directly from the network, using the naive algorithm (complexity of the algorithm is $O(N \times C_2^z)$, where N is the number of nodes in the network and z is the average degree of the network) in the previous study. Detailed explanation and discussion can be found in the master thesis [45]. In this study, we will use the approximate clustering coefficient to represent the clustering coefficient measure.

The last measure in the structure analysis is the *component distribution*. A component of a network is defined as the maximal subset of connected nodes. To formalize the definition of a component, first we define a path in a network as:

- A path $v_1 e_1 v_2 ... e_{n-1} v_n$ is a sequence of nodes such that from each of its nodes v_i there is an edge e_i to the next node v_{i+1} in the sequence. Normally, the first node v_1 is called the start node and the last node v_n is called the end node.

Then the component C of a network can be defined as:

- Component C is a subset of (V,E) of a network. For any pair of nodes v_i and v_j, where $v_i, v_j \in C$, there exists a path $v_i e_i ... e_{j-1} v_j$ between these two nodes. And for any any pair of nodes v_k and v_l, where $v_k \in C$ and $v_l \notin C$, there doesn't exist a path $v_k e_i ... e_{j-1} v_l$ between these two nodes.

It is important to realize that many of the real networks are composed of many components. We use the developer network as an example. First, there are developers that do not collaborate at all; i.e., each of them is the only developer of a certain project. As defined, these developers are isolated nodes in the developer network and are called isolated developers. The percentage of isolated developers in our developer network is not negligible (around 30% of total developers in the community). Second, there exists the biggest component in the developer network where there is a path between any two developers in the component. Both the biggest component and the isolated developers are important components in the developer network. Since the network evolves over time, the components are not static either. These component interact with each other by merging and splitting to keep network evolving. Study of the component distribution can also help us understand the structure of the complex network.

4.1.2 Centrality Analysis

The second analysis is the centrality analysis. Centrality analysis is used to inspect the micro-measures of the network structure or the relative importance of a node within a network. Study of these measures helped us understand the influence of individual nodes to the global network structure.

The first measure is *degree.* The degree of a node, k, equals the total number of other nodes to which it is connected, while $P(k)$ is the distribution of the degree k throughout the network. Degree distribution in real networks was believed to be a normal distribution (when $N \to \infty$), but recently, Albert and Barabási and others found it fit a power law distribution in many real networks [10]. The other measure related to degree is the average degree as $\sum P(k)/N$, which is the average of the node degrees in the network.

The next measure is *betweenness.* Betweenness is a centrality measure of a node within a network. Nodes that occur on many shortest paths between other nodes have higher betweenness than those that do not. For a graph $G(V, E)$ with n nodes, the betweenness $B(v)$ for node v is

$$B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}} \tag{4.3}$$

where σ_{st} is the number of shortest geodesic paths from s to t, and $\sigma_{st}(v)$ the number of shortest geodesic paths from s to t that pass through the vertex v. This may be normalized by dividing through by the number of pairs of vertices not including v, which is $(n-1)(n-2)$.

The last measure is *closeness.* Closeness is also a centrality measure of a node within a network. Nodes that are "shallow" to the other nodes (that is, those that tend to have short geodesic distances to other nodes within the network) have higher closeness. Closeness is preferred in centrality analysis to mean shortest-path length, as it gives higher values to more central nodes, and so is usually positively associated with other measures such as degree.

The closeness $C(v)$ for a vertex v is the reciprocal of the sum of geodesic distances to all other vertices in the graph:

$$C(v) = \frac{1}{\sum_{t \in V} d_G(v,t)}. \tag{4.4}$$

4.1.3 Path Analysis

All the previous analyses (structure analysis and centrality analysis) are based on network snapshot topology. They depict the characteristics of a static network at a given point of time. But these are not the only important analyses in a network, especially an evolving network. With sequence of network snapshots instead of just single snapshot of the networks, we are able to inspect not only the measures (diameter, clustering coefficient, component, degree, betweenness and closeness) in the previous analysis, but also the developing trends of these measures.

We conducted the path analysis on the diameter, clustering coefficient, average degree, betweenness and closeness. By inspecting these developing measures, we are looking forward to understanding more about the life cycle of the network and individual nodes in the network.

4.2 Case Study

We used the SourceForge.net community as our case study. A similar study was conducted on the SourceForge.net community in my master thesis. In that study, I studied measures like degree distribution, component distribution, diameter and clustering coefficient on the data set we collected from SourceForge.net. In this study, we expanded the study in the following aspects:

1. *More complete and accurate dataset*: In the previous study, the data set was collected by monthly web crawler. The data set is incomplete and inaccurate. Since the data set was collected through HTTP access, unexpected network error or server busy can cause connection failure, and then result in an incomplete data set. Also web crawler could take from one day to one week to finish a single execution and, meanwhile, SourceForge.net is updating itself, too. This simultaneous updating and crawling made the collected date set inaccurate. In this study, the data set is built from monthly database dump directly from SourceForge.net. The monthly database dumps include January 2003, February 2005 and every month after. Every dump file is sized from 4G to over 6G in GZIP format, except for the privacy information in the original database. The data set built on these monthly database dumps are more complete and more accurate than the data set we used in the previous study.

2. *More analysis on network snapshots*: In the previous study, we inspected only degree distribution, cluster distribution, diameter and clustering coefficient. Most of these analyses are focused on describing the structure of the complex network. In this study, we are more interested in the interaction between a single node (project) and the whole network. So in addition to more extensive

analysis on the network structure, we will inspect another group of analysis - centrality analysis. Centrality of a node (project) within a network (project network) that determines the relative importance of a node within the graph. In other words, the centrality of a node in a network is a measure of structural importance of the node. These measures attempt to quantify the prominence of an individual project embedded in a network. A central project, presumably, has a stronger influence on other network nodes. Normally, centrality measures include degree, betweenness, closeness, eigenvector centrality and flow centrality. In this study, we inspected the first three measures.

3. *Path analysis*: In statistics, path analysis is a type of multiple regression analysis. In the Internet website analysis, path analysis is a process of determining a sequence of pages visited in a visitor session prior to some desired event, such as the visitor purchasing an item or requesting a newsletter. In our study, path analysis is a process of inspecting the temporal trends of certain measures in the evolving complex network. By using path analysis, we can have more insight into the influence and interaction of individual node (project) to the other nodes in the network or the whole network.

Before discussing these analyses and measures, we need to explain the collaboration network that we studied. From SourceForge.net, we got data on two major entities – developers and projects. In this data, only one relationship existed – the participation between developer and project. There were no direct links between developers and between projects. So, we looked at this network as a bipartite network, where projects and developers were the two kinds of nodes, and edges could only connect different kinds of nodes. There are two transformations from this network – the developer network and the project network. In the developer network, there is only one type of node representing the developer in the collaboration network, and the edge in the network represents the relationship of collaboration. For every pair of nodes i and j, there is an edge connecting i and j only if i and j are collaborating on at least one project. The transformation of the bipartite network to a developer network is shown in Figure 3.4 [45]. Through the same method, we generated the project network, where a node represents a project in the collaboration network and the edge in the network represents the relationship of sharing the same developer(s). Formal definitions of these three networks can be found in Chapter 3. In the following discussion, we will abbreviate these three networks as P-NET(project network), D-NET(developer network) and C-NET(collaboration network).

Since these networks are generated from the dataset we used in the data mining, none of the "lurker" developers and inactive projects are included in these networks. The relationship between these developers or projects and the total developers or projects will be discussed later. We used the data of June 2006 as an example. There are 123,968 nodes in the D-NET, while there are totally 1,338,375 registered users in the SourceForge.net community. There are 91,713 nodes in the P-NET, while there are totally 124,393 registered projects in the SourceForge.net community.

4.2.1 Structure Analysis

The first analysis we applied is the structure analysis, including measures such as diameter, clustering coefficient and component distribution. As discussed in the previous section, the diameter is approximate on the whole network by the equation 4.1. The resulting approximate diameters for the D-NET are between 5 and 7, while the number of developers in the D-NET ranged from 97,705 to 123,968. Thus, the diameter of the network is quite small with regard to the overall network size (the number of developers in the network). On the other hand, the approximate diameters for the P-NET are between 6 and 8, while the number of projects in the P-NET ranged from 70,089 to 91,713. In the previous study, we yielded similar diameters for both D-NET and P-NET based on the monthly database dump on January 2003, which had significantly smaller networks (D-NET had about 60,000 nodes and P-NET had about 40,000 nodes). So the diameter is relatively stable compared to the significant increase of the network size.

The next measure is the clustering coefficient. We also used the approximate clustering coefficient by applying the equation 4.2. The resulting approximate clus-

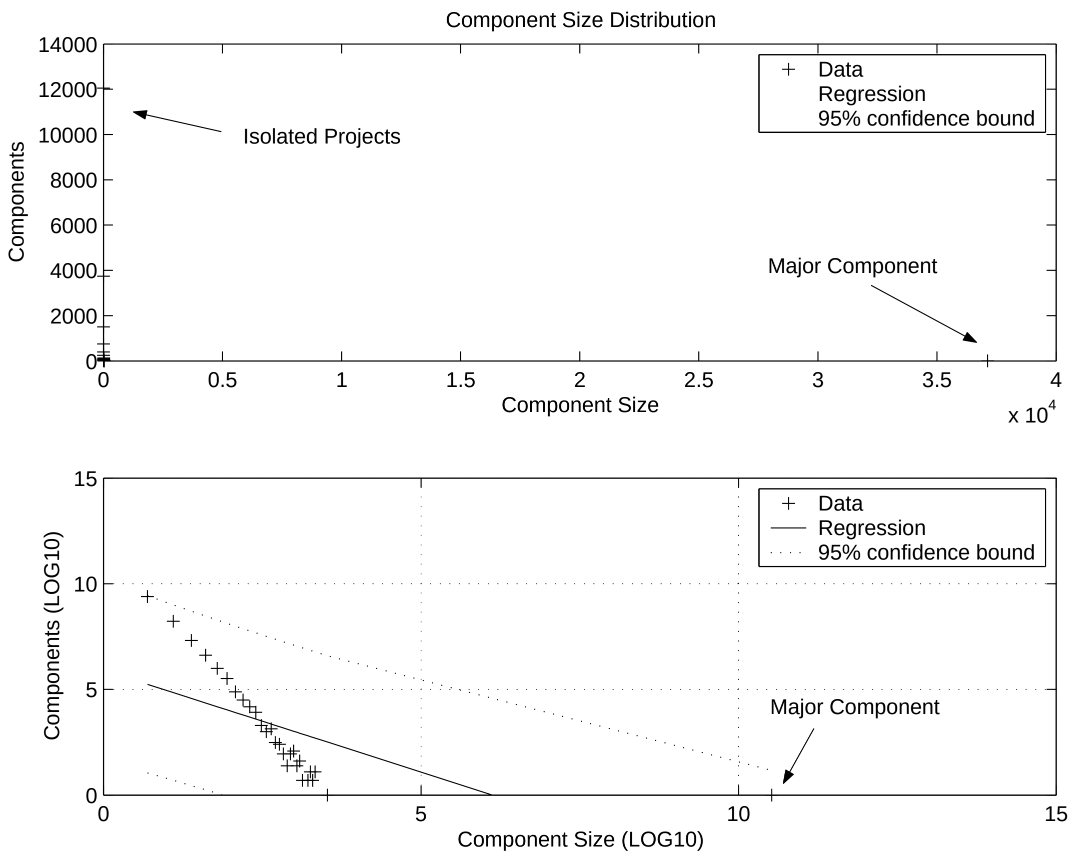


Figure 4.1. Project Network Component Distribution

tering coefficients for the D-NET are between 0.85 and 0.95. On the other hand, the approximate clustering coefficients for the P-NET are between 0.65 and 0.75. High clustering coefficients reveal the highly clustered property of both the D-NET and the P-NET, which is similar to the results we got from our previous study conducted in the master thesis, although the networks have been expanded significantly.

Both diameter and clustering coefficient are popular and efficient measures to describe the structure property of a network, especially the cluster property. Highly clustered networks are normally favored in real evolving complex networks like communication networks or social networks for better information propagation.

From the previous measures, we understand that both D-NET and P-NET are highly clustered networks. But these measures do not mean the networks are fully connected. Actually, most of the real networks are not fully connected. There will be connected parts in the network, which we called as "component". The next measure we inspected is the component distribution. In the SourceForge community, power law exists in the component distribution of the networks. In Figure 4.1, the component distribution for the P-NET for June 2006 is shown. There are two figures. In the lower figure, after applying *log* transformation on both coordinates, we found that the component distribution fits a straight line quite well without considering the biggest component (which will be called the major component in latter discussion). The R^2 [35] of linear regression with the major component is 0.4023 and the R^2 of linear regression without the major component is 0.9886. Also, in the lower figure, we illustrated the 95% confident boundary for the linear regression as the dot line. In the upper figure, where the coordinates are in normal scale, we made another interesting discovery: almost all the components are quite close to each other, except the two extremes. One extreme is the major component and the other is the isolated components. Isolated components in the graph were generated from the isolated developers we mentioned at the beginning of this subsection. This result is similar to the result we yielded in previous study.

4.2.2 Centrality Analysis

The second analysis we conducted is the centrality analysis, which focus on the following measures – degree, betweenness and closeness.

Degree is the simplest measure of the connectivity of a node in the network. We also used the C-NET, D-NET and P-NET for June 2006 as examples in this section. There are totally four degrees of developer and project in these three networks:

- *Degree of developer in the C-NET* is the number of projects in which a developer participated in the community.

- *Degree of developer in the D-NET* is the number of developers who have at least one collaboration with the given developer in the community.
- *Degree of project in the C-NET* is the number of developers who participated in the given project in the community.
- *Degree of project in the P-NET* is the number of projects share by at least one common developer with the given project in the community.

In the June 2006 dataset, the average degree of developer in the C-NET is 1.4525; the average degree of developer in the D-NET is 12.31; the average degree of project in the C-NET is 1.7572; the average degree of project in the P-NET is 3.8059, while the sizes of the C-NET, the D-NET and the P-NET are 215,681, 123,968 and 91,713. The average degree of developer and project in the C-NET is relatively low since the isolated developer (developer with single project) and the isolated project (project with only one developer) are big parts of the community. This result is also correlated to the discovery we got in the component distribution.

Then we investigated the degree distributions of the SourceForge.net community. Degree distribution is proven to have a normal distribution in the ER model when $N \to \infty$. This was believed to be a good model for the real complex network before the power law was reported for many real network systems by Barabási et. al [9]. In the SourceForge community, we found that the degree distributions (distributions for all four degrees) also followed power law, as shown in Figure 4.2, Figure 4.3, Figure 4.4 and Figure 4.5.

These figures are based on the dataset from SourceForge.net for June 2006. The left figures are the degree distributions in normal coordinates. To verify the existence of power law in these distributions, we applied log-log transformations on the data to generate the right figures, which are the degree distributions on log-log coordinates. The 95% confident boundaries for the linear regression also are

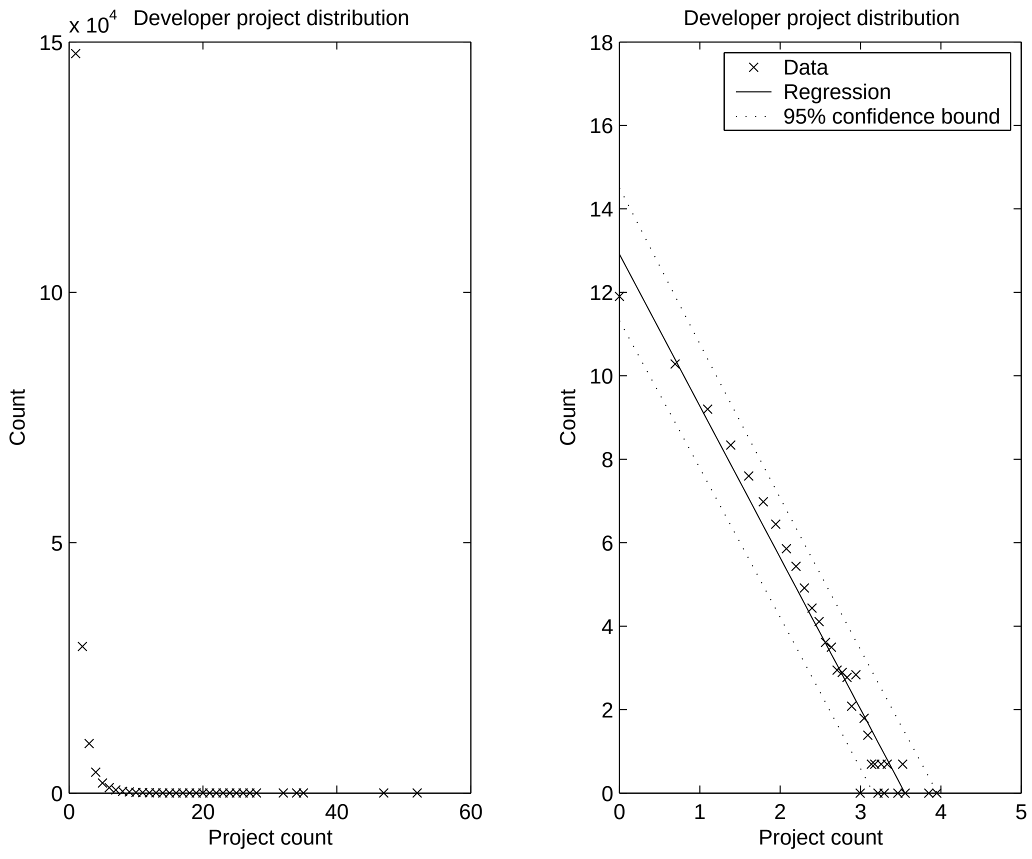
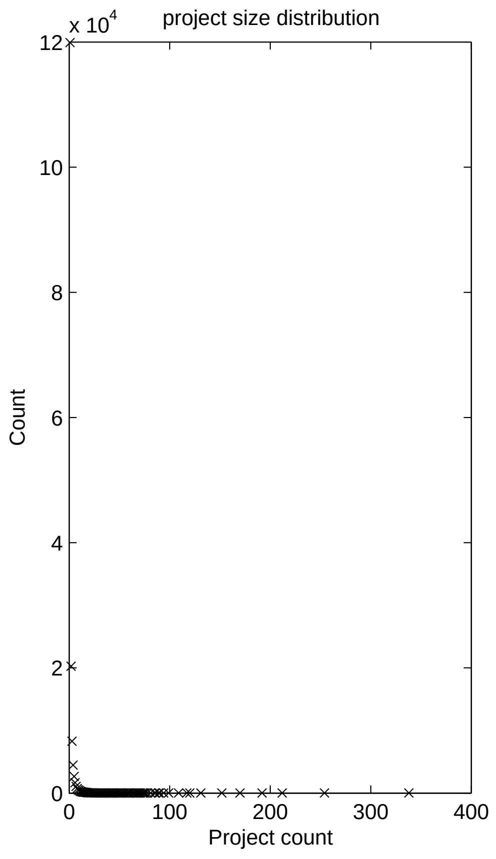


Figure 4.2. Developer Size Distribution



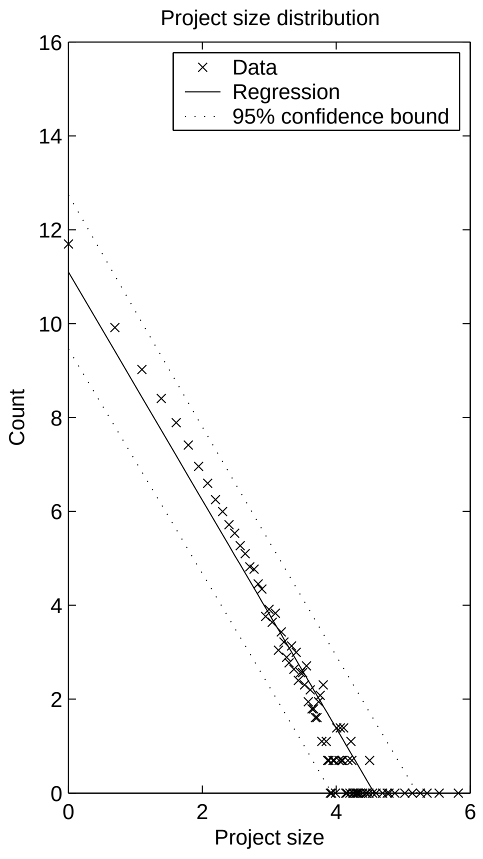


Figure 4.3. Project Size Distribution

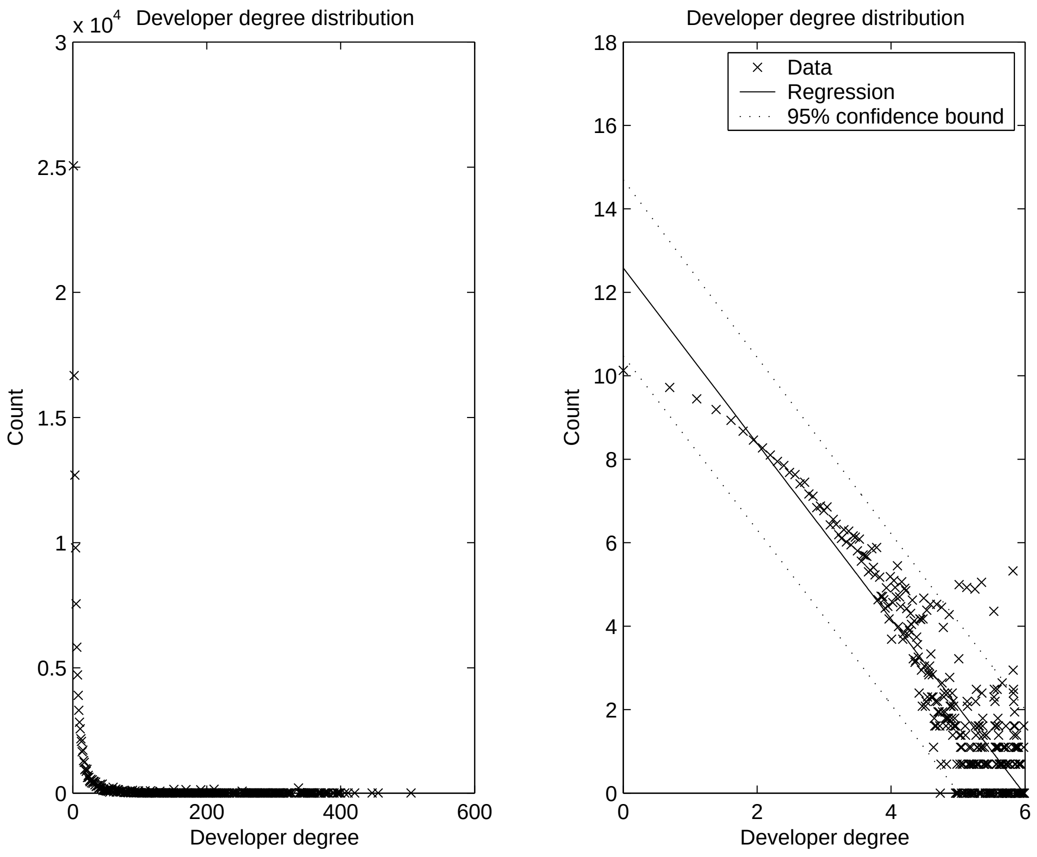


Figure 4.4. Developer Network Degree Distribution

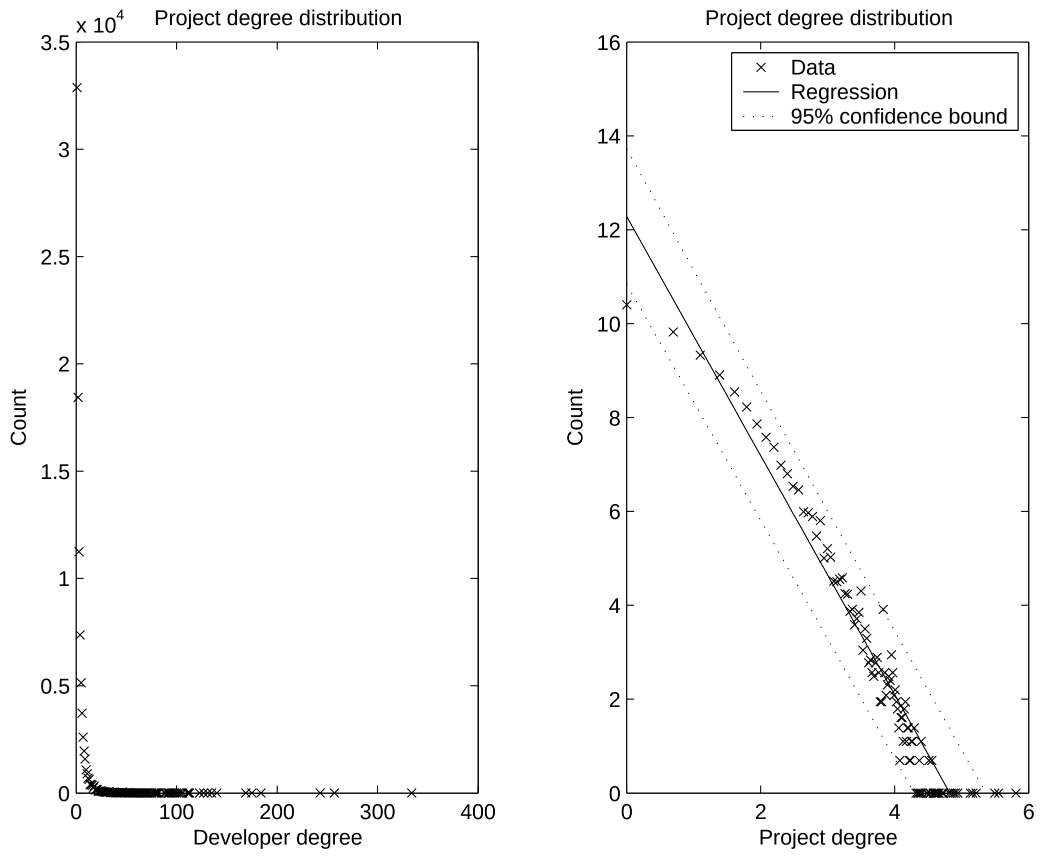


Figure 4.5. Project Network Degree Distribution

provided on the figures for all the log-log transformed degree distributions. The R^2 of linear regression for the developer degree distribution in the C-NET is 0.9577. The R^2 of linear regression for the developer degree distribution in the D-NET is 0.8132. The R^2 of linear regression for the project degree distribution in the C-NET is 0.9173. The R^2 of linear regression for the project degree distribution in the P-NET is 0.9375. Thus, these distributions all fit power law distributions very well.

Betweenness and closeness are also the common measures for centrality analysis. Betweenness is a measure to describe the importance of the node in the network according to shortest path, and closeness is a measure to describe how close the node is to other nodes. Betweenness is a normalized value in $[0, 1]$. The higher the measure is, the more central the node is to the network. Closeness is also bounded by $[0, 1]$, but it is not normalized. So closeness tends to decrease when the network size is increasing. Normally, these measures yield very small value in large networks ($N > 10,000$), so comparison of these measures only makes sense when comparing networks of similar size. Also, using the dataset from SourceForge.net for June 2006, the average betweenness for the P-NET is 0.2669e-003 and the average closeness for the P-NET is 0.4143e-005, which are relatively large for a network this size.

4.2.3 Path Analysis

All the measures in the previous sections (diameter, clustering coefficient, component, degree, betweenness and closeness) are about the topology of the networks. They depict the characteristics of a static network at a given point of time. These are not the only important attributes in a network, especially an evolving network [54, 5]. Since we had multiple monthly database dumps from SourceForge.net, we were able to investigate the development patterns of these measures of the networks. By applying the path analysis, we can study the life cycle and the evolving

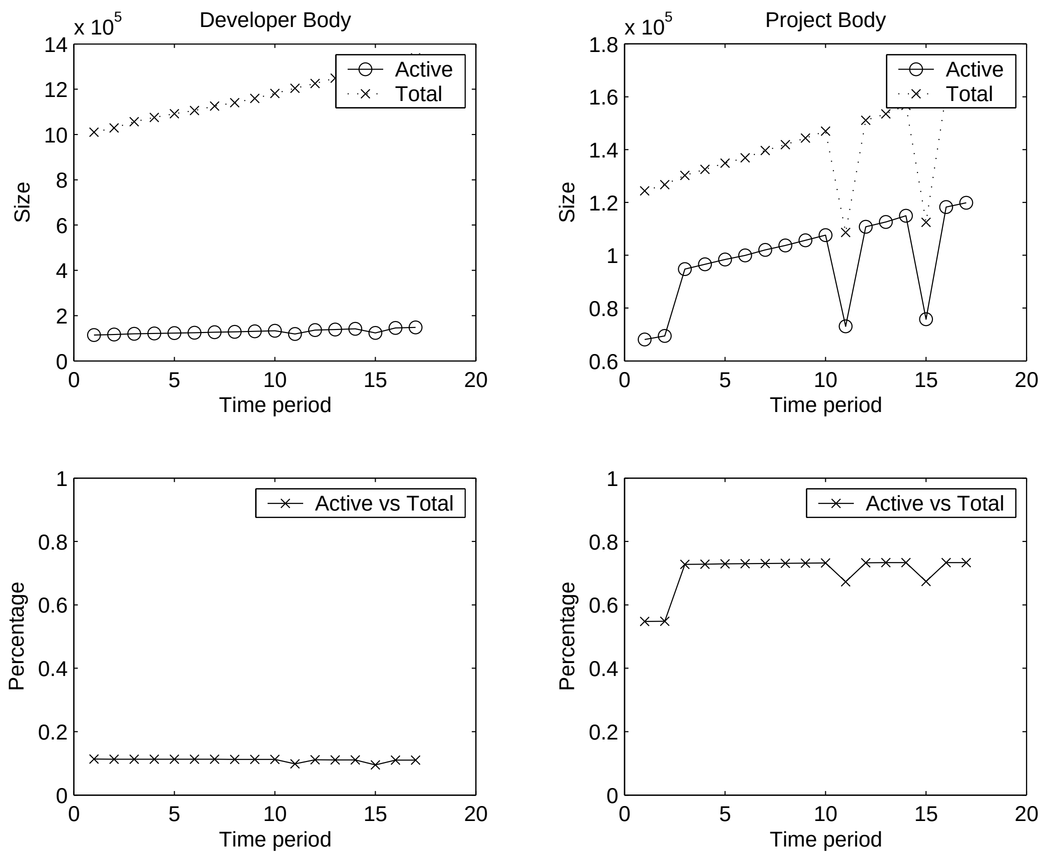


Figure 4.6. Active Developer or Project vs. Community Size. The X coordinate represents the number of months after January 2005.

patterns of the network and individuals in the network.

In the previous discussion, we realized that only a certain percentage of the registered user were active (had at least one activity) during a certain month. Also, the same situation can be discovered in projects; there was a certain percentage of the registered project that were inactive (no activity at all) during a certain month. In order to have an insight into this phenomenon, we inspected these users and projects with activities and their relationship with the whole user and project body. The result is in Figure 4.6.

The left two figures are related to the developer body and the right two figures are related to the project body. The upper two figures show the actual size of the active developer or project, compared to the whole developer or project body. The X coordinate are the time period from February 2005 to June 2006; the Y coordinate is the actual size. The sizes of active developer or project every month are marked by 'o', and the sizes of the total developer or project body are marked by 'x'. There are significant recessions at month 11 (December 2005) and month 15 (April 2006). These can be caused by data error or other abnormities. Despite these two recessions, the active developer or project have similar growth rates as the total developer or project body. We included the percentages of active developer/total developer and active project/total project in the lower two figures. We find that the percentage of active developer/total developer is maintaining a similar percentage throughout the 17 months of around 10%. And the percentage of active project/total project has a jump at month three (April 2005) from 55% to 76%, and then maintains the similar percentage throughout, except month 11 and month 15 at 76%.

Inactive developers and projects have no contribution to the community when they are inactive. But they are very important as the reserved force to keep the community running. The stable percentages in the active developer/total developer and active project/total project suggest that the SourceForge.net community is in a stable status. The initial jump in the active project percentage may suggest that the community network has some life cycle behavior. We can further analyze this when we have accumulated more data sets from SourceForge.net.

The next path analysis is on the average degrees. Average degree $< k >$, which gives the average number of links per node, is a good quantitative measurement for the connectivity of a graph. The developing pattern of the four possible average degrees is shown in Figure 4.7, Figure 4.8, Figure 4.9 and Figure 4.10. The X

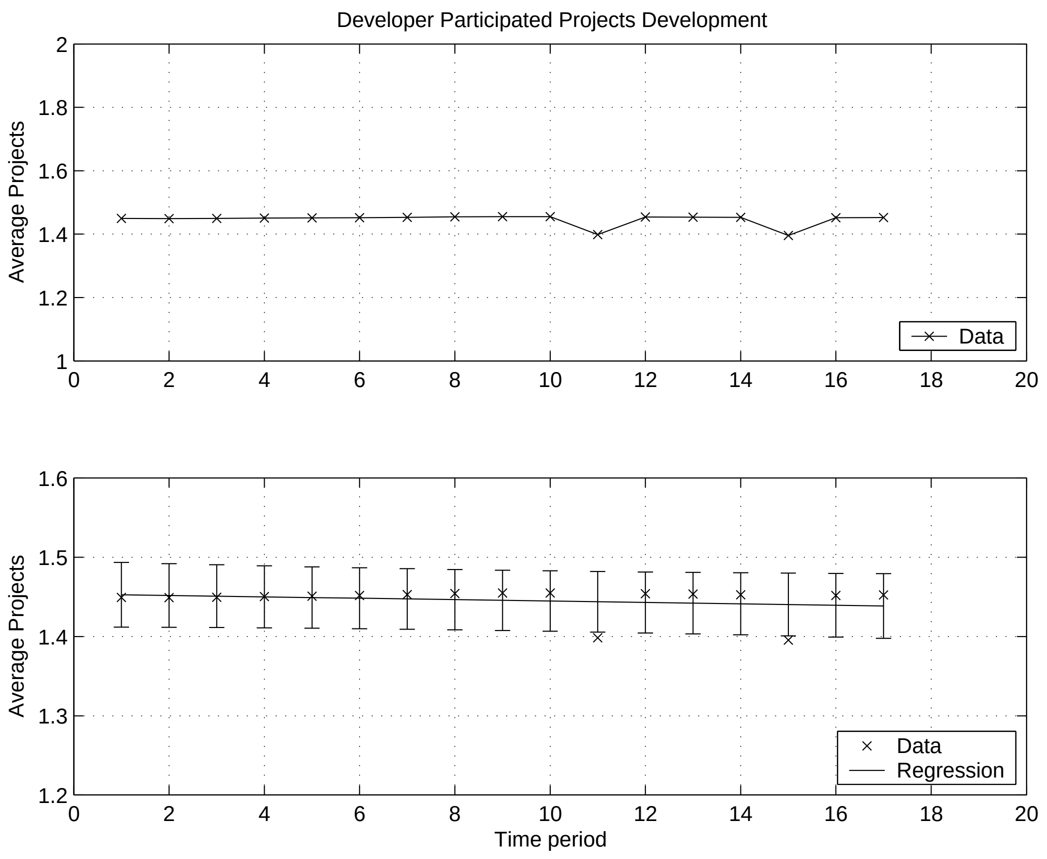


Figure 4.7. Average Developer Degree in the C-NET

coordinate in the figure is the number of months that passed after February 2005.

The average developer degree in the D-NET has increased by that time. The increase of average degree may have come from two changing parts in the network. First, the number of nodes in the network increased with time because of the arrival of new developers. Second, the total number of edges also increased through collaborations made by new developers with old ones, and collaborations made between old developers. These two changes worked together to increase the average degree. This is an important feature to characterize a network in evolution. We show the linear regression in the lower figure with a 95% error bar for every data point and

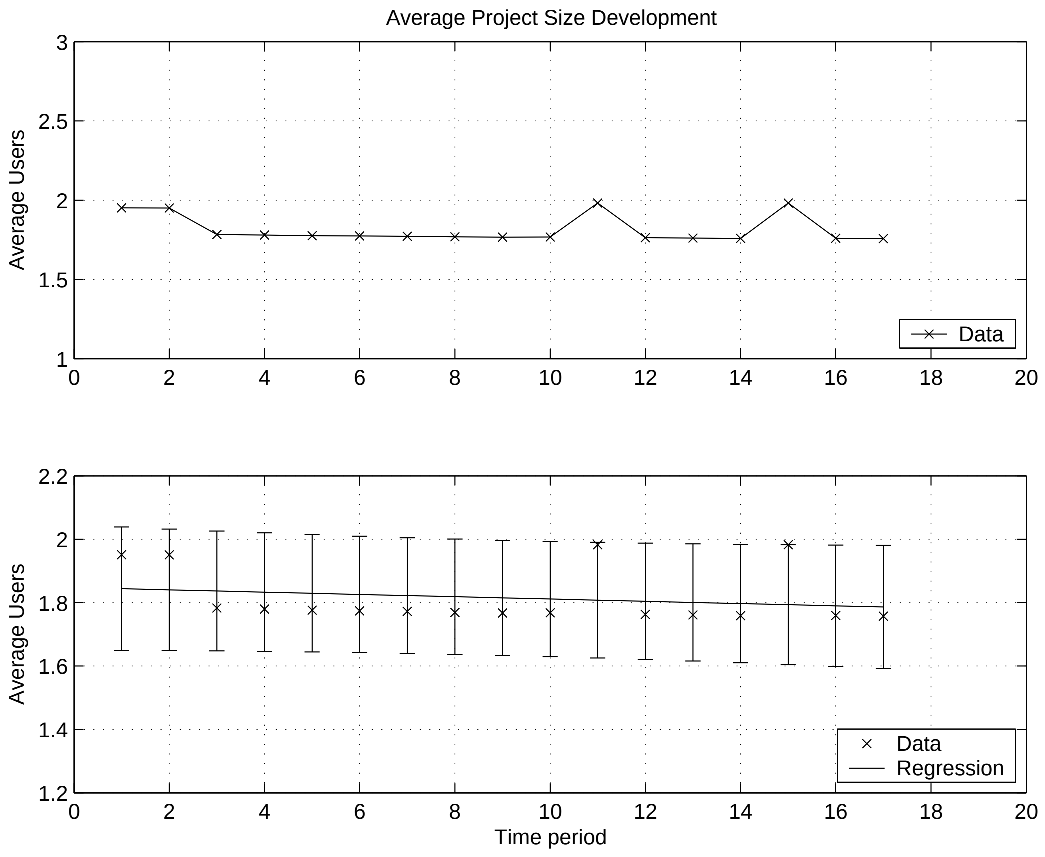


Figure 4.8. Average Project Degree in the C-NET

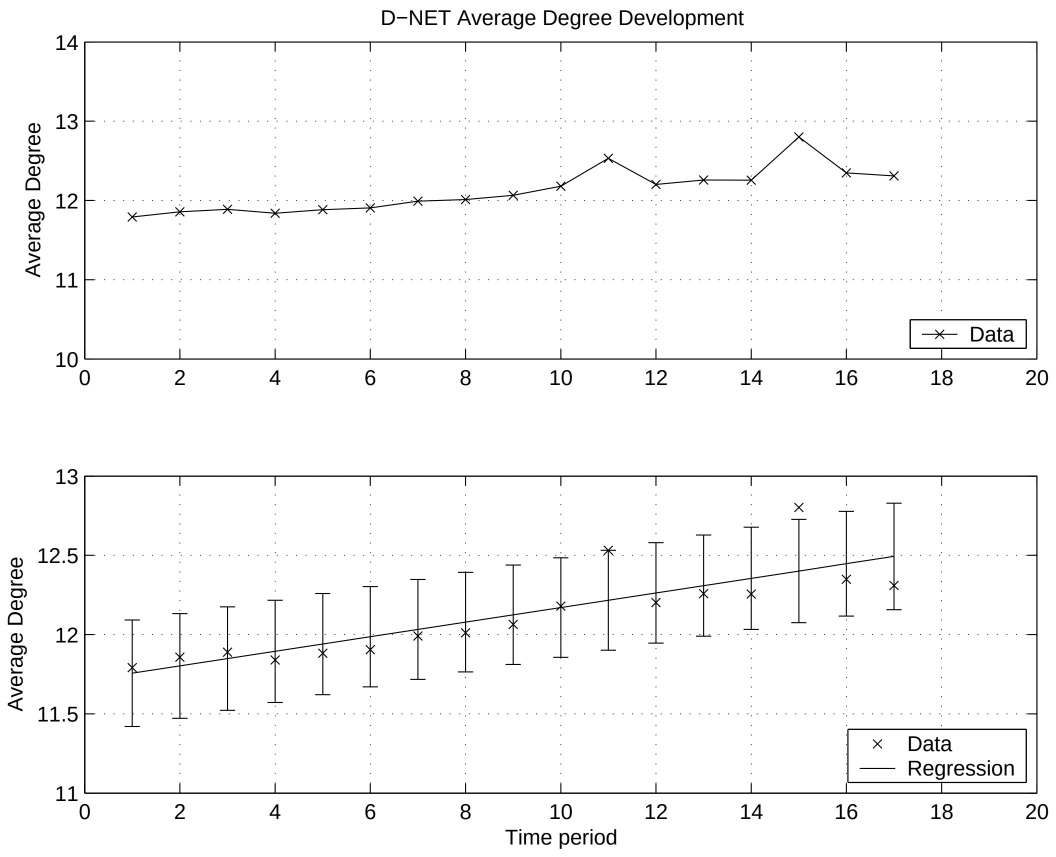


Figure 4.9. Average Developer Degree in the D-NET

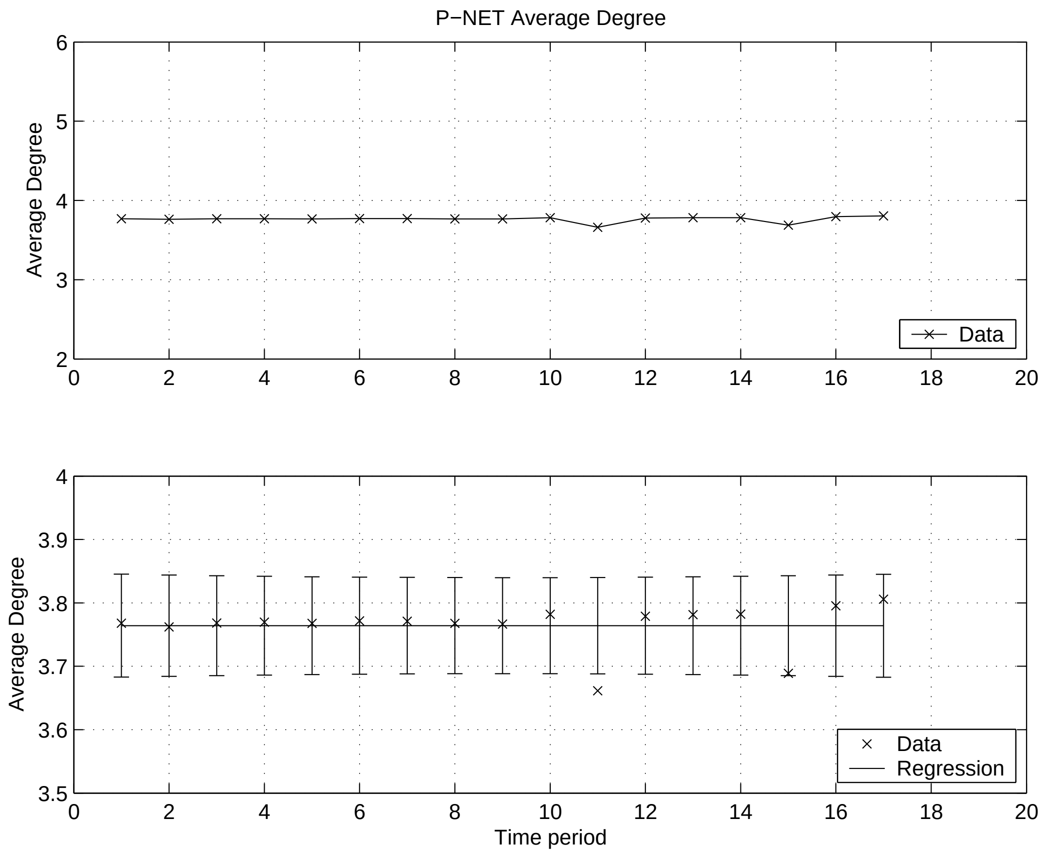


Figure 4.10. Average Project Degree in the P-NET

the slope of the regression is 0.0461. For the average project degree in the P-NET, it is very stable by this time. We observe the linear regression in the lower figure with a 95% error bar for every data point, and the slope of the regression is 0.00001.

For the average degrees in the C-NET, we show the linear regression in the lower figures with a 95% error bar for every data point. The slope of the regression for developer degree is -0.0009 and the slope of the regression for project degree is -0.0036. The average degrees are actually decreasing, which means the average project size and average number of projects a single developer participated in are decreasing. This may be explained by two reasons. First, there are more isolated projects joining the community every month than big projects. At the begining of the SourceForge.net, there are big projects elsewhere migrating to SourceForge.net, which means the C-NET has a bigger project degree initially. As the community develops, there are more small projects created in the community than migrated big projects. This will bring down the project degree of the C-NET to a stable level. The similar explanation can be applied on the developer degree in the C-NET. At the beginning of the SourceForge.net, users were actively joining projects, doing contributions which would generate an high initial developer degree. But as time went by, new users were more focused on specific projects instead of random joining and old users were also focused their interests on fewer projects. This brought down the developer degree in the C-NET.

In these three networks, P-NET and D-NET are more connected than C-NET and, thus, more robust to frequently underlying user and project changes. Study of the average degrees in P-NET and D-NET can represent more long-term behaviors of the overall community network structure than the average degrees in C-NET.

Diameter and clustering coefficient is closely related to average degree. We will conduct path analysis on these two measures next.

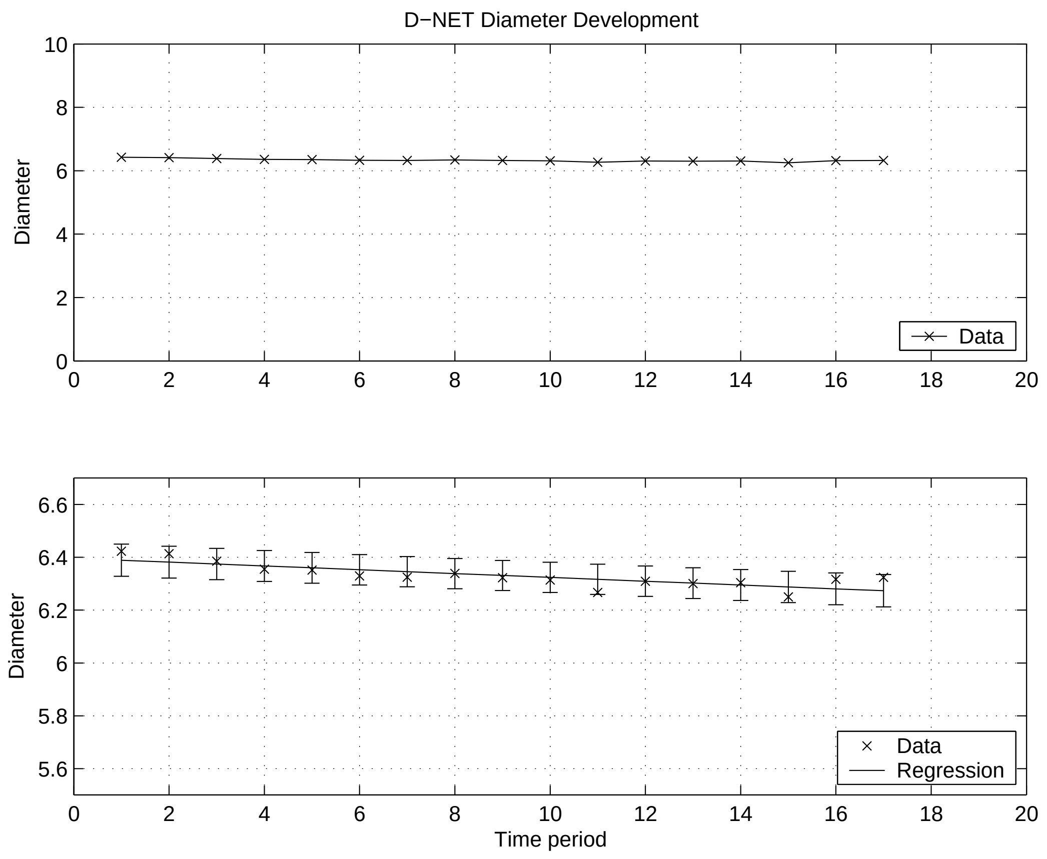


Figure 4.11. Developer Network Diameter

The diameter of the D-NET is a good measure of network communication ability. A shorter diameter results in fewer average steps needed for one developer to spread a message to another developer and less time needed for an idea to spread through the network. The D-NET has a small diameter, which was calculated in a previous section. Also, we investigated the evolution of the diameter of the D-NET, as shown in Figure 4.11.

The figure indicates that D decreases with time, which is different from the previous research [22] on random networks that reports that diameter increases with network size. The lower figure shows the linear regression with 95% error bar for the developing trend of diameter for the D-NET. The slope of the regresion is -0.0072. This decreasing trend could be because of three reasons. First, preferential attachment will cause a significant increase in the degree of hubs in the network (the vertices with more links than average in the network), thus decreasing the diameter. Second, the increasing arbitrary size of the major component means the connected component may weight more in the overall system, thus increasing the overall connectivity of the network. Finally, the diameter decreases as internal links (for example, old developers joining projects that previously existed) increase the interconnectivity of the network, thus decreasing the diameter.

Furthermore, we investigated the developing patterns of diameters in the P-NET. The result is shown in Figure 4.12. The developing patterns of diameters in the P-NET are different from the developing patterns of diameters in D-NET. The diameter is relatively stable with a slightly increasing trend. The lower figure shows the linear regression with 95% error bar for the developing trend of diameter for the P-NET. The slope of the regresion is 0.0069. This may be because there are many newly created isolated projects or small projects in the community to decrease the connectivity of the P-NET.

Clustering coefficient is another important measures of the topology of real networks. So the clustering coefficient, a quantitative measure of clustering, CC, is also a measure we investigated. The approximate clustering coefficient for the D-NET as a function of time is shown in Figure 4.13.

In the figure, we can observe the increasing trend of the clustering coefficient. The lower figure shows the linear regression with 95% error bar for the developing trend of clustering coefficient for the D-NET. The slope of the regression is 0.0011. The clustering coefficient for the D-NET tells us how much a node's co-developers are willing to collaborate with each other, and it represents the probability that two

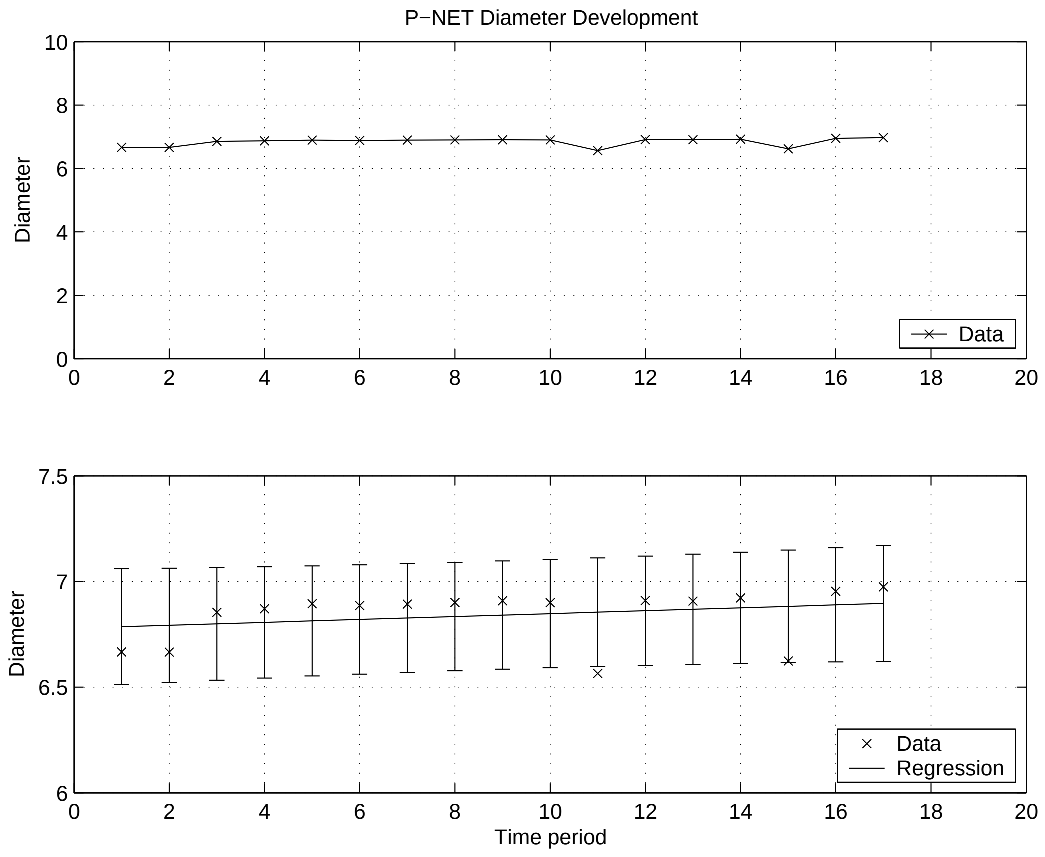


Figure 4.12. Project Network Diameter

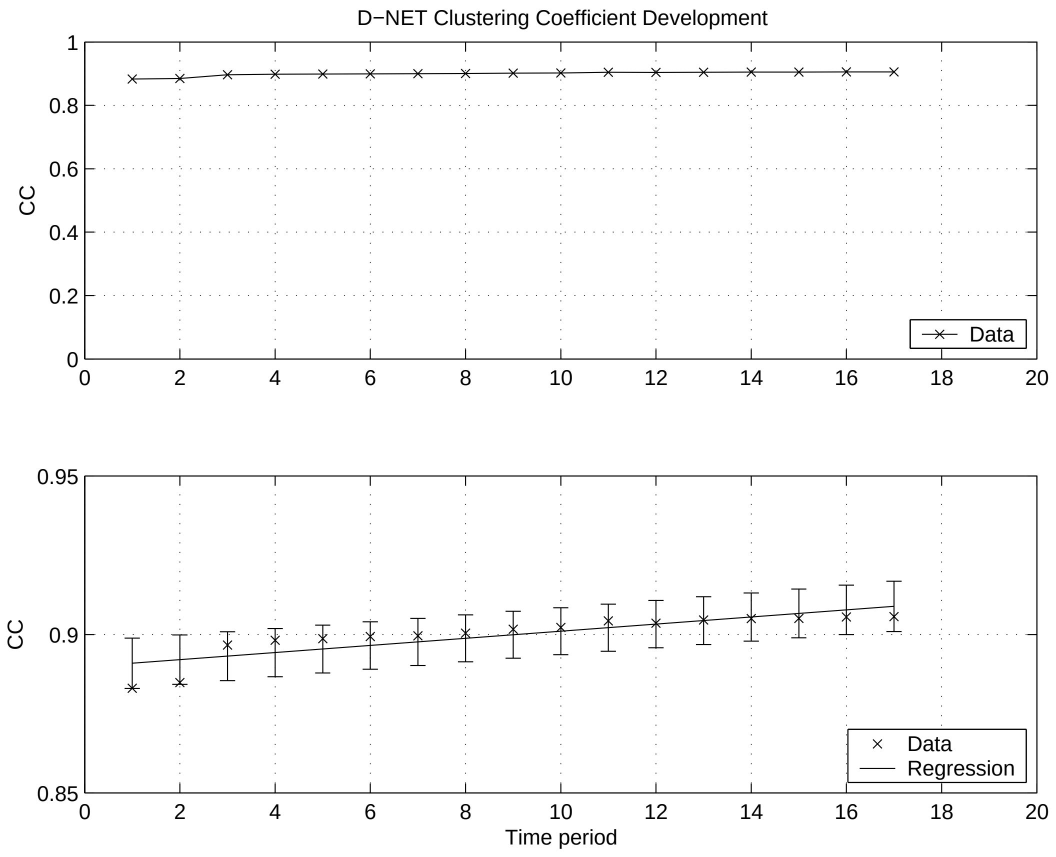


Figure 4.13. Developer Network Clustering Coefficient

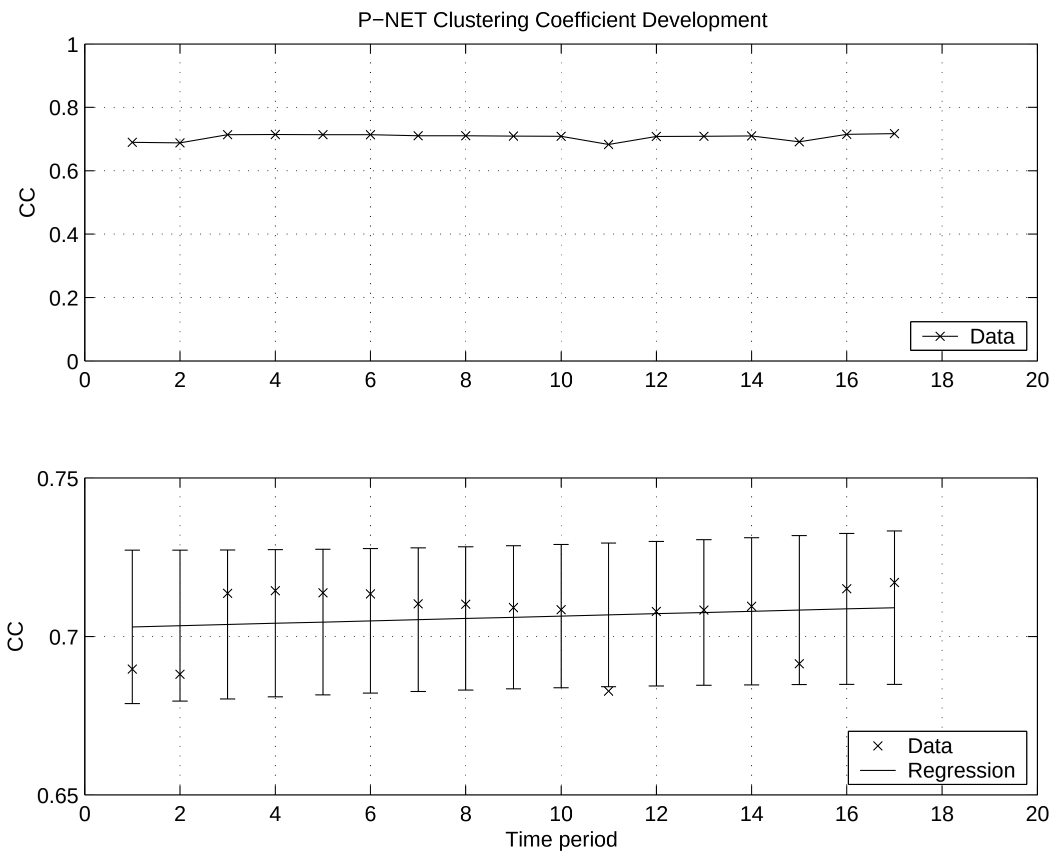


Figure 4.14. Project Network Clustering Coefficient

of its developers are collaborating on a project. Thus, with the evolution of the D-NET, more edges (collaboration relations) are formed. This will lead to an increase in the connectivity of the developer with the neighboring developers. Furthermore, this leads to the increase in the clustering coefficient.

The approximate clustering coefficient in the P-NET, shown in Figure 4.14, shares the same increasing trend with the approximate clustering coefficient in the D-NET. The lower figure shows the linear regression with 95% error bar for the developing trend of clustering coefficient for the P-NET. The slope of the regression is 0.0004.

Increases on both clustering coefficients in P-NET and D-NET suggest the network is evolving to improve its cluster property. The higher the clustering coefficient is, the more connected the network is.

Now we will apply the path analysis on average betweenness and average closeness. Figure 4.15 shows the developing trend of average betweenness in the P-NET. In the upper figure, we can observe the almost flat developing trend of the betweenness, although the overall size of the network has increased significantly during the same time period. This can suggest that the network is in a stable topology in its own evolution. In the lower figure, we magnify the Y coordinate to have a close look at the developing trend of the average betweenness. The average betweenness has a slightly increasing trend. This observation can be explained by the "rich get richer" phenomenon discovered in other complex networks such as the Internet. Although the network is constantly expanding, the hub (the node with most links) will keep gaining more connections than the others. Also, alternative hubs or regional hubs will also emerge from the network, and these hubs will increase the average betweenness of the network.

Average closeness is another measure of centrality. Figure 4.16 shows the developing trend of the average closeness in the P-NET. The upper figure is the developing trend in normal coordinates and the lower figure is the developing trend in magnified Y coordinates. We can observe the similar developing behavior of the average closeness to the average betweenness. But average closeness is more flat than the average betweenness. This is because closeness for individual node is not normalized like the betweenness for individual node. Thus, the significant increase in the overall network size will have more influence on the closeness than on the betweenness. Therefore the developing trend of the average closeness will be more flat than the developing trend of the average betweenness.

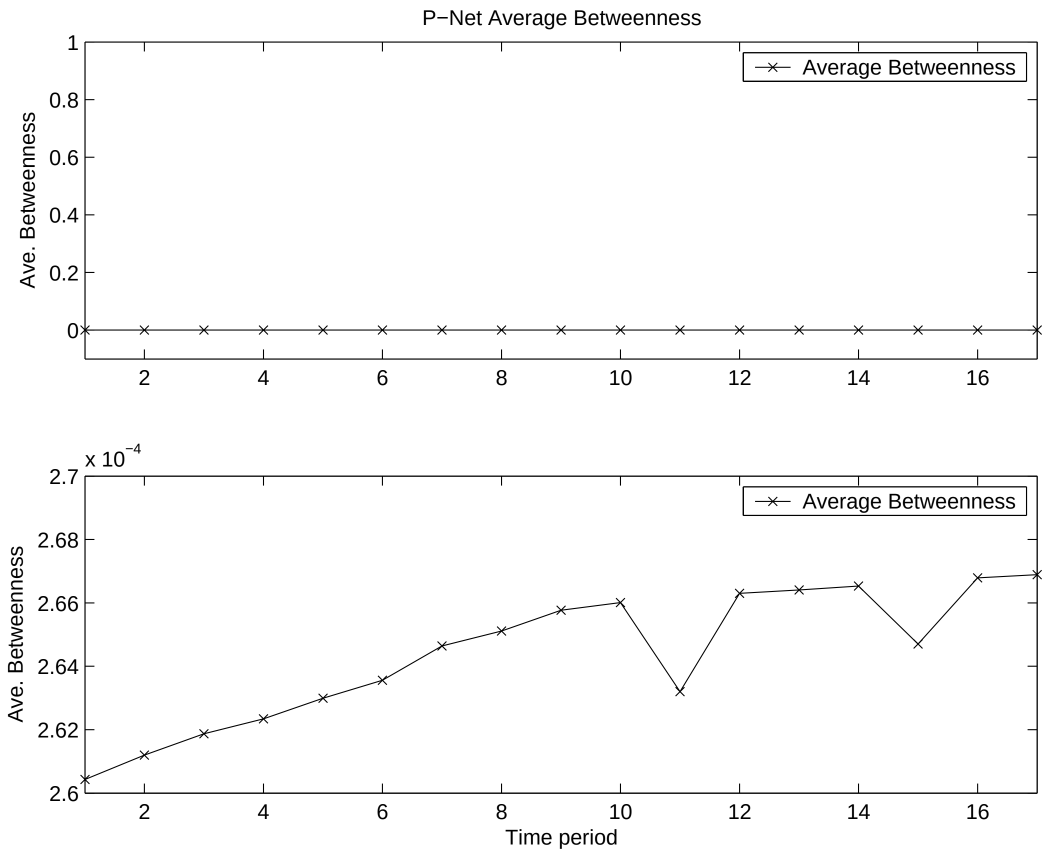


Figure 4.15. Project Network Betweenness

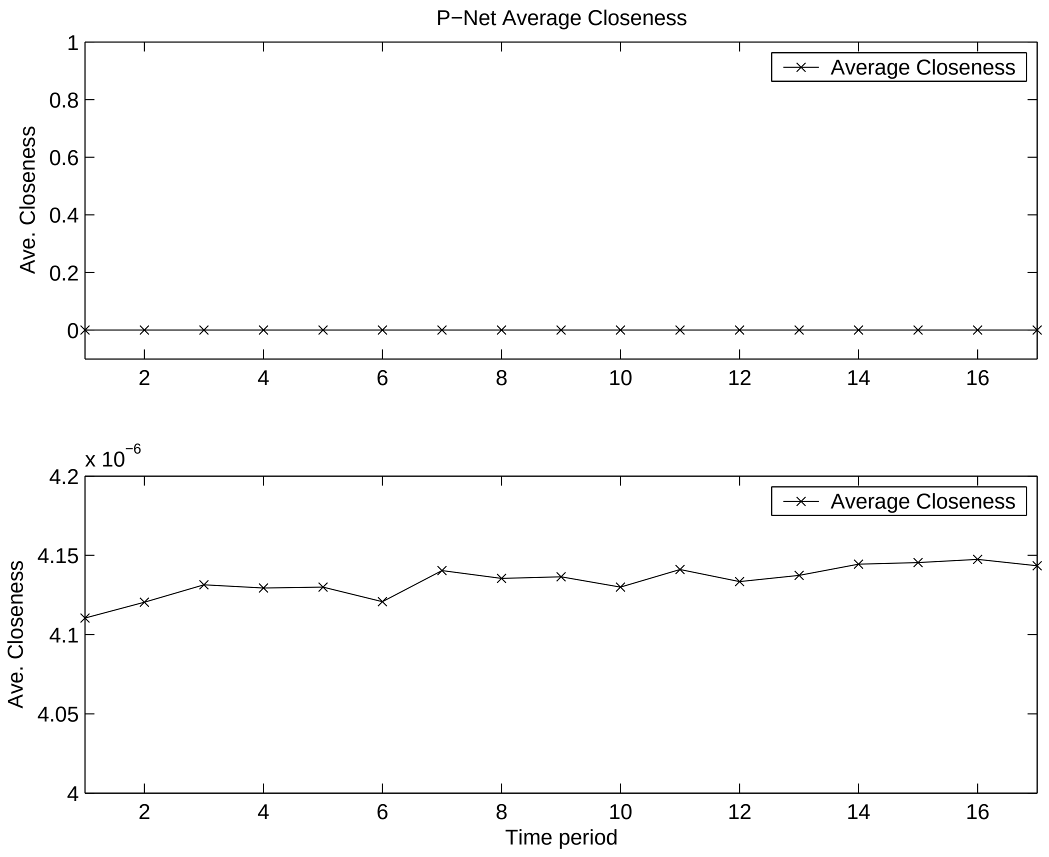


Figure 4.16. Project Network Closeness

By using path analysis, we are able to look at not only the topology of the network at a given time, but also at the evolution of the network topology, and the mutual inferences between a single entity in the network and the whole network.

One of the common discoveries in all the path analyses is life-cycle like behaviors. Most of the measures have sustained a stable level throughout the inspected period of time in this study and also have increased/decreased from the previous study carried out in the master thesis [45]. This phenomenon suggests that we have witnessed the beginning of the evolution of the SourceForge.net community and possibly the mature (stable) era of the SourceForge.net community. By closely watching the SourceForge.net community, we may have deeper insight into the evolution of the OSS community. Also, similar analyses can be applied to other evolving complex networks, such as communication networks or social networks to study the life cycle of those networks.

We also have made another discovery about the network measures. In the data mining part of this study, we observed that network measures have an important influence on the prediction of popular projects. By using the network analysis, we also observed a strong tie between the network measures and the evolution of the community networks. This discovery suggests that the network measures can be used not only as a predictor of the future of an individual entity in the network or the whole network, but also the control measures to verify possible simulation models, which we will discuss in the next chapter.

4.3 Summary

In this chapter, we studied multiple network measures of the data sets and their evolution patterns by applying multiple analyses, including structure analysis, centrality analysis and path analysis. In the structure analysis, we calculated the diameter, clustering coefficient and component distribution. The two approximate methods used to calculate the approximate diameter D and approximate clustering coefficient CC are

$$D = \frac{log(N/z_1)}{log(z_2/z_1)} + 1$$
$$CC = \frac{1}{1 + \frac{(\mu_2 - \mu_1)(\nu_2 - \nu_1)^2}{\mu_1 \nu_1 (2\nu_1 - 3\nu_2 + \nu_3)}}.$$

In the centrality analysis, we calculated four different average degrees and four different degree distributions. Also, we calculated average betweenness and average closeness. The equations to calculate the betweenness $B(v)$ and closeness $C(v)$ for individual nodes are

$$B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$$
$$C(v) = \frac{1}{\sum_{t \in V} d_G(v,t)}.$$

In the path analysis, we investigated developments of the multiple measures, including average degrees, diameter, clustering coefficient, average betweenness and average closeness. All the investigated measures are summarized in Table 4.1.

TABLE 4.1

SUMMARY OF MEASURES

Measures	D-NET	P-NET	C-NET
Average degree	Yes	Yes	Yes
Diameter	Yes	Yes	N/A
Clustering coefficient	Yes	Yes	N/A
Degree distribution	Yes	Yes	Yes
Component distribution	N/A	Yes	N/A
Major cluster	N/A	Yes	N/A
Average betweenness	Yes	Yes	N/A
Average closeness	Yes	Yes	N/A
Active entity size development	Yes	Yes	Yes
Average degree development	Yes	Yes	Yes
Diameter development	Yes	Yes	N/A
Clustering coefficient development	Yes	Yes	N/A
Average betweenness development	Yes	Yes	N/A
Average closeness development	Yes	Yes	N/A

CHAPTER 5

COMPUTER SIMULATION

In the previous two chapters, we used two computational methods to study the evolving complex networks. In this chapter, we will use another computational method – computer modeling and simulation [69, 42] – to extend our study on evolving complex networks. Simulation can be used to extend our study in the following two ways. First of all, simulation can be used to revise the results we get from previous processes. Secondly, simulation can be used to generate a "fit" model to replicate the evolution of the evolving complex network and thus predict the future development by carrying out the simulation. In our study, we defined a model for the simulation, based on the inspected measures in the empirical system, and then we used simulation to validate and improve the model [1].

5.1 Simulation

5.1.1 Simulation Methodology

We will use the same simulation procedure we proposed in the master thesis. The conceptual framework used in our research for agent-based modeling and simulation [74, 75] is shown in Figure 5.1. In the figure, there are three entities in the framework: empirical data collection, model and simulation. The model is the definition of procedures and related parameters by which we could reproduce the

[1]Materials in this chapter are partially reported in [47, 76, 49, 45].

evolution of empirical data in simulation. The simulation is an implementation of the model. We also had six relationships in the framework, each representing one kind of process: characterization, description, generation, adjustment, verification and validation. Characterization is a process of abstracting the characteristics of empirical data and generating the procedures and parameters in the model. Description manifests the underlying mechanisms of the evolution of the empirical data. Generation builds a simulation based on the given model. Adjustment modifies the model according to the feedback from the verification process. Verification is used to test the simulation's behaviors by comparing the simulation output with the empirical data and the designed model behaviors. Validation is a process of interpolating or extrapolating the simulation output and comparing the simulation output with the empirical data by attributes not used to define the model. Validation may add more rules or attributes into the model for prospective improvement. The main goal of the iterations is to get a "fit" model to describe the evolution of a complex network by simulation.

5.1.2 Verification Strategy

Verification strategy is one of the important steps in the simulation iterations [19]. By comparing against the empirical data set repeatly, verification can make the model converge to the empirical system. There are two important components in the verification step – measures and methods. The measures are the criterion for the verification and the methods are how the verification is carried out.

Statistical and network measures inspected in the previous chapters can be used in the verification to model and simulate the evolving complex networks.

1. Network measures could be used in the verification if the focus of the study is about network structure. The measures we inspected in the network analysis chapter belong to network measures: e.g., degree distribution, average diameter, average betweenness and so on. These measures describe the characteristics of the network structure and its evolution pattern.

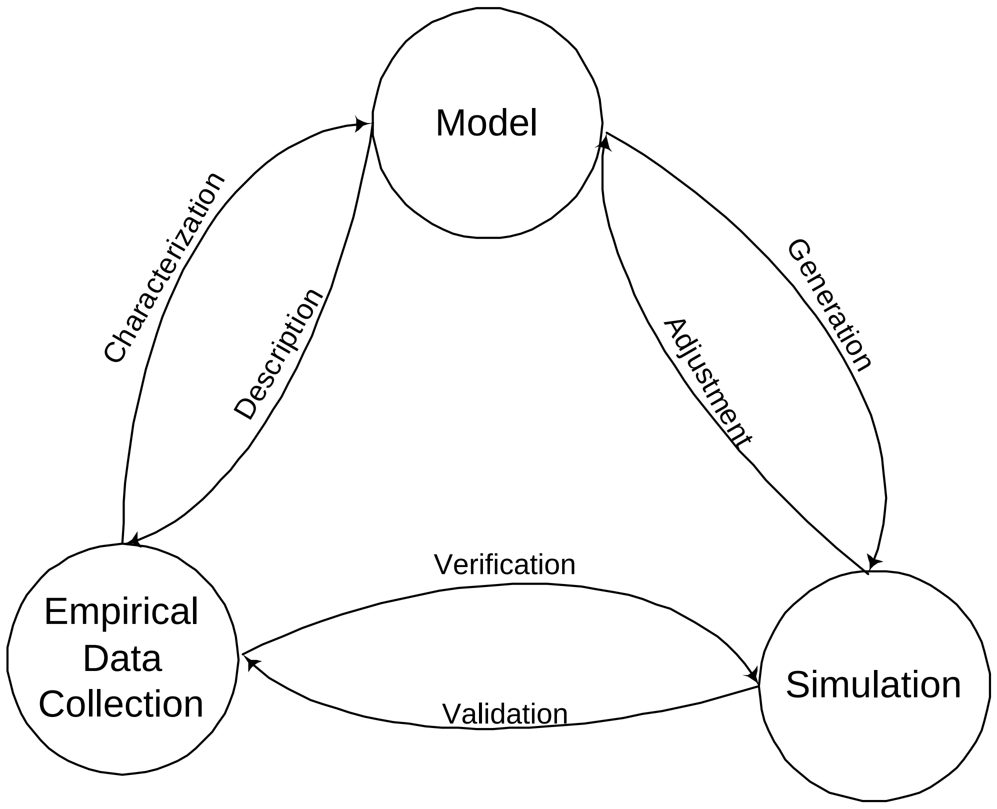


Figure 5.1. Conceptual Framework for Agent-based Modeling and Simulation

2. Statistical measures could be used in the verification if the focus of the study is about entity behaviors. The measures we inspected in the data mining chapter belong to statistical measures: e.g., monthly pageviews, downloads, number of developers in the group and so on. These measures describe the characteristics of individual entity in the network and its evolution pattern.

Depending on the actual research topic, we can choose different set of measures or even both sets in the verification step.

About the verification methods, we used two verification methods to verify – visual inspection and statistical hypothesis test. Visual inspection is the method we used in the previous study. In this study, we will also use the statistical hypothesis test method, in addition to visual inspection to increase the accuracy.

5.2 Case Study Setup

As in the previous two chapters, we used the SourceForge.net community as our case study. In the previous two chapters, we used data mining and network analysis to study the SourceForge.net community, inspected important features for individual projects and analyzed network measures about the community networks. In this chapter, we will use modeling and simulation to generate a model and simulate the evolution of the community networks, including D-NET, P-NET and C-NET. Network measures are the measures we used in the verification step. The network measures we used include:

- Degree distributions
- Average degrees
- Approximate diameter and approximate clustering coefficient
- Average betweenness and average closeness

5.2.1 Experiment Environment

Our simulations are executed on a small computer cluster, which includes multiple simulation servers and duplicate database servers. These machines were connected by a 100M(bps) switched LAN (Local Area Network).

The simulation servers, which are used to run the simulations, are all dual Xeon 3.06GHz with 2G of memory and 2G of swap. The database servers are used to host the repository database and working database. One (the repository database server) is dual Xeon 3.06GHz with 4G of memory, 6G of swap and 1T storage and the other (the working database server) is dual Xeon 3.06GHz with 2G of memory and 2G of swap.

The operating systems of the cluster are as follows: The simulation servers are Linux with 2.4.21-40.ELsmp kernel. The database servers are Linux with 2.4.21-40.ELsmp kernel with PostgreSQL 7.4. The simulation toolkit we used was Swarm and the programming language we used was Java.

5.2.2 Model Description

There are three networks in our study: C-NET, D-NET and P-NET. Since D-NET and P-NET can be transformed from C-NET, we modeled only the C-NET of the SourceForge.net community, which is the collaboration network including both developer and project as node.

The model for simulation is like a procedure definition with related parameters. We are using the same procedure definition as our previous study in my master thesis. In the model, there were two kinds of entities – developer and project. Developers, which are the active roles in real SourceForge network, will also be the active entities (agents) in our model. Projects will stored in the simulation as an accessory data structure. Our model is a time-step based simulation, which means that every agent in the simulation is triggered to act at every time step. We define the time step in our model as reflecting a single day in the real world.

At every time step, a given number of new developers, $N(ND)$, will be generated and added into the simulation. In our model, the actual number of new developers at every time step was a random number meaned at $N(ND)$. Then every existing developer (existing and new developers) in the simulation was triggered to act one by one. The procedure definitions for new developer and existing developer were different, which we will discuss separately.

- *New developer.* Every new developer is triggered to act right after it is generated. There are two possible actions for a new developer – creating or joining. Creating meant that the developer would create a new project and thus add a link from this developer to a new project. Joining meant that the developer would select an existing project according to some rules, and thus a link from this developer to the project was added. $P(CN)$ and $P(JN)$ are the two parameters that controlled the probability of creating and joining. The one exemption was that the very first developer in the simulation had to create a new project because there were no existing projects in the system yet.
- *Existing developer.* Every existing developer was triggered to act at every time step. There were four possible actions for an existing developer – creating, joining, abandoning and idling. Creating and joining were similar to the definitions for the new developer. Dropping meant that the developer would drop a randomly selected project in which he had participated. Idling meant that the developer did nothing, which is normally the most frequent action a developer will take. $P(CO)$, $P(JO)$, $P(DO)$ and $P(IO)$ are the four parameters to control the probability of creating, joining, dropping and idling. The probabilities that a developer would choose a certain action are different in different models for simulations based on different considered attributes.

Figure 5.2 illustrates the procedure implemented in our simulation. A double circle marks the start and end points of the simulation. The procedure will iterate for the preset number of simulation steps. The cloud entities in the figure are the objects used in the simulation, which include two object collections – users and projects. The square entities in the figure are the accessory data structures used in the simulation, which include the weighted project pool (user will select project in this project pool) and the user_project list (the list maintains all the user-project links in the simulation). The ellipse entities in the figure are the common functions used in the simulation, including function **new_user** (creating new users at every time step), function **user_action** (picking action to take every user at every time step) and

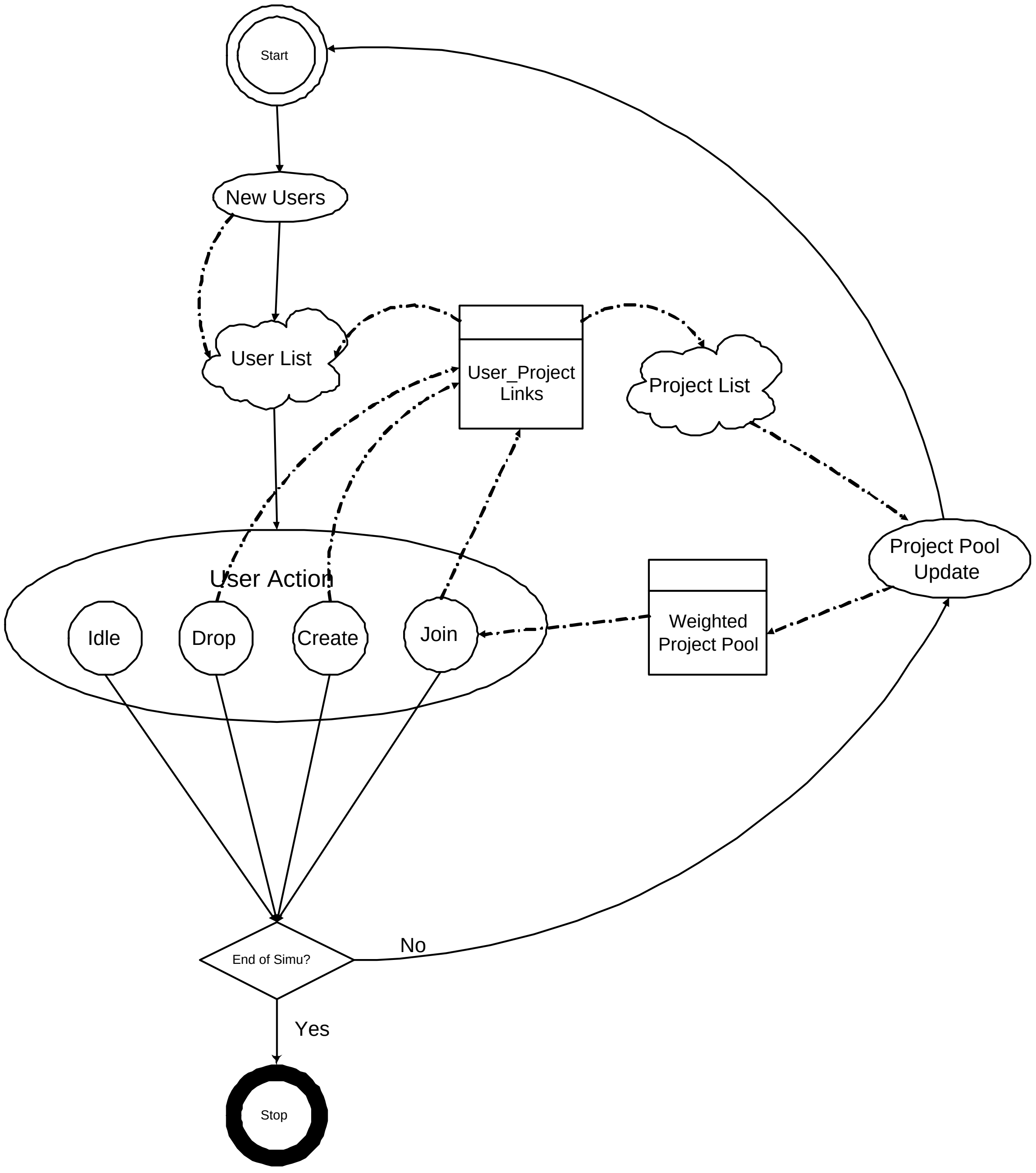


Figure 5.2. Program Flow for the Model

function **project_pool_update** (updating the weighted project pool based on the project list). The circle entities in the figure represent possible user actions, the selection of user action will be decided by a stochastic function for every user. The solid lines in the figure are the control flow in the simulation and the dotted lines in the figure are the data flow in the simulation.

This is only the skeleton of the procedure definitions in our models. Different simulation models may have different procedure definitions. Most of these differences are related to the rules a developer used to choose an action and the rules a developer used to decide which project to join. Also, the related parameters may be different for different simulation iterations. All the procedure definitions are based on the skeleton procedure described in this section. We will explain the detailed procedure definitions and related parameters for every model when we discuss that specific model.

5.2.3 Simulation Iterations

According to the proposed simulation methodology, we ran the simulations in an iterative fashion. This means we started from a basic model, running a simulation based on that model, and then verified and validated it using the empirical data. After adjusting the model and related simulation parameters, we ran the simulations again. We repeated the iterations until we got the "fit" model for the empirical data. Now we will discuss the details of the iterated simulation models one by one.

We will start with a brief introduction about the models iterated in the previous study, including adapted ER model, BA model, BA with fitness and BA with dynamic fitness. Then we will discuss new iterations in this study based on these previous models.

ER model is the first well-known complex network model and was the basis of the others. This is the first model that employed nonlinear dynamics and displays some extraordinary properties caused by phase transition [66, 6]. The first iterated model in the previous study is an adaptive ER model. The network in this model is growing, which is different from the traditional ER model. So we call it an adapted ER model. The result of the simulation yielded a network with approximate normal distributions in all the degree distributions, small and decreasing clustering coefficient, decreasing average degree and increasing diameter, which is significantly different from the network measures of the empirical data set.

BA model is the second iterated model, which is based on the scale-free network model [21, 23]. There is one improvement made in this model: preferential attachment. Preference will influence both the rules that a developer used to choose an action and the rules that a developer used to choose which project to join. The result of the simulation yielded a network with decreasing diameter, large and decreasing clustering coefficient and power laws in the degree distributions. These are similar to the network measures of the empirical data set, except the "younger upcomer" phenomenon.

BA model with constant fitness is the third iterated model. Fitness is introduced in this model to depict the "young upcomer" phenomenon. The result of the simulation not only matches most of the network measures of the empirical data set, but also is capable of explaining the "young upcomer" phenomenon.

BA model with dynamic fitness is the fourth iterated model. Instead of constant fitness we used in the last model, the fitness in this model is dynamic – decaying with time. The result of this model yielded a network matching the inspected network measures better than the previous three models.

These four models are iterated in the previous study and details can be found in the thesis [45]. Now we will start to discuss the models we iterated in this study.

The first model is an improved scale-free network model with dynamic fitness. The simulation setup we used is similar to the setup we used in the previous study. The simulation will run 1080 steps. Every 30 steps in the simulation is one simulated month, so there are a total of 36 simulated months generated by the simulation. We will compare the network measures of these 36 simulated months against the network measures we calculated from the empirical data set from December 1999 to November 2002.

As the starting point of the current study - model one, we improved the last model in the previous study - BA model with dynamic fitness in the following aspects.

The first improvement is on the stochastic procedures. Since the networks we are simulating are complex networks, we will utilize multiple stochastic procedures in the model to replicate these complex characteristics. In our models, we have the following stochastic procedures:

- Generating new developers every time step is a stochastic procedure. At every time step, we will generate a certain number of new developers and put them into the simulated "virtual" community. This number is a random number decided by a stochastic function.
- Generating fitness for new projects is a stochastic procedure. Every time a new project is created by a user, we will set some initial parameters of the project. Fitness is one of the parameters. Instead of a uniform fitness for every project, we will use a stochastic function to generate a random fitness value for every project.
- Every user picks an action at every time step, which is also a stochastic procedure. At every time step, each user will pick an action (dropping, joining, creating or idling) in turn. Which action the user will pick will be decided by a stochastic function, too.
- Picking a project to join is a stochastic procedure. When a user decides to take the "joining" action, the user will decide which project to join. This decision will be made through a stochastic function, too.

The latter two stochastic procedures are well defined by developer preferential attachment and project preferential attachment discussed in the master thesis [45].

On the other hand, the first two stochastic procedures are not well defined in the previous models. Redefinition of these two stochastic procedures will be the first improvement in our first iteration model.

The heart of the stochastic procedure is the random number generator. In the previous models, the random number generators for the first two stochastic procedures are simple random number generators based on discrete uniform distributions [28]. The PDF (Probability Density Function) is

$$F(k;n) = \frac{1}{n} \tag{5.1}$$

where k represents the discrete events and n is the number of events.

In real networks such as SourceForge.net community networks, this is not a reasonable assumption. We will replace the underlying distributions for these random number generators in the first two stochastic procedures.

For the first stochastic procedure – generating new developers every time step – a Poisson distribution is a reasonable approximation for the underlying distribution. Poisson distribution is a discrete probability distribution. Normally, it expresses the probability of a number of events occurring in a fixed period of time if these events occur with a known average rate, and are independent of the time since the last event. The new developer procedure in our simulation can be described as a poisson process. Thus, the underlying distribution of the stochastic procedure of generating new developers should be poisson distribution. The probability density function for the poisson distribution is

$$F(k;\lambda) = \frac{e^{-\lambda}\lambda^k}{k!} \tag{5.2}$$

where e is the base of the natural logarithm, $k!$ is the factorial of k and λ is the expected number of occurrences that occur during the given interval. So we replaced the underlying distribution for the random number generator in the first stochastic

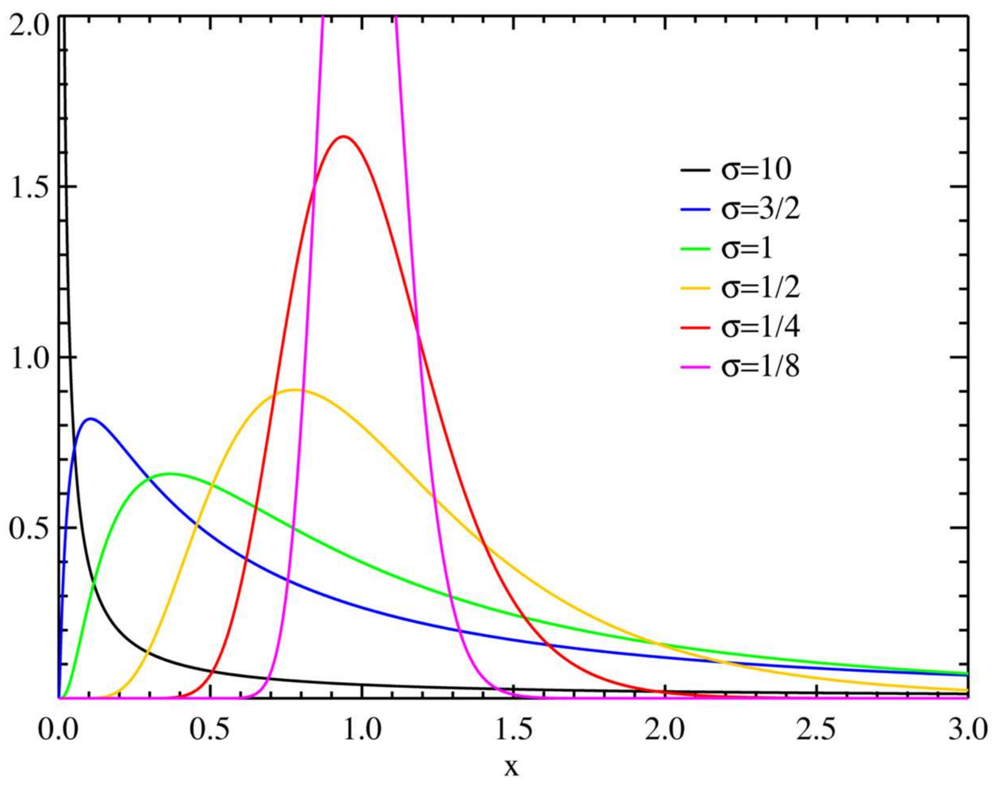


Figure 5.3. Probability Density Function for Log-normal Distribution with Different Variance σ^2 (http://en.wikipedia.org/wiki/Log-normal_distribution)

procedure with the poisson distribution with average λ at the average we inspected in the empirical data – 122.

For the second stochastic procedure – initial fitness for new project – we used log-normal distribution as the underlying distribution for the stochastic procedure. Fitness is an artificial property added into the project. Without empirical features as reference, it is difficult for us to get the exact distribution for the fitness. From intuitive understanding of the project fitness, log-normal distribution should be a good assumption of the project fitness, which should look like a skewed normal distribution as in Figure 5.3.

The probability density function is

$$f(x;\mu,\sigma) = \frac{1}{x\sigma\sqrt{2\pi}}e^{-(\ln x-\mu)^2/2\sigma^2} \tag{5.3}$$

for $x > 0$, where μ and σ are the mean and standard deviation of the variable's logarithm. The actual mean and standard deviation used in our model are picked according to the empirical data set to achieve a good "fit."

The second improvement is applying a different update procedure for the weighted project pool. In the previous models, we updated the project pool every time we made a change to any of the projects in the community. This was based on the assumption that global knowledge is always recent and available to every single user. This assumption is not practical in real networks. Normally, users will have only recent local knowledges (information about connected projects and users) and their knowledge about the global network is based on the information published by the system, which will only be updated at a preset interval. To better match the empirical data set, we applied a different update procedure for the project pool. The project pool will be updated only at certain intervals. An interval is set to be one simulated day in our simulation.

By applying these two improvements, we got our first model in this study. As we discussed before, we verified this model by comparing the network measures calculated from the simulation data set with the network measures calculated from the empirical data set. Our simulation starts from the very beginning of the C-NET (the first node) and up to three simulated "virtual" years. The empirical data set starts from January 2001 and intermittently after that. The comparison will be taken in the overlapped time period, reflecting January 2001 to December 2003 for empirical data and simulated month 13 to simulated month 36 in the simulation.

As mentioned before, we applied multiple verification methods. In addition to visual inspection, which we used in the previous study, we will also use statistical hypothesis testing [2] to verify the match of the network measures. This is because it is usually difficult and easily misleading to use only the visual inspection in comparison. For example, power law distribution and log-normal distribution are both asymptotic distributions. Without robust statistical methods, these two distributions can often look very similar. Even in a log-log plot, a log-normal distribution can often look nearly straight for certain ranges of x and y, which is the fundamental property of a power law distribution.

For a statistical hypothesis test, normally there are five important aspects:

1. Null hypothesis H_0 is the assertion we made.
2. Alternative hypothesis H_A is the alternative assertions about the problem.
3. Significance level α is the probability of incorrectly rejecting null when it is true, which is usually called "Type I error."
4. P-value p is the probability of observing the given sample result under the assumption the null is true.
5. Confidence interval ci.

There are two kinds of statistics hypothesis tests. Parametric hypothesis testing is the most popular hypothesis testing, including t-test and z-test. They are used to analyze measures which, in theory, are continuous and comply to normal distribution. The assumption about underlying normal distribution gives parameters to the hypothesis test, so it is called a parametric hypothesis test. Unfortunately, sometimes such an assumption is not warranted. To be less sensitive to these assumptions, a non-parametric hypothesis test is introduced. A non-parametric hypothesis test makes only mild assumptions about the data, and is appropriate when the distribution of the data is not normal or cannot be guaranteed as normal. On the other hand, non-parametric hypothesis tests are less powerful than the parametric hypothesis testing for normally distributed data. In our case study, we cannot make the

[2]Hypothesis testing is a procedure for determining if an assertion about a characteristic of a population is reasonable.

assumption that the data set is normally distributed, so we will use non-parametric hypothesis testing to verify the similarity of the measures from the simulation and the empirical data set.

One of the tests we used is **Chi-squared** χ^2 **test** for categorized data. χ^2 test evaluates the results by reference to the χ^2 distribution. It tests a null hypothesis that the relative value of the observed events follow a specified distribution. The categories must be mutually exclusive. In our case study, the null hypothesis could be "the approximate diameters in certain months follows the diameters witnessed in the corresponding months in the empirical data." The statistics χ^2 is calculated by finding the difference between each observed and expected value, squaring them, dividing each by the theoretical value, and taking the sum of the results:

$$\chi^2 = \sum_{i=1}^{n} \frac{(O_i - E_i)^2}{E_i} \tag{5.4}$$

where O_i is an observed value and E_i is an expected (empirical) value. Then we will consult the χ^2 distribution with *df* (degree of freedom) of the problem. Given the statistical significance value, we can decide to reject or accept the null hypothesis. We will use the χ^2 test to verify the network measures like average degrees, approximate diameter, approximate clustering coefficient, average betweenness and average closeness.

The other test we used is the **Kolmogorov-Smirnov** K-S **test** for determining whether two underlying distributions of two data sets differ, or whether an underlying distribution differs from a hypothesized distribution, in either case based on finite samples. The two samples K-S test is one of the most useful and general non-parametric hypothesis tests for comparing the underlying distributions of two samples, as it is sensitive to differences in both location and shape of the underlying distribution of the two samples. To show the K-S test mathematically, we will first introduce the empirical distribution function F_n for n observations y_i as

$$F_n(x) = \frac{1}{n}\sum_{i=1}^{n}\{0 \quad {}^{1 \quad if \ y_i \leq x,} \; otherwise. \tag{5.5}$$

Then the two one-sided Kolmogorov-Smirnov test statistics are given by

$$D_n^+ = max(F_n(x) - F(x)) \tag{5.6}$$

$$D_n^- = max(F(x) - F_n(x)) \tag{5.7}$$

The details about K-S test can be found at [29]. In our case study, we used the K-S test to verify the degree distributions in the simulated data set matching the corresponding distributions in the empirical data.

The first network measure we inspected is the average degree and their development through the interested time period. The results are shown in Figure 5.4. As we discussed in the last chapter, there are four different degrees in the SourceForge.net community: developer degree in the C-NET, project degree in the C-NET, developer degree in the D-NET and project degree in the P-NET. All the figures share the same coordinate configuration. The X coordinate is the time and every data point represent a single month. The time range is [1,24], which is reflects the time range [January 2001, December 2003] in the empirical data set and time range [month 13, month 36] for the simulated data set. Since we have no complete empirical data during the time period, we are only comparing the simulation data set against the existing empirical data set. There are a total of 15 eligible months in which we can compare the simulation data against the empirical data, which are the X coordinates of the data points in the figures. The Y coordinate is the average degree value. All the four average degrees yield similar data points in empirical data and simulated data.

In the figure, we zoomed in the Y coordinates for a detailed comparison. We found that the developing trend of the average degrees of the simulated data is

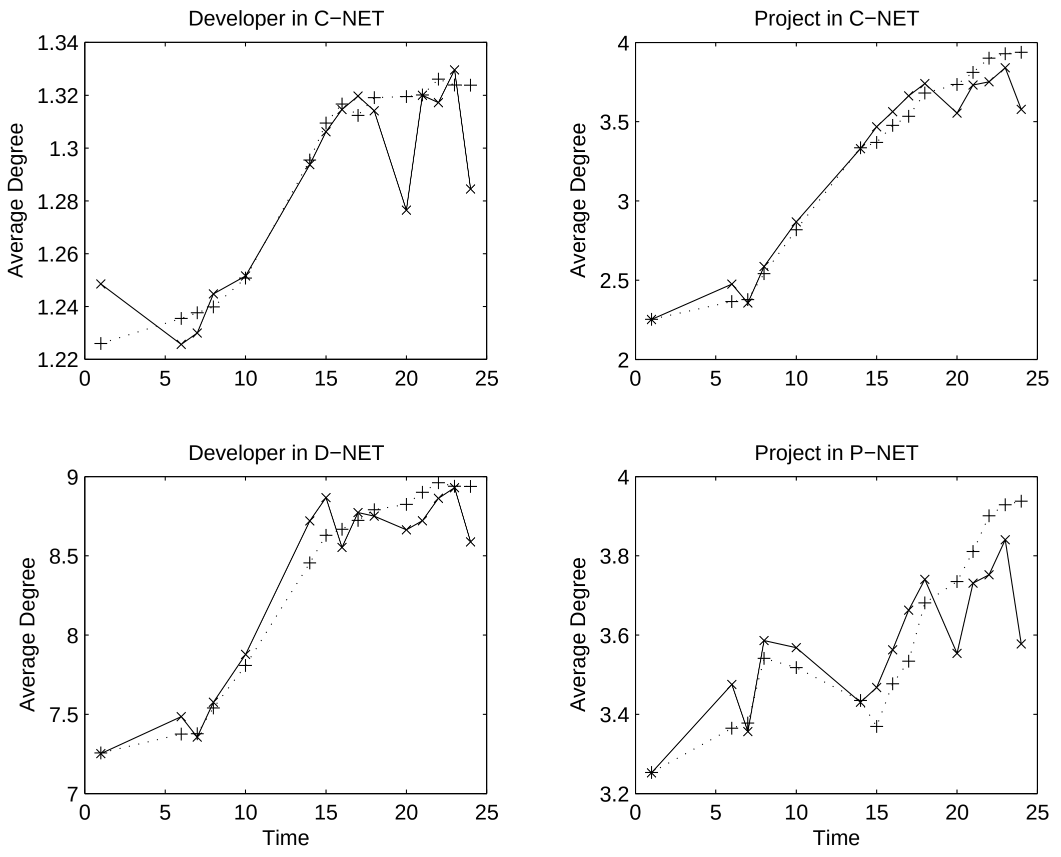


Figure 5.4. Average Degrees Comparison: the left upper figure shows the developer degree in the C-NET, the right upper figure is the project degree in the C-NET, the left lower figure is the developer degree in the D-NET and the right lower figure is the project degree in the P-NET. All the figures share the same coordinate configurations. The X coordinate is the time and the Y coordinate is the value of the corresponding average degree. 'x' represents the empirical data and '+' represents the simulated data.

smoother than that of the empirical data. This mismatch is expected since we cannot include all the features that can influence the evolution of the real complex network into our simulation models.

We verified the similarity of the average degrees and their developing trend using visual inspection. We also verified this by using the statistical hypothesis test. The hypothesis test is this:

- H_0: The average degree developments share the same trends in the empirical data and the simulated data.
- H_A: The average degree developments do not share the same trends in the empirical data and the simulated data.

We calculated the χ^2 values for all the average degrees. The χ^2 for the developer degree in the C-NET is 0.0034, the χ^2 for the project degree in the C-NET is 0.0723, the χ^2 for the developer degree in the D-NET is 0.041 and the χ^2 for the project degree in the P-NET is 0.07. By choosing the significance level α at 0.01 and the *df* value at 14, the critical value is 4.66. These values are all well below the critical value. Thus, the null hypotheses are accepted. So the average degrees in the simulated "virtual" data set have no statistically significant difference from the average degrees in the empirical data set.

Next, we verify the simulation results using the approximate diameter and approximate clustering coefficient. Figure 5.5 shows the diameter development in the P-NET and the clustering coefficient development in the P-NET. Using visual inspection, we found the developments of diameter and clustering coefficient from the simulated data to be similar to those of the empirical data, except that the measures of the empirical data have more jagged trends than the simulated data.

We also used the χ^2 test to verify the similarity of the developments in the simulated data and the developments in the empirical data with α as 0.01 and *df* as 14. The χ^2 value for the diameter development in the P-NET is 0.0146 and

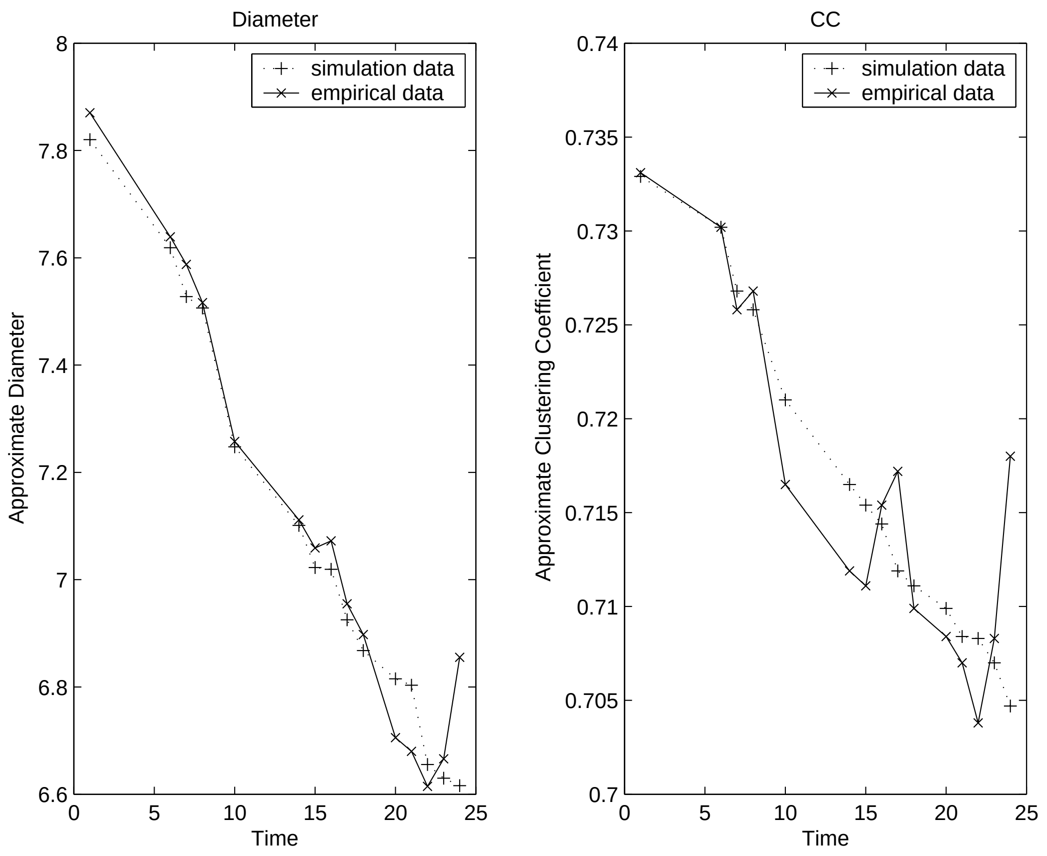


Figure 5.5. Diameter and Clustering Coefficient Comparison: The left figure is the diameter development trends in the P-NET and the right figure is the clustering coefficient development trends in the P-NET. Both figures share the same coordinate configuration, where X is the time and the Y coordinate is the value of the corresponding measures. 'x' represents the empirical data and '+' represents the simulated data.

the χ^2 value for the clustering coefficient in the P-NET is 4.13e-04, which are well below the critical value at 4.66. So the null hypotheses are accepted and, thus, the simulated data yields similar diameter and clustering coefficient development trends as the empirical data. We also verified the measures for the D-NET, which have the same results.

The next network measures are the betweenness and closeness. We show the measures for the P-NET in Figure 5.6. By using the χ^2 test, we found the χ^2 value for the diameter development in the P-NET is 5.8e-08 and the χ^2 value for the clustering coefficient in the P-NET is 1.64e-09. Both of the χ^2 values are well below 4.66, which is the critical value using α as 0.01 and df as 14. So the null hypotheses are accepted and, thus, the simulated data yields similar average betweenness and average closeness development trends as the empirical data.

Finally, we also verified the simulation model using the degree distributions. The degree distributions for the C-NET are shown in Figure 5.7.

We used the K-S test to test whether the degree distributions in the simulated data set are the same as the degree distributions in the empirical data. The hypothesis is defined as:

- H_0: The degree distribution in the simulated data is the same as the corresponding degree distribution in the empirical data.
- H_A: The degree distribution in the simulated data is not the same as the corresponding degree distribution in the empirical data.

The p-value for the developer degree distribution is 0.3754 and the p-value for the project degree distribution is 0.3979. Using the common significance level at 0.05, the hypotheses are accepted. Thus, both the developer degree distribution and project degree distribution in the simulated data are the same as the corresponding distributions in the empirical data.

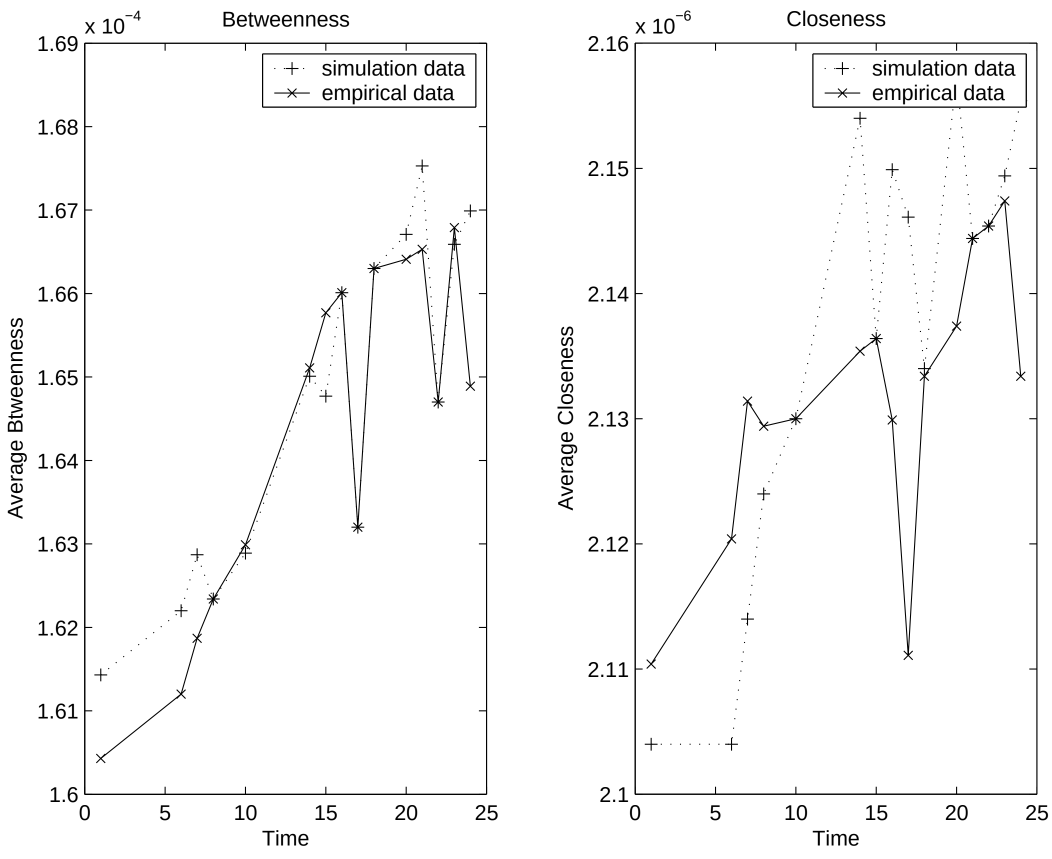


Figure 5.6. Betweenness and Closeness Comparison: The left figure is the average betweenness development trends in the P-NET and the right figure is the average closeness development trends in the P-NET. Both figures share the same coordinate configuration, where X is the time and the Y coordinate is the value of the corresponding measures. 'x' represents the empirical data and '+' represents the simulated data.

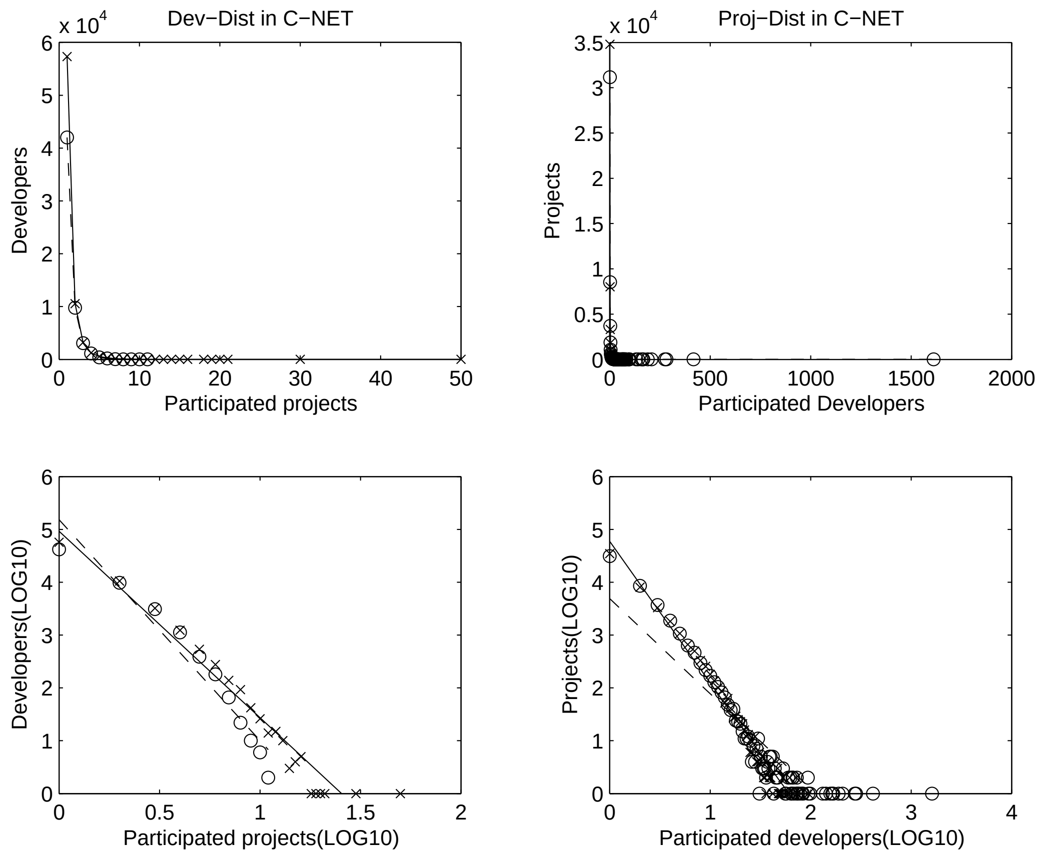


Figure 5.7. Degree Distributions Comparison: The left upper figure is the developer degree distribution in normal coordinates, the left lower figure is developer degree distribution in log-log coordinates, the right upper figure is project degree distribution in normal coordinates and the right lower figure is project degree distribution in log-log coordinates. All figures share the same coordinate configuration, where X is the time and the Y coordinate is the value of the corresponding measures. 'x' represents the empirical data and 'o' represents the simulated data.

Until now, we have matched all of the network measures inspected, including approximate diameter, approximate clustering coefficient, average degrees, average betweenness and average closeness between the simulated data and the empirical data by both the statistical hypothesis test and visual inspection. And we also matched the degree distributions by the statistical hypothesis test. However, by visual inspection, we can still find differences between the degree distributions of the empirical data set and the simulated data set. For the developer degree distribution in the C-NET, the simulated data set has a significantly shorter tail than the empirical data. For the project degree distribution in the C-NET, the simulated data set has a significantly longer tail than the empirical data. These deficiencies (the busiest developer in the simulated data set participated in just 11 projects instead of 50 in the empirical data, and the biggest project in the simulated data set has more than 1500 developers instead of 95 in the empirical data) are not reasonable for the real-world community such as SourceForge.net. So we iterated more models to make these deficiencies.

The second iterated model is trying to make up for the deficiencies discovered in the previous model. To achieve this, we added a new parameter into the model – user energy. In the previous models, we introduced the "fitness" factor into the project preference. On the other hand, user preference is based only on the number of projects he/she participated. The deficiency in the previous model could be the result of lacking user "fitness," so in this model we introduced a "fitness" factor for user preference, which is called user energy.

In the real community such as SourceForge.net, different users have different influences on the community. This difference is decided not only by the number of projects in which he/she participated, but also by the ability of that user. Some new user could have more contributions than some old user who participated in multiple projects. We describe this using the user energy factor. We define the energy as a constant parameter γ_i for every user. Thus every time a new user(say, user j) is created, an energy γ_j was added to the system, where γ_i is chosen for a uniform continuous distribution between $(0, 1]$. This energy level can be described as the probability that a user will take an action instead of idling during every time step.

After simulating with this model, we also verified this model using the same measures and methods as the previous model. This model yielded similar results for most of the network measures, except the degree distributions. The degree distributions of this model are shown in Figure 5.8.

The biggest project in the project distribution decreased from more than 1500 to below 500, which is improved, compared to the same measures in the empirical data. However, the developer distribution still has significant deficiency at the tail, compared to the same measures in the empirical data.

Further iteration on the modeling introduced dynamic user energy into the model. Through the previous iterations, we still have some deficiencies in the degree distributions in our simulated data. In this model, we proposed a dynamic energy factor in the model, which allows the energy for every user to decay with respect to time and adjusts according to the roles the user is taking in various projects. In the model, we made the energy decay linearly by time and, meanwhile, it can be self-adjusted by taking roles and responsibilities or giving up roles and responsibilities in the community.

By using this dynamic energy, we expected to catch the details of the possible influence of a user to the related project, even the whole community, and then generate better matches in both the developer distribution and the project distribution. The results of the simulation of this model still have the similar average degrees, approximate diameter and clustering coefficient, average betweenness and average

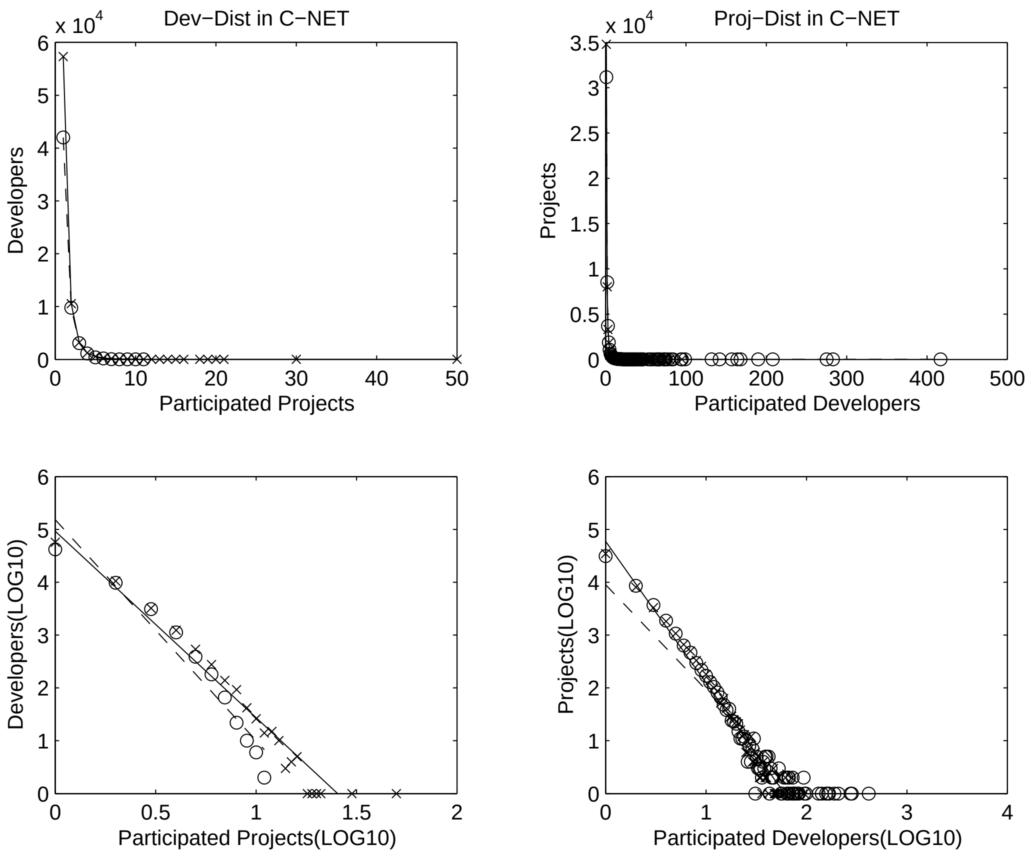


Figure 5.8. Degree Distributions Comparison: The left upper figure is the developer degree distribution in normal coordinates, the left lower figure is developer degree distribution in log-log coordinates, the right upper figure is project degree distribution in normal coordinates and the right lower figure is project degree distribution in log-log coordinates. All figures share the same coordinate configuration, where X is the time and the Y coordinate is the value of the corresponding measures. 'x' represents the empirical data and 'o' represents the simulated data.

closeness as the empirical data. And the results of the simulation of this model improve the matches of developer distribution and project distribution. These distributions comparison are shown in Figure 5.9.

In the figures, the busiest user is increased to 43 projects, which is more close to the the empirical data, while the size of the largest project is kept below 200. This is a more reasonable tail for the SourceForge.net and close to the empirical data set. Thus, model three is able to replicate all the network measures we used in the verification by both visual inspection and statistics hypothesis test methods.

Through the iterations, we discovered that degree distributions are the most sensitive network measures in the model iterations. Other network measures such as average degrees, approximate diameter, approximate clustering coefficient, average betweenness and average closeness have similar values and behaviors in the models we iterated from the BA model in the previous study to the third model in the current study. This discovery can be used in other similar simulation iterations, where degree distributions should be used as the most closely watched network measures to verify a simulation result against an empirical data set.

After analyzing and iterating the models, our third model reproduced all the inspected network measures we chose of the empirical data and captured all the phenomenon we wanted to highlight in the community and can be used to simulate the community and possibly predict the future of the SourceForge.net community by extending the simulations.

5.3 Summary

In this chapter, we used simulation methods to assist in understanding the topology and evolution of the SourceForge.net developer collaboration network. First, we discussed a framework for simulation-related study. In the case study, we used

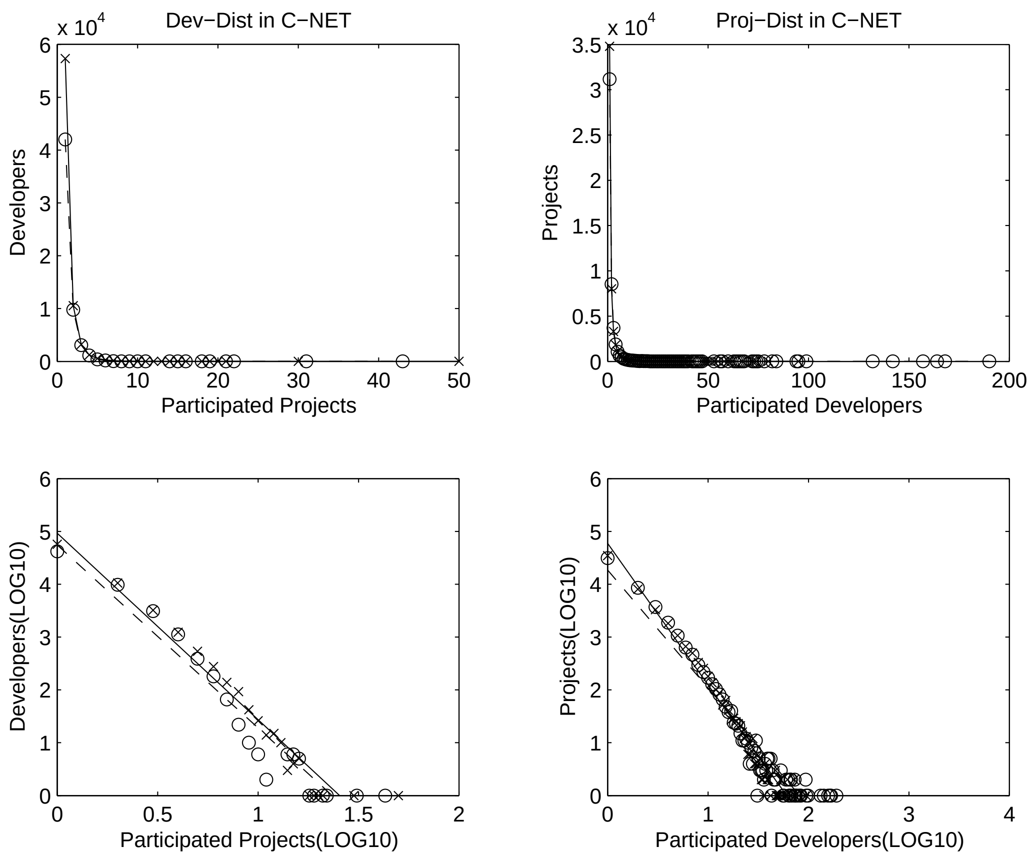


Figure 5.9. Degree Distributions Comparison: The left upper figure is the developer degree distribution in normal coordinates, the left lower figure is developer degree distribution in log-log coordinates, the right upper figure is project degree distribution in normal coordinates and the right lower figure is project degree distribution in log-log coordinates. All figures share the same coordinate configuration, where X is the time and the Y coordinate is the value of the corresponding measures. 'x' represents the empirical data and 'o' represents the simulated data.

this framework to develop more models for the SourceForge.net community. We iterated three more models from our previous study to generate the “virtual” SourceForge.net. In the simulation iteration, we verified the result by comparing the measures from the simulated data and measures from the empirical data using both visual inspection and statistical hypothesis test. The final result of this chapter is a model to simulate the SourceForge.net with matched network measures. Using this model, we are able to simulate the SourceForge.net community and possibly predict the future development of the community by simply extending the simulation time. The inspected measures in this chapter are summarized in Table 5.1.

TABLE 5.1

SUMMARY OF SIMULATION RESULTS

Model	Measure	Expected Pattern	Observed Pattern
ONE	Developer Distribution	Power law (large tail)	Power law (small tail)
	Project Distribution	Power law (small tail)	Power law (large tail)
	Average Degrees	Increasing	Increasing
	Clustering Coefficient	Decreasing	Decreasing
	Diameter	Decreasing	Decreasing
	Average Betweenness	Decreasing	Decreasing
	Average Closeness	Decreasing	Decreasing
TWO	Developer Distribution	Power law (large tail)	Power law (small tail)
	Project Distribution	Power law (small tail)	Power law (reasonable tail)
	Average Degrees	Increasing	Increasing
	Clustering Coefficient	Decreasing	Decreasing
	Diameter	Decreasing	Decreasing
	Average Betweenness	Decreasing	Decreasing
	Average Closeness	Decreasing	Decreasing
THREE	Developer Distribution	Power law (large tail)	Power law (large tail)
	Project Distribution	Power law (small tail)	Power law (small tail)
	Average Degrees	Increasing	Increasing
	Clustering Coefficient	Decreasing	Decreasing
	Diameter	Decreasing	Decreasing
	Average Betweenness	Decreasing	Decreasing
	Average Closeness	Decreasing	Decreasing

CHAPTER 6

RESEARCH COLLABORATORY

Collaboratory is, as a commonly accepted definition by Cogburn [33], "more than an elaborate collection of information and communications technologies; it is a new networked organizational form that also includes social processes; collaboration techniques; formal and informal communications; and agreement on norms, principles, values, and rules."

As described by Cogburn, we designed our collaboratory as not only a repository, where the raw data set and related analytical toolkits are stored, but also a community, where researchers sharing the same interest can work and discuss with each other about the related research. Nowadays, there are many similar research collaboratories existing to support various research. NCBI [1], a collaboratory for molecular biology research, is one of them. Although these collaboratories have different requirements and support different research domains, they usually share the similar structure. We will discuss the popular structures first of all.

6.1 Collaboratory Design

Common structures include two-tier, three-tier and four-tier hierarchies. We will focus our introduction on the three-tier hierarchy, which is used in our research collaboratory design. These three tiers are:

[1]http://www.ncbi.nlm.nih.gov/.

1. Data tier (back-end data storage): Since the research collaboratory is a data-centrical system, a data repository is at the back end of the system to store all the related information. Normally, database is the implementation of the data repository at the back end. The design and implementation of this tier define the reliability, integrity and performance of the collaboratory.

2. Presentation tier (front-end web interface): This tier is responsible for formatting the information and delivering it to the end user. It relieves the user of concerns in regard to data representation within the logic tier or the data tier. Web interface is one of the commonly used front-end tier. The design and implementation of this tier define the availability and usability of the collaboratory.

3. Logic tier (middle-layer service provider): This tier implements most of the processes and functions provided in the presentation tier and usually these processes and functions are based on the information stored in the data tier. Most of the programming language supporting networking can be used in this tier. The design and implementation of this tier define the functionality of the collaboratory.

For each tier, there are several major tasks to be accomplished. We will discuss these tasks for every tier in the following subsections.

6.1.1 Data Tier

As we discussed before, data tier includes the repository where the information will be stored. To ensure the reliability, integrity and performance of the collaboratory, there are several tasks that should be fulfilled in this tier.

Designing an appropriate schema is the first task. Schema defines the structure, content and, to some extent, the semantics of the information stored in the repository. Different research collaboratories will have different requirements for the data storage, thus different data schema. The first task of the data tier is designing and maintaining a good schema for the repository to well-serve the collaboratory.

Designing an appropriate backup strategy is the second task. Backup is a method to keep a copy of the data so that it could be restored after a data-loss event. For the repository of a research collaboratory, the information stored in the repository is vital to the collaboratory. So maintaining the reliability of the data is very important to the collaboratory. Normally, at least periodic backups should be engaged to improve the data recovery reliability. So the second task of the data tier is designing a good backup strategy for the repository.

Designing an appropriate load-balancing technique is the third task. Load balancing is a technique to spread jobs across multiple computers, processes, disks or other resources in order to get optimal resource utilization and decrease overall computing time. Also, when the load of the system is getting heavy, load balancing is also needed to limit the workload of the database to avoid overloading the database. So the third task of the data tier is designing a load balancing technique to control access to the back-end repositories.

6.1.2 Presentation Tier

The presentation tier is the interface interacting with the end user (developer). Intuitive and rich interface can help users access and use the information stored in the collaboratory and can also promote the availability and usability of the collaboratory. The following basic functions should be included in the presentation tier.

First of all, various access methods to the data repository are the most important functions the interface should provide to the user. Controlled query is one of the common access methods, so that users can query the back-end repository directly to retrieve information they want. Programmable access is another method that should be provided for more sophisticated use of the back-end repository.

Secondly, documentation and reference are important functions of the interface. Documentation is important for users to understand the background and context of the information provided by the collaboratory, so users can use the information more efficiently and wisely. Reference is also important to a collaboratory, especially a research collaboratory. With sufficient knowledge of peer studies, users can improve their own research.

Finally, community support is also another important function of the interface. The collaboratory provides a “virtual” community for the users sharing the same research interests to interact with each other around the world and thus promote efficient teamwork.

6.1.3 Logic Tier

Logic tier is a set of applications and processes dealing with user requests from the presentation tier, processing and generating relative queries to the database, and formatting the returned result and sending it back to the presentation tier. This tier can improve the scalability of the collaboratory by separating the logic tier from the presentation tier and data tier.

The major functionality of this tier is implementing every function provided in the presentation tier. So the design and implementation of this tier is closely related to the functions the presentation tier provides to the user.

The three-tier hierarchy is a common structure for a research collaboratory. Two-tier hierarchy and four-tier hierarchy are also available for collaboratories with less or more complexity.

6.2 Case Study

As the previous chapters, we used the SourceForge.net as our case study. We built a research collaboratory for the Open Source Software Research based on the monthly database dumps we got from SouceForge.net. We used the three-tier hierarchy to implement our collaboratory. Details about the hierarchy used in our collaboratory are shown in Figure 6.1.

We will explain these three tiers in our collaboratory in the rest of this section.

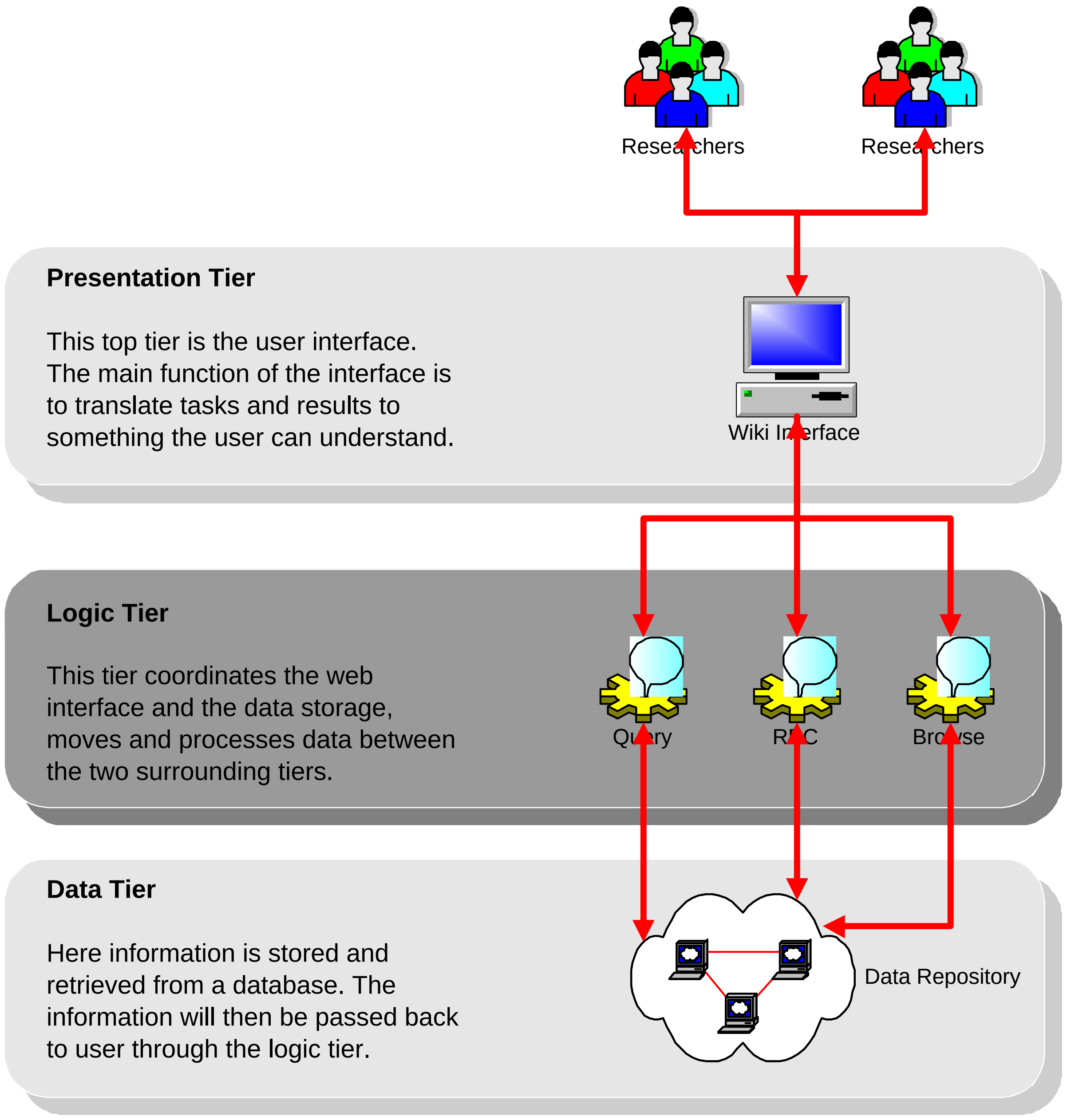

Researchers
Researchers
Presentation Tier
This top tier is the user interface. The main function of the interface is to translate tasks and results to something the user can understand.
Wiki Interface
Logic Tier
This tier coordinates the web interface and the data storage, moves and processes data between the two surrounding tiers.
Query
RDC
Browse
Data Tier
Here information is stored and retrieved from a database. The information will then be passed back to user through the logic tier.
Data Repository

Figure 6.1. Three-tier Hierarchy for Our Collaboratory

6.2.1 Data Tier

First of all, we will explain the data tier design. As discussed in the previous section, there are three major tasks in designing the data tier – repository schema, backup strategy and load balancing.

Every month we get a database dump from SourceForge.net with privacy scrutiny. The database dump is a SQL file with all the commands needed to restore the database to the designated state.

This data repository in our collaboratory is different from common data repository because of the following two reasons.

First of all, there are significant overlaps between two adjacent database dumps. Each of the database dumps is a full database dump for the month, instead of an incremental database dump. This means that part of the data replicated from the previous month. So if we simply restore these database dumps over the previous database dump, much information in the previous database dump will be overlapped by the current database dump. Then we will lose the evolving history information during monthly restoration.

Secondly, the pattern of updates is different from that in many commercial database applications. For example, the ticketing database records information about booking airline tickets. In the ticketing database, there is much record-removing and record-changing because of cancellation and rebooking. In our database, the situation is different. As new observations are made, they are added to the database, and only if an observation is found to be incorrect will his/her entry be changed or removed. Therefore, the database tends to grow monotonically. As well as adding more instances of the same kind of data, new kinds of data items and new properties are introduced and new kinds of relationships between data items are added.

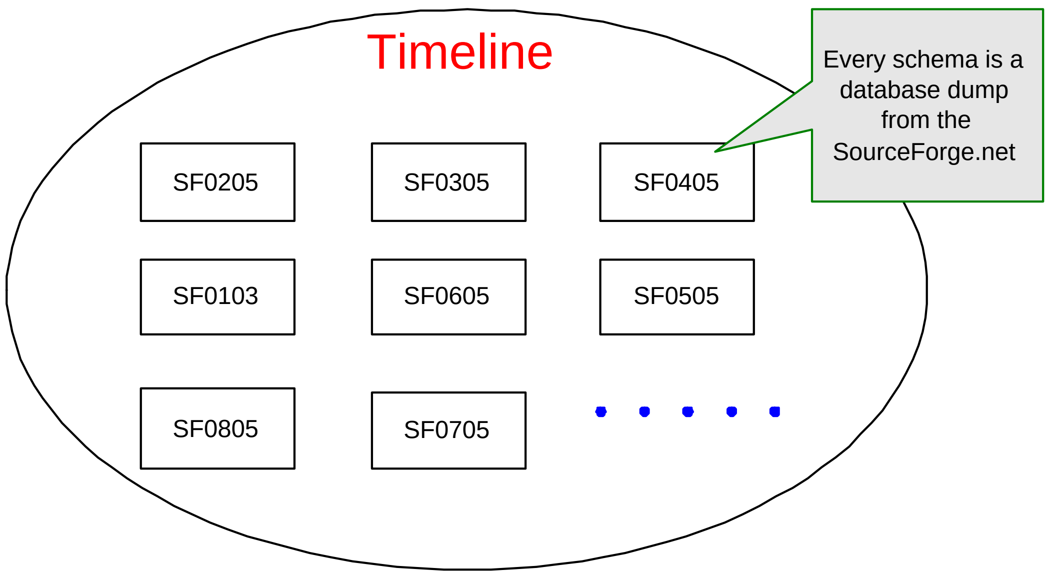


Figure 6.2. Data Schema for the Repository

Therefore, it is to be expected that the database schema used to describe our repository will itself develop as the database grows. Based on these, a multiple-layer data schema is used in the data repository shown in Figure 6.2.

Timeline is the name of the top level – the database, where all the data dumps from SourceForge.net are stored. In the database, there are multiple schemas [2]. Every schema represents a single month, and stores the database dump from SourceForge.net of the corresponding month. These schemas are named from the month of the corresponding database dump. For example, schema "SF0103" stores the database dump of January 2003. In every schema, we copy the intact table definitions from the corresponding database dump as the table definitions.

Another important task of the data tier is the backup strategy. Since our repository should expand only every month, a new database dump is restored. A periodic

[2]A logic level between database and table.

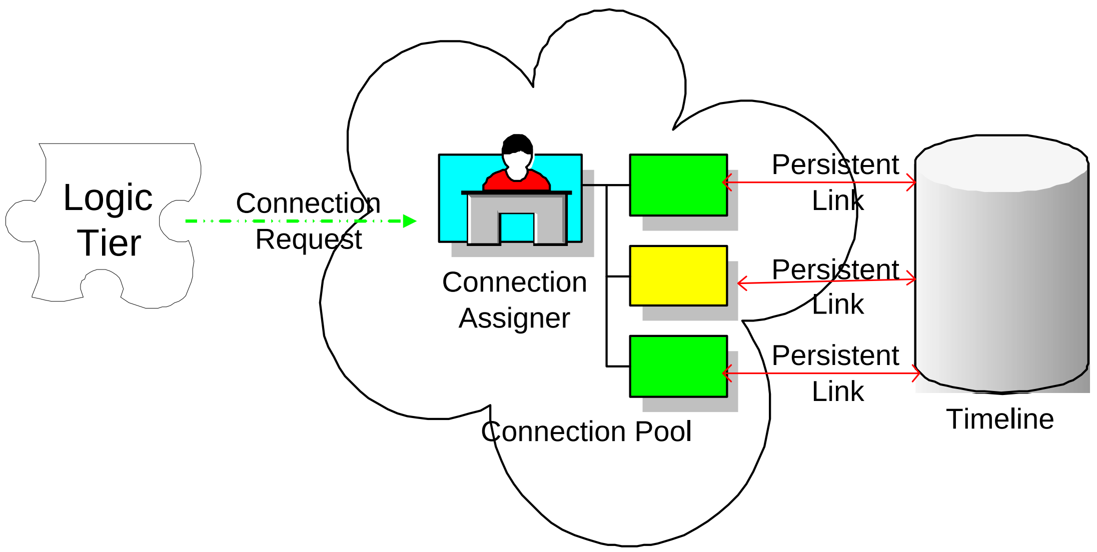


Figure 6.3. Connection Pool Design

(monthly) full repository backup is good enough to keep the repository fully recoverable. This backup action can be initialized by cron job or manually. In addition to this periodic full repository backup, we also keep a full backup of the database dump files in case there is potential error during data loading.

As to load balancing, we applied connection management only for our collaboratory. This is because currently our back-end database is only a single server hosting the "timeline" database and load balancing is not necessary. Instead, connection management is needed to relieve the back-end server from a possible heavy load of queries [3]. In order to mitigate the possible overloading issue for the database server, a connection pool mechanism is engaged. The connection pool mechanism is shown in Figure 6.3.

The connection pool entity acts like a daemon running on the database server,

[3]According to our statistics, in June 2006, we received 16,947 queries, returned 13,343 files in total size of over 26G from just 24 active users. Further information about the web site utilities can be found in the Appendix C.2.

which keeps a predetermined number of persistent links to the database – "timeline." This predetermined number can be adjusted, according to the processing power of the database server. From the logic tier point of view, this connection pool entity works like a virtual back-end database. Every request to access the database will be submitted to the connection assigner, and the assigner will assign an idle link to the request procedure, if available, and recollect the link when the work is done. If every link is busy, the request will be queued at the connection assigner for the next available link to access the real back-end database. By engaging the connection pool mechanism and setting the appropriate number of persistent links, it is guaranteed that the database server won't be swamped by connections and every requests will be queued at the connection pool without suffering from bad performance or dropping. Also, by adjusting the number of persistent links, we can tune up the performance of the back-end database.

6.2.2 Presentation Tier

Presentation tier is the user interface of the research collaboratory. As we discussed in the last section, community support and repository access are two of the major functions of the presentation tier. However, normally these two functions have different requirements for authorizations. For the repository access, tight authorization is needed to avoid information abuse. On the other hand, community support needs relatively loose authorization and, actually, sometimes anonymous users' contributions are vital to a "virtual" community (e.g., online bulletin board service). Thus, in our presentation tier, public community service and restricted repository access should be provided with layered authorization. All users, including anonymous users, can access the community; registered users are allowed to post on the community and only specially authorized users can access the back-end repository through the web interface or programmable interface.

For the public community service, we used wiki technology [4]. The wiki engine (the collaborative software) we are using is *MediaWiki*, which is also the wiki engine used by wikipedia [5]. We chose wiki to implement our public community service for the following reasons:

1. Good content management: wiki has shared, and encouraged, several features with generalized content management systems that are used by enterprises and communities-of-practice. These features include articles that are editable through the web browser; each article has one-click access to a discussion page and the name of the article is embedded in the hyperlink.

2. Easy to use: A defining characteristic of wiki technology is the ease with which pages can be created and updated. Each article in the wiki can be created or edited at anytime by anyone. Also, the wiki engine provides linking and searching functions to locate an article easily. By far, the most common wiki systems are server-side wiki systems. In essence, the edit, display and control functions are provided on the server through the wiki engine that renders the content into an HTML-based page for display in a web browser. The client of the wiki does not need special support to browse a wiki site.

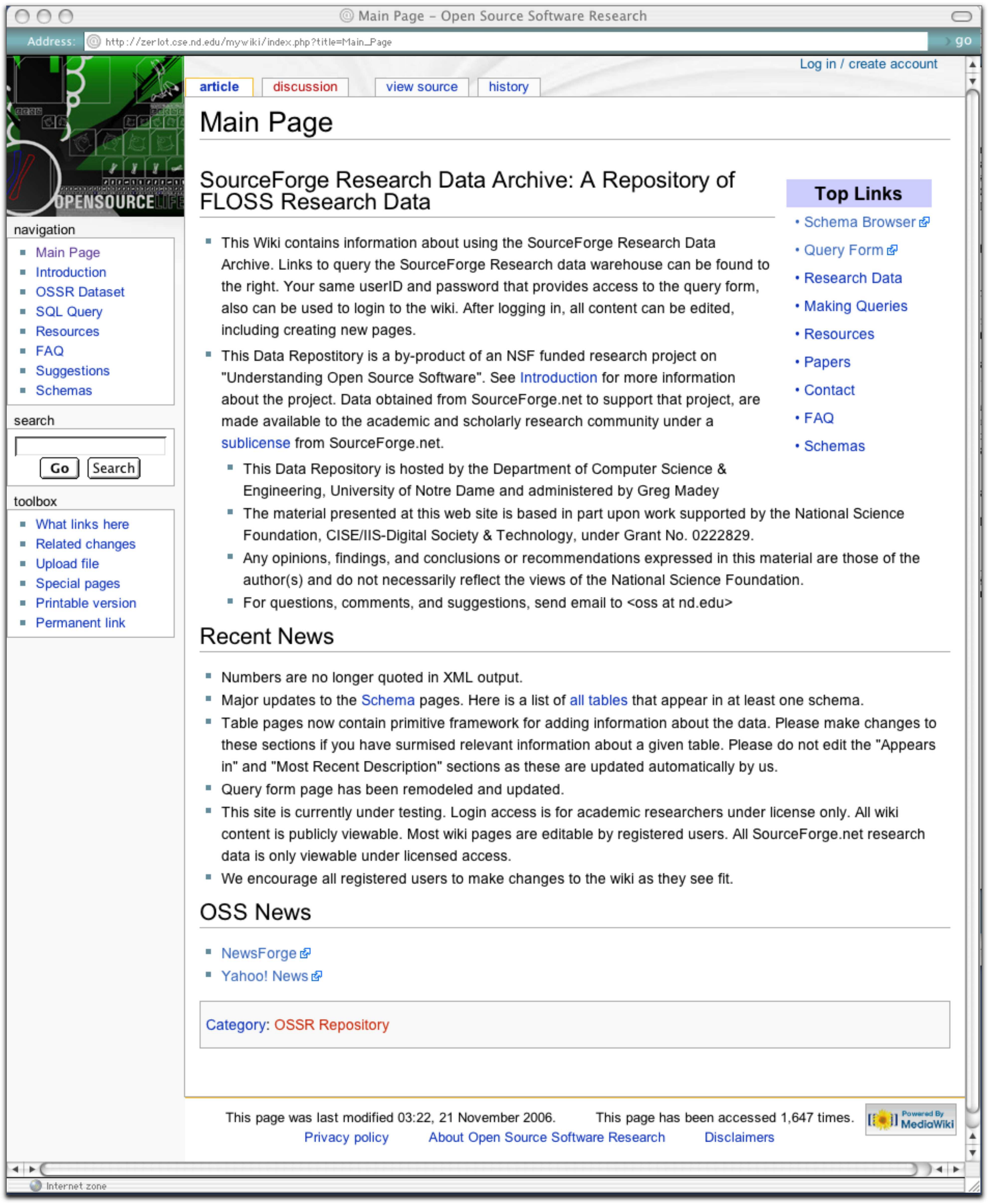
3. Complete change control: Wiki is generally designed with the philosophy of making it easy to correct mistakes, rather than making it difficult to make them. Thus, while wiki is very open, it provides a means to verify and validate recent additions to the body of the pages by changing logs. And each article provides one-click access to the history/versioning page, which also support version differencing and retrieving prior versions.

For these reasons, we implemented our user interface using wiki technology. Figure 6.4 is a screen shot of the user interface of the collaboratory using the wiki technology.

Another major function of the presentation tier is the restricted repository access. The web interface module is implemented using Perl script and ODBC. There are two sub-interfaces provided in the web interface – database query interface and the schema browser interface. Authorized users can submit queries directly to the

[4]A *wiki* is a type of website that allows visitors to easily add, remove, or otherwise edit and change some available content, sometimes without the need for registration. The ease of interaction and operation make a wiki an effective tool for collaborative authoring.

[5]http://www.wikipedia.org.



Main Page

SourceForge Research Data Archive: A Repository of FLOSS Research Data

- This Wiki contains information about using the SourceForge Research Data Archive. Links to query the SourceForge Research data warehouse can be found to the right. Your same userID and password that provides access to the query form, also can be used to login to the wiki. After logging in, all content can be edited, including creating new pages.
- This Data Repostitory is a by-product of an NSF funded research project on "Understanding Open Source Software". See Introduction for more information about the project. Data obtained from SourceForge.net to support that project, are made available to the academic and scholarly research community under a sublicense from SourceForge.net.
  - This Data Repository is hosted by the Department of Computer Science & Engineering, University of Notre Dame and administered by Greg Madey
  - The material presented at this web site is based in part upon work supported by the National Science Foundation, CISE/IIS-Digital Society & Technology, under Grant No. 0222829.
  - Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.
  - For questions, comments, and suggestions, send email to <oss at nd.edu>

Recent News

- Numbers are no longer quoted in XML output.
- Major updates to the Schema pages. Here is a list of all tables that appear in at least one schema.
- Table pages now contain primitive framework for adding information about the data. Please make changes to these sections if you have surmised relevant information about a given table. Please do not edit the "Appears in" and "Most Recent Description" sections as these are updated automatically by us.
- Query form page has been remodeled and updated.
- This site is currently under testing. Login access is for academic researchers under license only. All wiki content is publicly viewable. Most wiki pages are editable by registered users. All SourceForge.net research data is only viewable under licensed access.
- We encourage all registered users to make changes to the wiki as they see fit.

OSS News

- NewsForge
- Yahoo! News

Category: OSSR Repository

This page was last modified 03:22, 21 November 2006. This page has been accessed 1,647 times.
Privacy policy About Open Source Software Research Disclaimers

Figure 6.4. Screenshot of the Wiki Interface for "SourceForge Research Data Archive: A Repository of FLOSS Research Data" [1].

backed repository through the database query interface. And the schema browser provides a dynamic interface to browse the database schema of the repository, including available schemas and table definitions. Details about the implementation of these functions will be discussed in the next subsection about logic tier.

As we discussed before, authorization is required before a user can submit a query to the back-end repository. In our implementation, we used a single-user database to provide uniformed user authorization for both the public wiki site and the restricted repository access. This uniformed user authorization is easy to manage and is more secure.

6.2.3 Logic Tier

Logic tier includes all the processes implementing every function in the presentation tier. In our implementation, access to the data repository is one of the major functions needed to be implemented in the logic tier. There are two kinds of access methods to the back-end repository provided in our collaboratory – web access and programmable access.

Web access is the simple method to access the back-end repository and it is used to implement the database query interface discussed before. In case the query can be described as a single query and submitted through a web page, web access is the preferred access method. In our implementation, authorized users can submit queries through a provided web page; the result will be saved in the user directory and be made available for download by the user. We use files instead of returning the result directly to the user to avoid unexpected network delays and other problems. Also, by saving the result in a file, we can avoid duplicate requests to mitigate the burden on the back-end repository.

In our previous discuss, we mentioned that the schema of the data repository is evolving. It is difficult for us to keep publishing up-to-date schema on the wiki site and it is impossible for the user to query the data repository without the current knowledge about the schemas. To solve this problem, we provided a dynamic schema browser, which is simply another web query system about the data schemas only. By using the schema browser, users can pull the current schema directly from the back-end repository and then query the back-end repository according to the current schema.

With the schema browser, a typical user query can be divided into two steps. The first step is to retrieve the related table names and attribute names. The next step is to generate the query using the information retrieved in the first step and submit the query. This two-step approach provides a complete solution for users for submitting queries to access the dynamic evolving back-end repository.

For the result file, multiple formats are provided for different usages of the results. Delimited plain text is one of the available formats and users can pick the different delimiters. The other format provided is XML format for seamless integration with other information-exchange services.

Although most of the query for the data repository can be completed through the simple web access, there are some situations where simple web access is not enough. For example, if the user has a sophisticated request than cannot be described using single query or the user needs use procedures (queries with control structure like loop and condition) in the query, programmable access is needed. Currently only web access is implemented, but web service functionality is proposed to implement the programmable access to the back-end repository.

This collaboratory is implemented for the Open Source Software Research community. We provided access to the data repository of the SourceForge.net database dump and published our research about SourceForge.net in the collaboratory. Although it is a fairly new community, the statistics show that the usage of the collaboratory is promising (Appendix C.2). We expect this collaboratory could contribute to the research of Open Source Software.

6.3 Summary

In this chapter, we discussed the design and implementation of a research collaboratory. The collaboratory includes a repository for the research-related information and a community for the researchers to cooperate with each other in the research. The collaboratory provides a method to share information, publish the research results, discuss it with peer researchers and promote future research in the domain.

We implemented our research collaboratory for OSSR (Open Source Software Research) using three-tier hierarchy.

For the data tier, we designed a self-evolving data schema, periodic backup strategy and a connection control mechanism to ensure the reliability, integrity and performance of the collaboratory.

For the presentation tier, we built a community service for the collaboratory using wiki technology and a web access module (restricted query and dynamic schema browser). We also proposed to use web service to implement the programmable access module.

For the logic tier, we implemented the functions in the web access module, including restricted query and the dynamic schema browser, using Perl and ODBC.

CHAPTER 7

CONCLUSION

7.1 Summary of Results

Study of the evolving complex networks has vital practical importance. It can help in efficient indexing and searching of the Internet; stability analysis for biological and communication networks; epidemic threshold studies for social science and understanding the networking influence in software engineering practice. Because of its practical importance, many research topics are proposed to study the evolving complex network in computer science, physics and social science domain. Our research is focused on using computational discovery methods to study the structure and evolution of the evolving complex networks.

In this dissertation, we proposed a computational discovery methodology to study the evolving complex networks. The methodology is a cyclic procedure with four different processes: data mining, network analysis, computer simulation and research collaboration. In the data mining process, we used traditional methods such as feature selection and distance-based clustering to discover potential patterns in the complex network. Meanwhile, we also applied novel distribution-based clustering methods in the data mining process. In the network analysis process, we applied multiple analyses, including structure analysis, centrality analysis and path analysis, on the evolving networks. In the computer simulation process, we used iterated simulation to generate a "fit" model to simulate the evolution of empirical evolving complex networks. Finally, we proposed a design for research collaboratory supporting research about evolving complex networks.

We also applied this proposed methodology on a real world problem – a study of the Open Source Software community in the dissertation. SourceForge.net development community is the designated community we studied. By applying the methodology on the SourceForge.net community network, we gained many useful insights into the structure and evolution of the community. In the data mining process, we identified the important features that contributed to the success of a project and generated a novel activity distribution-based predictor to predict the "popularity" of a single project by applying the distribution based clustering algorithm. In the network analysis process, we expanded our previous study by applying more analysis on the more complete data set. By using the network analysis, we had more discoveries about possible life cycle behaviors for both the whole network and individual entities in the network. In the computer simulation process, we iterated multiple models based on the network measures we inspected in the network analysis process and generated a "fit" model to simulate the evolution of the SourceForge.net community network to possibly predict the future development of the SourceForge.net community by extending the simulation time. In the collaboration process, we designed and implemented a research collaboratory for the OSS research, storing the accumulated database dumps from SourceForge.net and providing access and analysis tools to the users through the collaboratory. This collaboratory has had good utilization since it was established.

7.2 Limitations and Future Work

Our work has some limitations, but by overcoming these limitations, there are several research topics that can be further studied. These limitations and possible future topics are as follows:

1. By using the distribution-based data mining method, we had an activity distribution-based predictor to foresee the "popularity" of a single project. However, because of the limited computational resources, we did not perform a larger scale (over the whole data set), longer-term (extended time span of the inspected projects) analysis of the algorithm on the available data set. Such an analysis could help in verifying and improving the method, and generate more interesting predictors for the SourceForge.net community.

2. We had interesting discoveries about the network structures and evolutions, especially the widely existing life cycle behaviors. Nevertheless, we still did not observe the substantial evidence of the life cycle behavior – the complete life cycle of the network evolution, because Sourceforge.net is still actively evolving. By keeping monitoring, accumulating and analyzing the data set from the SourceForge.net community, we could gain more insights about the life cycle behaviors about the network evolution.

3. Our modeling and simulation yielded good results to model and simulate the SourceForge.net community, but the modeling and simulation practice is limited to the SourceForge.net community because of the lack of information from the other OSS communities. Applying similar modeling and simulation practice on other OSS community could verify and expand our result.

4. As an experimental collaboratory, we hosted our collaboratory on a single server. To avoid excessive load to the server, we engaged connection control and deployed only the web query and schema browser modules. However, we have witnessed large numbers of requests since the launch of the online collaboratory (details about the utilization statistics can be found in Appendix C.2). Redesigning the back-end repository as a server cluster, engaging load balancing in the application layer, deploying programmable access module (web service) in the front-end interface could significantly improve the availability, performance and functionality of the collaboratory.

APPENDIX A

SUPPLEMENTS FOR NETWORK ANALYSIS

In this appendix, we included the formal proof of the approximate method for graph diameter [102].

Given

$$G_0(x) = \sum_{\infty}^{k=0} p_k x^k \tag{A.1}$$

as the generating function for p_k, which is the probability of a vertex have k degree, we have the following properties:

$$\sum p_k = G_0(1) = 1 \tag{A.2}$$

$$p_k = \frac{1}{k!}\frac{d^k G_0}{dx^k}|_{x=0} \tag{A.3}$$

$$z_1 =< k >= \sum_k k p_k = G_0'(1) \tag{A.4}$$

$$\tag{A.5}$$

We define p_k^1 as the probability of the degree of the vertices that we arrived at by following a randomly chosen edge. So

$$P_k^1 = \frac{k p_k}{\sum_k k p_k} \tag{A.6}$$

Thus, we define the generating function for p_k^1 as

$$G_1(x) = \frac{\sum_k k p_k x^{k-1}}{\sum_k k p_k} \tag{A.7}$$

We have the similar properties for $G_1(x)$ as for $G_0(x)$:

$$\sum p_k^1 = G_1(1) = 1 \tag{A.8}$$

$$p_k^1 = \frac{1}{(k-1)!}\frac{d^k G_1}{dx^(k-1)}|_{x=0} \tag{A.9}$$

$$\tag{A.10}$$

Using power property of the generating function, we can get the generating function for the distribution of the number of second neighbors of the original vertex as:

$$G_2(x) = \sum_k p_k[G_1(x)]^k = G_0(G_1(x)) \tag{A.11}$$

Thus, by the moment property of generating function, we get

$$z_2 = [\frac{d}{dx}G_0(G_1(x))]_{x=1} = G_0'(1)G_1'(1) \tag{A.12}$$

Furthermore, we can get every z_m as

$$z_m = G_0'(1)[G_1'(1)]^{m-1} = [\frac{z_2}{z_1}]^{m-1}z_1 \tag{A.13}$$

The overall network size, N, is

$$N = 1 + \sum_m z_m \tag{A.14}$$

when $N \gg z_2 \gg z_1$, we can get

$$N \approx z_m \tag{A.15}$$

Combining the Equation A.13 and Equation A.15, when $z_m \to N$ and $m \to D$, where D is the diameter of the network, we get

$$N = [\frac{z_2}{z_1}]^{D-1}z_1 \tag{A.16}$$

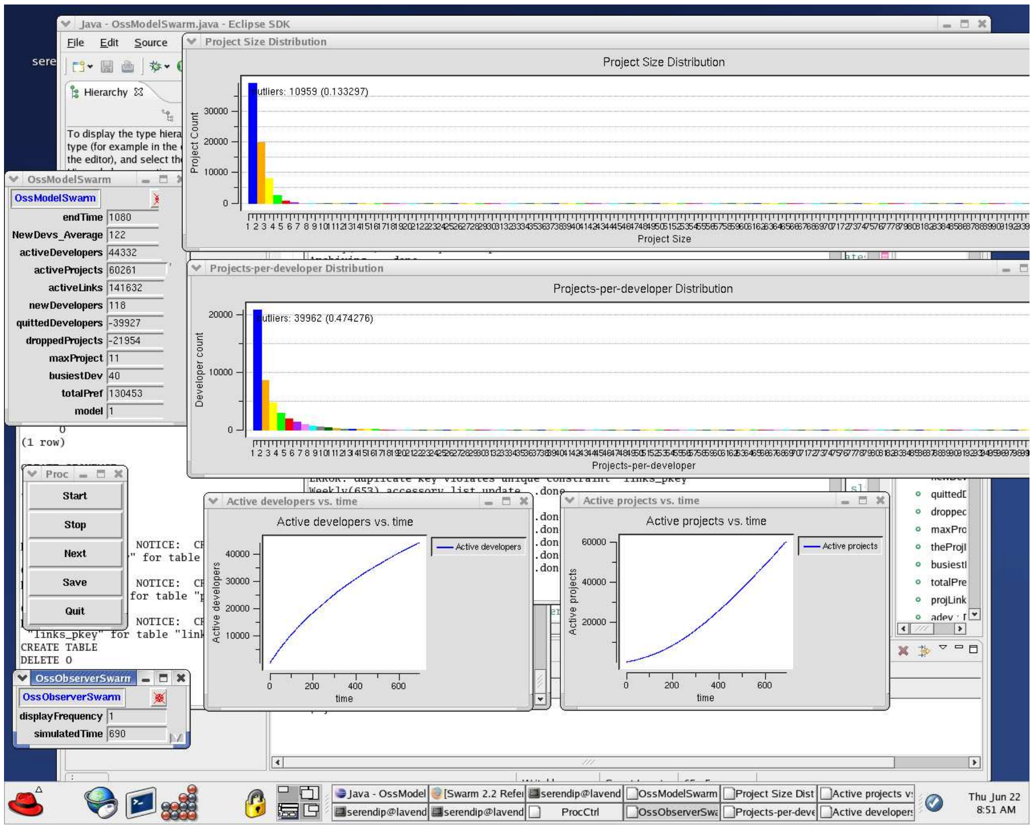
So, we can get

$$D = \frac{log\frac{N}{z_1}}{log\frac{z_2}{z_1}} + 1 \tag{A.17}$$

APPENDIX B

SUPPLEMENTS FOR SIMULATIONS

In this appendix, we included a screenshot of the simulation execution as shown in Figure B.1. The lower two figures are the developing size of the developer body and the project body. And the upper two distribution figures are the dynamically generated distributions for developer size distribution and project size distribution. They show power law properties.



OssModelSwarm

endTime	1080
NewDevs_Average	122
activeDevelopers	44332
activeProjects	60261
activeLinks	141632
newDevelopers	118
quittedDevelopers	-39927
droppedProjects	-21954
maxProject	11
busiestDev	40
totalPref	130453
model	1

OssObserverSwarm

displayFrequency	1
simulatedTime	690

```
(1 row)
NOTICE:  CR
" for table
NOTICE:  CR
for table "
NOTICE:  CR
"links_pkey" for table "link
CREATE TABLE
DELETE 0
```

Figure B.1. Screenshot of the Running Simulation

APPENDIX C

SUPPLEMENTS FOR COLLABORATORY

C.1 Timeline Database

This database is the repository of all the database dumps from SourceForge.net. Since the database dump is monthly, there are always overlaps among the dumps. We used the schemas to save the different database dumps. The hierarchy we used in the repository is three levels. The highest level is the database, which is the timeline database. The middle level is the schemas, which are used to store different monthly database dumps. And inside the schema are the tables from a single database dump, which is the lowest level in the hierarchy. We will explain the repository according to this hierarchy.

C.1.1 Database

The first level is the database. We maintained two databases for the repository. They are timeline and userinfo. The DBMS is PostgreSQL 7.4.3.6. The userinfo database stores the information about the users authorized to access the repository. Since we are still in the beta state, we handle the applications only manually. The collected user information includes user name and user email. This database is just an accessory database. The timeline database is the main database, where the monthly database dumps from SourceForge.net are stored.

C.1.2 Schema

Since the monthly dumps we get from SourceForge.net are database dumps, we utilized the schema feature to store multiple dumps in one single database – timeline database. We use different schema to store database dumps for different months. The name convention of the schema is sf(mm)(yy), (mm) and (yy) stands for the month and year of the specified database dump. For example, the database dump for January 2005 is stored in schema sf0105. The data schema inside each schema is decided by the related database dump, respectively. Since SourceForge.net is evolving itself, the data schemas in different schema may be different.

C.1.3 Table

As we discussed in the last section, the data schema for different schema is different, but they all include the general data schema. In this section, we will explain the general data schema (table definitions) in the timeline database. Basically, there are three types of tables in every schema. First, there are data tables, which provide all the information for their website. There are over 40 tables in this category. Some of the tables have very intuitive names, like "groups" (stands for the projects in SourceForge.net),"user" (stands for the users in SourceForge.net), "group_type" (stands for the group types existing in SourceForge.net) and "user_group" (stands for the user group relationships in the SourceForge). The other tables can be further categorized. These tables normally have a name starting with a prefix. These common prefixes will be explained one by one.

- The prefix "artifact" stands for the activities in a project. These activities include bug reports, feature requests, file releases, file downloads, patches, forum posts and CVS activities. The tables with this prefix store all the information related to these artifacts in the specific project.
- The prefix "forum" stands for the discussions in a project. The tables with this prefix store the information related to forum discussions.

- The prefix “foundry” stands for the Source Forge community. The tables with this prefix store the information related to the global information of the Source Forge community.
- The prefix “frs” stands for the file releases of a project. The tables with this prefix store the information related to the file releases of the projects.
- The prefix “people” stands for the users in the community. The tables with this prefix store the information used to describe the users.
- The prefix “project” stands for the groups in the community. The tables with this prefix store the information related to the groups.
- The prefix “trove” stands for the software category used by SourceForge. The tables with this prefix store the information related to this software category. The related pages in the website are the software map section in the homepage of SourceForge.net.

Statistics tables are another type of table. Normally, these tables have a name starting with a “stats” prefix. SourceForge started to collect statistics for the community in 2003. There are several different statistics collected by SourceForge.

- Tables with names starting with “stats_site” describe the site-wide statistics. These tables are used to support the rating system of SourceForge.net.
- Tables with names starting with “stats_project” describe the project statistics, including message posted, bugs opened and closed, support opened and closed, patches opened and closed, artifact opened and closed, tasks opened and closed and help requested of a given project.
- There are also specific statistical stables for http and ftp downloads for the SourceForge community.

There are also some temporary tables in the database, which are not important to the whole system. Please refer to the schema map attached for further study.

C.1.4 Comments

Since our repository is still evolving, there are some things that are still not precise. I have tried to clarify some common questions about the repository.

1. **The coverage of the repository:** Currently only the database dumps for the following month are available and the corresponding schema names are also given.

- January 2003 (sf0103)
- November 2004 (sf1104)
- December 2004 (sf1204)
- February 2005 (sf0205)
- March 2005 (sf0305)
- April 2005 (sf0405)
- May 2005 (sf0505)
- June 2005 (sf0605)
- July 2005 (sf0705)
- August 2005 (sf0805)
- September 2005 (sf0905)

The missing months will not added, and we will try to keep the data dump continuous from now on.

2. **The time expression:** There are table columns (attributes) that represent time values, such as add_data for user table. All these values are given as integer, as the passing seconds to January 1, 1970. Most programming languages have standard library support to transform these integers to human-readable dates.

3. **Current supported query:** Currently, our query web interface only supports single simple queries, which are combined with three sections – "select" clause, "from" clause and "where" clause.

   "Select" clause includes the attribute names. Users can also format the output in the "select" clause. Please refer to the attached documents for complete attribute list.

   - *Example 1: select user_id, user_name from ...*
   - *Example 2: select user_id||' :' ||user_name from ...*

     "from" clause includes the target tables of the query. Since we used schema to store different database dumps, schema name must be included before a table name is specified.

     "where" clause includes the condition expression for the query. Here are two examples for possible complicated queries.
   - *Example 3: select b.artifact_id, b.summary*
     from sf1104.artifact a, sf1204.artifact b
     where a.artifact_id = b.artifact_id AND a.status_id != b.status_id

   - *Example 4: select distinct project_id from sf0505.user_group*
     where user_id in (select distinct user_id from sf0505.user_group
     where project_id = 100) and project_id != 100

4. **Expected query time:** Expected query time for simple queries will take seconds or minutes to finish. But some tables are huge in record size. For example, artifact table includes 863,068 records in July 2005 dump; forum table includes 1,723,458 records in July 2005 dump; users table includes 1,105,685 records in July 2005 dump; stats_project table includes 1,064,812 records in July 2005 dump. Joining some of these tables could easily create a view with billions of records. In that case, query time can take hours or even days.

5. **Possible usage of the repository:** The data in the repository is sufficient for researchers to study not only the snapshot of the SourceForge community, but also the evolution of the SourceForge community.

C.2 Statistics about the Repository

C.2.1 April 2006 Report

Total queries submitted: 17861

Total query data files retrieved: 11647

Total bytes of query data: 29279517609

User bdhutche retrieved 2 query data files of total size 105662

User xiangl retrieved 294 query data files of total size 4455388394

User mgencer retrieved 22 query data files of total size 53978955

User roberta retrieved 4 query data files of total size 924161262

User srs retrieved 14 query data files of total size 2249650

User jwang2 retrieved 27 query data files of total size 655854988

User - retrieved 1884 query data files of total size 142540910

User jphahn retrieved 222 query data files of total size 17374078680

User claragar retrieved 8 query data files of total size 37382009

User jhoh retrieved 18 query data files of total size 176265228

User beechung retrieved 47 query data files of total size 1588946921

User pierce retrieved 130 query data files of total size 60905316

User zhoul retrieved 2 query data files of total size 6456

User sdaniel retrieved 8782 query data files of total size 80796425

User cconley retrieved 104 query data files of total size 298126886

User gzm108 retrieved 1 query data files of total size 147185

User aradroy retrieved 11 query data files of total size 37190828

User yoris.au retrieved 74 query data files of total size 3391391838

User psingh retrieved 1 query data files of total size 16

C.2.2 May 2006 Report

Total queries submitted: 6073

Total query data files retrieved: 5810

Total bytes of query data: 7015257746

User zhoul retrieved 1 query data files of total size 2423

User sdaniel retrieved 610 query data files of total size 737476

User crowston retrieved 28 query data files of total size 11833077

User xiangl retrieved 534 query data files of total size 1652175935

User jwang2 retrieved 2227 query data files of total size 325282296

User - retrieved 776 query data files of total size 142000206

User wrayb retrieved 1 query data files of total size 2423

User gangpeng retrieved 55 query data files of total size 3058225587

User claragar retrieved 733 query data files of total size 1201135135

User yoris.au retrieved 74 query data files of total size 496684130

User pierce retrieved 771 query data files of total size 127179058

C.2.3 June 2006 Report

Total queries submitted: 16947

Total query data files retrieved: 13343

Total bytes of query data: 26684556278

User srikanth retrieved 1 query data files of total size 134236978

User crowston retrieved 25 query data files of total size 72026072

User xiangl retrieved 187 query data files of total size 40117814

User tripathi retrieved 14 query data files of total size 20584

User jwang2 retrieved 419 query data files of total size 742040886

User pclay retrieved 1 query data files of total size 2423

User - retrieved 531 query data files of total size 1265097014

User u retrieved 1 query data files of total size 200

User jphahn retrieved 292 query data files of total size 13728867679

User jhoh retrieved 4 query data files of total size 157300985

User pierce retrieved 704 query data files of total size 53521142

User u_stettner retrieved 86 query data files of total size 190908368

User ddzega retrieved 114 query data files of total size 8985433218

User jmoon retrieved 11 query data files of total size 17094636

User sdaniel retrieved 10787 query data files of total size 606513015

User newusertest retrieved 1 query data files of total size 27

User cconley retrieved 94 query data files of total size 410039214

User srikanth.mudigonda retrieved 1 query data files of total size 2423

User wrayb retrieved 1 query data files of total size 2423

User mvanantw retrieved 44 query data files of total size 238564242

User aradroy retrieved 2 query data files of total size 28012957

User yoris.au retrieved 1 query data files of total size 13891532

User cliche retrieved 18 query data files of total size 812797

User david.hinds retrieved 4 query data files of total size 49649

BIBLIOGRAPHY

[1] OSS research portal (http://zerlot.cse.nd.edu). 2006.

[2] L.A. Adamic and B.A. Huberman. Scaling behavior of the world wide web. *Science*, 287(2115), 2000.

[3] Charu C. Aggarwal and Philip S. Yu. Outlier detection for high dimensional data. *SIGMOD Proceedings*, 2001.

[4] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. *Proceedings of ACM SIGMOD Int. Conf. Management of Data*, pages 207–216, 1993.

[5] J. Ahola. Mining sequential patterns. *VTT research report TTE1-2001-10*, 2001.

[6] W. Aiello, F. Chung and L. Lu. Random evolution in massive graphs. *IEEE Symposium of Foundations of Computer Science*, 2001.

[7] R. Albert and A.L. Barabási. Emergence of scaling in random networks. *Science*, 286:509–512, 1999.

[8] R. Albert and A.L. Barabási. Dynamics of complex systems: Scaling laws for the period of boolean networks. *Physics Review*, 85(5234), 2000.

[9] R. Albert and A.L. Barabási. Statistical mechanics of compex networks. *Reviews of Modern Physics*, 74(47), 2002.

[10] R. Albert, H. Jeong and A.L. Barabási. Diameter of world-wide web. *Nature*, 401(130), 1999.

[11] R. Alessandra and M. Alessandra. Decoding the "free/open source (f/oss) puzzle" – a survey of theoretical and empirical contributions. *EconPapers, EFI, Italy*, 2004.

[12] L.A.N. Amaral, A. Scala, M. Barthelemy and S. HE. Classes of small-world networks. *Proceedings of the National Academy of Sciences*, 97(21), 2000.

[13] R. Axelrod. The evolution of strategies in the iterated prisoner dilemma. *Genetic algorithm and simulated annealing*, pages 32–41, 1987.

[14] R. Axelrod. Advancing the art of simulation in the social sciences. *Simulating Social Phenomena, ed. Rosaria Conte, Rainer Hegselm ann, and Pietro Terna.*, pages 21–40, 1997.

[15] R. Axelrod. *The complexity of Cooperation: Agent based models of competition and collaboration.* Princeton University Press, Princeton, NJ, 1997.

[16] Z. Bar-Joseph and G. Gerber. Computational discovery of gene modules and regulatory networks. *Nature Biotechnology*, 10(1038), 2003.

[17] A.L. Barabási, H. Jeong, Z. Neda, E. Ravasz, A. Schubert and T. Vicsek. Evolution of the social network of scientific collaborations. *PhysicA*, 311(590), 2002.

[18] J.M. Barrie and D.E. Presti. The world wide web as an instructional tool. *Science*, 274(371), 1996.

[19] R. Bellman. Adaptive control processes: A guided tour. *Princeton University Press, Princeton, NJ*, 1961.

[20] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9), 1975.

[21] G. Bianconi and A.L. Barabási. Competition and multiscaling in evolving networks. *Europhys. Lett.*, 54(436), 2001.

[22] B. Bollobás. Random graphs. *London:Academic*, 1985.

[23] B. Bollobás, O. Riordan, J. Spencer and G. Tusnöady. The degree sequence of a scale-free random process. *Random structures and algorithm*, 18(3):279–290, 2001.

[24] J. Bozman. "linux server growth accelerates, topping 60% year-over-year". http : //www.idc.com/getdoc/jsp?containerId = pr2005_12_26_193650, 2005.

[25] F. Brooks. No silver bullet: essence and accidents of software engineering. *IEEE Computer Magazine*, pages 10–19, 1987.

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

[27] J. Camacho, R. Guimerà and L.A.N. Amaral. Preprint cond-mat/0102127. 2001.

[28] E. Castro and M. Houle. Fast randomized alogorithms for robust estimation of location. *Proc. International workshop on Temporal, Spatial and Spatio-temporal Data Mining, Lyon, France*, 2000.

[29] I. M. Chakravarti, R. G. Laha and J. Roy. Handbook of methods of applied statistics. *John Wiley and Sons*, I, 1967.

[30] S. Chawla, B. Arunasalam and J. Davis. Mining open source software (oss) data using association rules network. *Proceeding of PAKDD conf.*, 2003.

[31] S. Christley. Analysis of activity in the open source software development community. *HICSS 40*, 2007.

[32] S. Christley, Y. Gao, J. Xu and G. Madey. Public goods theory of the open source software development community. *Agent, Chicago*, 2004.

[33] D. L. Cogburn. Hci in the so-called developing world: What's in it for everyone. *Interactions*, 10(2):80–87, 2003.

[34] L. Darden. Recent work in computational scientific discovery. *Proceedings of the Ninteenth Conference of the Cognitive Science Society*, pages 161–166, 1997.

[35] Online document. Pearson's r. $http://www.analytics.washington.edu/rossini/courses/intro-nonpart/text/Pearson\_s\_l\_r\_l\_.html$.

[36] G. Drummond. Open source software and documents: A literature and online resource review. $http://www.omar.org/opensource/litreviewa$, 1999.

[37] M. Elliott. The virtual organizational culture of a free software development community. *Proceeding of 3rd workshop on Open Source Software Engineering*, 2003.

[38] P. Erdös and A. Rényi. On random graphs. *Publicationes Mathematicae*, 6:290–297, 1959.

[39] M. Ester, H. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. *Proc. of 2nd international conference on knowledge discovery and data mining*, 1996.

[40] M. Faloutsos, P. Faloutsos and C. Faloutsos. On power-law relationships of the internet topology. *Computer Communication Review*, 29(251), 1999.

[41] A. Feelders, A. Loux, and J. Zand. A perspective on database and data mining. *Proc. of KDD*, pages 150–155, 1995.

[42] J. Feller and B. Fitzgerald. A framework analysis of the open source software development paradigm. 2000.

[43] R. A. Fisher. On the interpretation of χ^2 from contingency tables, and the calculation of p. *Journal of the Royal Statistical Society*, 85(1):87–94, 1922.

[44] W. Frawley, $G. Piatetsky-Shapiro$ and C. Matheus. Knowledge discovery in databases: an overview. *AI Magazine*, 1992.

[45] Y. Gao. Topology and evolution of the open source software community. *Master Thesis*, 2003.

[46] Y. Gao and G. Madey. Project development analysis of the oss community using ST mining. *NAACSOS, Notre Dame*, 2005.

[47] Y. Gao, G. Madey and V. Freeh. Modeling and simulation of the open source software community. *ADS'05, San Diego*, 2005.

[48] Y. Gao, V. Freeh and G. Madey. Analysis and modeling of the open source software community. *NAACSOS, Pittsburgh*, 2003.

[49] Y. Gao, V. Freeh and G. Madey. Conceptual framework for agent-based modeling and simulation. *NAACSOS, Pittsburgh*, 2003.

[50] Y. Gao, V. Freeh and G. Madey. Topology and evolution of the open source software community. *SwarmFest, Notre Dame*, 2003.

[51] Y. Gao, Y. Huang and G. Madey. Data mining project history in open source software communities. *NAACSOS, Pittsburgh*, 2004.

[52] L. Gasser and W. Scacchi. Understanding continuous design in f/oos projects. *16th International Conference Software and Systems Engineering and Their Applications*, 2003.

[53] Swarm Development group. A tutorial introduction to swarm. http : //www.swarm.org.

[54] J. Hamilton. Time series analysis. *Princeton University Press, Princeton, NJ*, 1994.

[55] C. M. Hannum. The future of netbase. *http://mail-index.netbsd.org/netbsd-users/2006/08/30/0016.html*, 2006.

[56] D. Hiebeler. The swarm simulation system and individual-based modeling. *Advanced technology for natural resource management*, 1994.

[57] W. Howe. When did internet start? A brief capsule history. *http://www.delphi.com/navnet/faq/history*, May 1996.

[58] Repast Information. http://repast.sourceforge.net/. 2002.

[59] A. Jain and R. Dubes. Algorithms for clustering data. *Princeton Hall*, 1988.

[60] A. Jain, M. Murty, and P. Flynn. Data clustering: a review. *ACM Computing Surveys*, 31(2), 1999.

[61] P. Jens. Network formation via contests: the production process of open source software. *Working paper, Johann-Wolfgan-Goethe University, Frankfurt, Germany*, 2004.

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

[63] D. Jensen and J. Neville. Data mining in social networks. *Symposium on Dynamic Social Network Modeling and Analysis, Washington, DC, USA*, 2002.

[64] S. T. Jensen, X. S. Liu, Q. Zhou and J. S. Liu. Computational discovery of gene regulatory binding motifs: A bayesian perspective. *Statistical Science*, 19(1):188–204, 2004.

[65] G. Chin Jr. and C. Lansing. The biological sciences collaboratory. *Mathematics and Engineering Techniques in Medicine and Biological Sciences*, 2004.

[66] S. Kauffman. *Origins of Order: Self-Organization and Selection in Evolution.* Oxford University Press, Oxford, 1993.

[67] D. Kempe, J. Kleinberg and E. Tardos. Maximizing the spread of influence through a social network. *SIGKDD '03, Washington, DC, USA*, 2003.

[68] C. Kevin, A. Hala and H. James. Defining open source software project success. *24th International conference on information systems, Seattle, USA*, 2003.

[69] P.J. Kiviat. Simulation, technology, and the decision process. *ACM Transactions on Modeling and Computer Simulation*, 1(2):89, 1991.

[70] D. Lee and H. Seung. Algorithms for non-negative matrix factorization. *Bell Lab. technical report*, 2000.

[71] D. D. Lee and H. S. Seung. Algorithms for non-negative matrix factorization. *Advances in Neural Information Processing Systems*, 2001.

[72] F. Liljeros, C. Edling, A. Lan, S. He and Y. Aberg. Hub caps could squash STDs. *Nature*, 411(907), 2001.

[73] S. Lohr. Group of university researchers to make web science a field of study. *New York Times*, 2006.

[74] G. Madey, V. Freeh and R. Tynan. Agent-based modeling of open source using swarm. *Eight Americas Conference of Information Systems*, 2002.

[75] G. Madey, V. Freeh and R. Tynan. The open source software development phenomenon: an analysis based on social network theory. *Eight Americas Conference of Information Systems*, 2002.

[76] G. Madey, V. Freeh, R. Tynan and Y. Gao. Agent-based modeling and simulation of collaborative social networks. *AMCIS, Tampa, USA*, 2003.

[77] J. G. March. Exploration and exploitation in organizational learning. *Organizational science*, pages 71–87, 1991.

[78] A. Border, R. Kumar, F. Maghoul, P. Raghavan, S. Rajalopagan, R. Stata, A. Tomkins and J. Wiener. Graph structure in the web: experiments and models. *Computer Networks*, 33(309), 2000.

[79] H. Miller and J. Han. Spatial clustering methods in data mining: a survey. *Geographic data mining and knowledge discovery, Taylor and Francis*, 2001.

[80] J.M. Montoya and R.V. Solé. Preprint cond-mat/0011195. 2000.

[81] M. Moreno and M. Faldani. Open source as a complex adaptive system. *Emergence*, 5(3), 2003.

[82] M.E.J. Newman. Clustering and preferential attachment in growing networks. *Physics Review*, 64(025102), 2001.

[83] M.E.J. Newman. Scientific collaboration networks: I. network construction and fundamental results. *Physics Review*, 64(016131), 2001.

[84] M.E.J. Newman. Scientific collaboration networks: Ii. shortest paths, weighted networks, and centrality. *Physics Review*, 64(016132), 2001.

[85] M.E.J. Newman. The structure of scientific collaboration networks. *Proceedings of National Academic Science*, 98:404–409, 2001.

[86] M.E.J. Newman and D.J. Watts. Random graph models of social networks. *Physics Review*, 64(026118), 2001.

[87] N.S. Padian. STD risks and latino adolescents' sexual network. http : //aidscience.org/Preventionproject.asp?ID = 661, 2002.

[88] S.L. Pimm. *The Balance of Nature.* University of Chicago Press, Chicago, 1991.

[89] W. Poundstone. *The recursive universe.* Contemporary Books, Chicago, IL, 1985.

[90] M. Prietula, K. Carley and L. Gasser. *Simulating organizations: computational models of institutions and groups.* MIT Press, Cambridge, MA, 1998.

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

[92] R. Riolo. The effects of tag-mediated selection of partners in evolving populations playing the iterated prisoner's dilemma. *Santa Fe Institute working paper*, (97-02-016), 1997.

[93] W. Scacchi. Free/open source software development practices in the computer game community. *IEEE Software, Special Issue on Open Source Software*, 21:59–67, 2004.

[94] T. Schelling. On the ecology of micromotives. *The corporate society*, pages 19–64, 1974.

[95] J. Schendure and G. M. Church. Computational discovery of sense-antisense transcription in the human and mouse genomes. *Genome Biology*, 3(9), 2002.

[96] D. H. Sonnenwald. The conceptual organization: an emergent collaborative R&D organizational form. *Science Public Policy*, 30(4):261–272, 2003.

[97] L. Steve and C.L. Giles. Searching the world wide web. *Science*, 280(5360):98–100, 1998.

[98] S. Thorsten. Open source software within organization – critical factors for consulting. *Working paper*, 2004.

[99] H. Wang, W. Wang, J. Yang, and P. Yu. Clustering by pattern similarity in large data sets. *Proc. of ACM SIGMOD, Madison, WI*, 2002.

[100] D.J. Watts. *Small world.* Princenton university press, NJ, 1999.

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

[102] H.S. Wilf and D. Zeilberger. Rational functions certify combinatorial identities. 3:147–158, 1990.

[103] R.J. Williams, N.D. Martinez, E.L. Berlow, J.A. Dunne and A.L. Barabási. Two degrees of separation in complex food webs. *Santa Fe Institute Working Paper Series*, (01-07-036), 2001.

[104] J. Xu, Y. Gao, J. Goett, and G. Madey. A multi-model docking experiment of dynamic social network simulations. *Agent conference, Chicago, IL*, 2003.