OPEN SOURCE SOFTWARE DEVELOPMENT AND MAINTENANCE: AN EXPLORATORY ANALYSIS

A Dissertation

by

UZMA RAJA

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

August 2006

Major Subject: Information and Operations Management

© 2006

UZMA RAJA

ALL RIGHTS RESERVED

OPEN SOURCE SOFTWARE DEVELOPMENT AND MAINTENANCE:

AN EXPLORATORY ANALYSIS

A Dissertation

by

UZMA RAJA

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Approved by:

Chair of Committee,	Marietta J. Tretter
Committee Members,	Arun Sen
	Dean W. Wichern
	Christopher J. Wolfe
Head of Department,	Dean W. Wichern

August 2006

Major Subject: Information and Operations Management

ABSTRACT

Open Source Software Development and Maintenance: An

Exploratory Analysis. (August 2006)

Uzma Raja, B.Sc., University of Engineering and Technology, Lahore, Pakistan;

M.S., Texas A&M University

Chair of Advisory Committee: Dr. Marietta J. Tretter

The purpose of this research was to create measures and models for the evaluation of Open Source Software (OSS) projects. An exploratory analysis of the development and maintenance processes in OSS was conducted for this purpose. Data mining and text mining techniques were used to discover knowledge from transactional datasets maintained on OSS projects. Large and comprehensive datasets were used to formulate, test and validate the models.

A new multidimensional measure of OSS project performance, called project viability was defined and validated. A theoretical and empirical measurement framework was used to evaluate the new measure. OSS project data from SourceForge.net was used to validate the new measure. Results indicated that project viability is a measure of the performance of OSS projects.

Three models were then created for each dimension of project viability. Multiple data mining techniques were used to create the models. Variables identified from process,

product, resource and end-user characteristics of the project were used. The use of new variables created through text mining improved the performance of the models.

The first model was created for OSS projects in the development phase. The results indicated that end-user involvement could play a significant role in the development of OSS projects. It was also discovered that certain types of projects are more suitable for development in OSS communities. The second model was developed for OSS projects in their maintenance phase. A two-stage model for maintenance performance was selected. The results indicated that high project usage and usefulness could improve the maintenance performance of OSS projects. The third model was developed to investigate the affects of maintenance activities on the project internal structure. Maintenance data for Linux project was used to develop a new taxonomy for OSS maintenance patches. These results were then used to study the affects of various types of patches on the internal structure of the software. It was found that performing proactive maintenance on the software moderates its internal structure.

DEDICATION

To Scheherbano and Jahanzeb

ACKNOWLEDGEMENTS

All praise and thanks to Allah, the most beneficent the most merciful

I am grateful to my committee chair Dr Marietta Tretter. Her guidance, support and encouragement made this dissertation possible. I would also like to thank my committee members Dr Dean Wichern, Dr Arun Sen and Dr Christopher Wolfe for their guidance and support.

This research was funded in part by a SAS software grant as part of the SAS Fellowship Program. I would like to thank SAS for providing me with both SAS Enterprise Miner and SAS Text Miner to complete my research. I am also thankful to Dr Evelyn Barry, Dr Ravi Sen and Dr Krishnna Narayanan for their advice. Thanks to all my friends and colleagues and the department faculty and staff at the INFO department for a great experience.

I am grateful to my parents, Raja Shafaat Ullah and Kalsoom Shafaat, for their love, support and encouragement. I am thankful to my siblings; Usman, Ayisha and Irfan Raja for being my support network. My two children: Scheherbano and Jahanzeb, are a source of joy and inspiration for me. This work would have been impossible without their love and patience. Finally, I have no words to express thanks to my husband Rafay Ishfaq. I would be lost, without his love and friendship.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

LIST OF TABLES

TABLE Page

TABLE Page

LIST OF FIGURES

FIGURE Page

CHAPTER I

INTRODUCTION

The purpose of this chapter is to provide a background on Open Source Software (OSS). It also discusses the significance of studying software development and maintenance in OSS. It ends with presenting the scope of this dissertation.

1.1 OSS BACKGROUND

In 1970, less than one percent of the public could describe what computer software meant (Pressman 2004). Unlike other human inventions, software is a logical element rather than a physical entity. Therefore, in the early days of computers, there was no concept of charging money for selling software. It was an entity, packaged with the computer hardware and all the source code was freely available. As the use of computers became more pervasive, software became commercialized (Feller and Fitzgerald 2002). In order to preserve the commercial value of software, companies started to deliver software as a black box, where the user could only access the output features, but the internal structure of the software or the source code was hidden from the user. Protecting the source code is a tool commercial software development companies use to keep control over the life cycle

---

This thesis follows the style of *Management Information Systems Quarterly.*

operations of a software system. Thus, the end user of commercial software has no visibility or control over the source code. If a fault occurs, it has to be reported back to the manufacturer or the maintenance service provider and only authorized parties can locate and remove faults. This also holds for upgrading the software to adapt to new requirements. For example, during the Y2K crisis, companies had to spend millions of dollars to have their systems upgraded. This type of software development is referred to as Closed Source Software (CSS).

As a reaction to the mass commercialization of software, some programmers felt the need for upholding the concept of freedom of sharing software. In 1981, Richard Stallman founded the Free Software Foundation (http://www.fsf.org/fsf/fsf.html). People of similar interests and ideology started to develop software, which was available to users free of cost and with full access to the source code (MacCormack 2002). The software structure was no longer a black box, but was an open artifact, available to be used, upgraded, maintained or changed. The GNU Project was launched in 1985 to develop a complete Unix-like operating system, which is free software: the GNU[1] system. Variants of the GNU operating system, which use the kernel Linux, are now widely used; though these systems are often referred to as “Linux”, these are more accurately called GNU/Linux systems (Wu and Ling 2001). Linux was started in 1991 as a personal project of a Finnish graduate student, Linus Torvaldas. He posted his code on the Internet and invited people to use it and find bugs. Over the period of time, Linux has grown to become one of the most popular operating systems being used (Godfrey and Tu 2001; Kerstetter et al. 2003). In 1976, Bill Gates in

[1]GNU is a recursive acronym for ``GNU's Not Unix"; pronounced as "guh-NEW".

"open letter to hobbyists" predicted that free sharing of software would prevent the writing of good code. However, Linux prevailed over such predictions and has become one of the most successful open source projects. Linux, with its interesting development phenomenon is currently competing well with its commercially developed counterparts. Estimates reveal 8000 person years of effort went into the development of Linux Red Hat 7.1 (Wheeler 2003). It would cost over 1.08 billion dollars, had it been developed commercially.

Eric Raymond in 1990 wrote an article titled "Cathedral and the Bazar" (Feller and Fitzgerald 2002; Raymond 2001). In the article, he argued that the OSS development method was a credible competitor to CSS projects. He identified the characteristics of OSS development that enabled it to maintain a superior quality despite being a non-commercial endeavor. That paper gained much fame and attention and the OSS phenomenon became a topic of discussion in the research community.

The OSS paradigm has led to the development of some very successful software by a community of contributors who share the source code for free (Kerstetter et al. 2003). In recent years, there has been a focus on OSS projects and several streams of research have emerged in these areas. However much of the work has been either focused on case studies of large-scale development projects (Aoki et al. 2001; German 2004; Kerstetter et al. 2003; Markus et al. 2000; Scacchi 2004b; Wheeler 2003). Some have looked into issues like participation in OSS projects (Bergquist and Ljungberg 2001; Crowston and Scozzi 2002; Dempsey et al. 2002; Hertel and Herrmann 2003; Hippel and Krogh 2003; Huntley 2003), effects of licensee choices (Lerner and Tirole 2001; Lerner and Tirole 2002; Stewart and

Ammeter 2006; Stewart and Gosian 2006), and organization structure of OSS communities (Jensen and Scacchi 2005; Koch and Schneider 2002; Krishnamurthy 2002; Krogh et al. 2003). There is a lack of body of work that analyses the immense datasets available and explores them for new information within this domain.

The empirical research in OSS has been focused on evolution of OSS projects and testing the laws of software evolution in the OSS domain (Capiluppi et al. 2003; Capiluppi et al. 2004; Capliuppi 2003; Godfrey and Tu 2001; Scacchi 2002; Scacchi 2004b). Some studies have been focused on analysis of fault detection in OSS projects and comparing them to CSS projects (Koru and Tian 2004; Paulson et al. 2004). Empirical work in OSS has also focused on success and failure of OSS projects (Crowston et al. 2003; Crowston et al. 2004; Jensen et al. 2004; Krishnamurthy 2002). However, these studies used a small sample of projects and an abstract definition of success.

1.2 SIGNIFICANCE OF OSS RESEARCH

Open source software is typically developed by online volunteer communities of programmers and is available to the public for download, use, modification and upgrades (Feller and Fitzgerald 2002). Most OSS products are free or have a nominal charge associated with them. While CSS projects are struggling to meet costs, user requirements and schedule, OSS projects are becoming more and more popular. According to recent reports, 80% of the web servers use Apache, an OSS web server. Many small and medium scale OSS projects are also finding their way into corporate use. Organizations claim to have

saved millions by switching to OSS solutions. Amazon's switch to Linux has reportedly saved them over 7 million dollars. IBM has launched over 30 of its projects in the OSS community. According to a recent Gartner survey, by year 2010, 80% of the businesses would have considered using OSS projects and 25% would be using OSS projects in their business transactions (Cearly et al. 2005). This interest in OSS projects is not just because of their free availability, but also because of their high quality and ability to fulfill user requirements (Lerner and Tirole 2002).

The increase in the use of OSS projects demands a deeper understanding of the OSS development and maintenance process. Procurement of the software is not the only cost associated with the use of a software system. Software systems usually have a high operational cost (Banker et al. 1998; Kemerer 1995). Thus, before an organization makes a decision to use new software, there is the need for considerable evaluation of the product. The maintenance costs of a software system can be very high, if frequent changes are made to it. If the adoption of the new system fails, there could be additional financial losses in terms of loss of data, loss of time, and loss of technical expertise.

The OSS teams also need to be able to have more control over the evolution of their projects. Some companies like, IBM, have launched their projects in the OSS domain and have assigned staff to OSS development. Therefore, for OSS developers, there is a need for research to determine what factors would affect the long-term lifecycle outcomes of their projects.

1.3 SIGNIFICANCE OF SOFTWARE DEVELOPMENT AND MAINTENANCE RESEARCH

Software development is the process that is carried out before it is launched as an operational system. In traditional software development, the activities during this phase include requirements and system specification, initial design, software coding and testing. There are various methodologies available for carrying out software development e.g. Waterfall, Spiral, Agile and Rapid Application Development etc. These methodologies specify issues like team formation, task assignment, milestone definition and cost and scheduling of the project (Pressman 2004).

The failure rate of software development projects is very high. The 90% syndrome in software development projects implies that the majority of software development projects fail to meet the expected time and cost schedules (Abdel-Hamid 1988; Brooks 1995). This failure affects the over all system costs and performance. Much of software engineering research has been focused on identifying the causes of software project failure. The community has been long in search of a silver bullet that would help overcome this great challenge (Brooks 1995) .

Once software code is complete and has been tested, it is ready to be operational and to be used by the customer. However, the nature of software is such that it undergoes change throughout its life. Any change made to the software product after its development has been completed, is called software maintenance (Pressman 2004; Zelkowitz et al. 1979). Software

maintenance may include activities like fault correction, improvement of performance or adaptation to changes in the operational environment (Pressman 2004).

Software maintenance claims a large proportion of the lifecycle costs of a software system and is a large component of the Information Systems (IS) budgets of organizations. According to estimates, software maintenance consumes more than 80% of the lifecycle costs of software systems (Erlikh 2000; Moad 1990). From the point of view of organizational resources, IS departments spend 50-80% of their budgets on software maintenance (Deklava 1992; Huff 1990). Therefore, there is a considerable interest in the improvement and control of the process of software maintenance. Many of the problems of software maintenance are a result of software development and design inadequacies (Schneidewind 1999; Swanson and Beath 1997). Therefore while studying the operational maintenance of software; it is critical to have a deeper understanding of software development (Banker and Slaughter 1995).

1.4 RESEARCH SCOPE

The purpose of this research is to create models that explain the development and maintenance of OSS projects. The objective is to identify the key factors that affect the outcomes of OSS projects. This will help the OSS teams to better evaluate and control their projects. For the user and business community, it will provide quantitative evaluation of OSS projects, so that decisions regarding use of OSS projects could be made. It will utilize the immense volume of data artifacts available for OSS projects.

First, this study develops a new multidimensional measure of OSS project performance. The measure is based on research from ecology, information theory and engineering. It provides a tangible quantification of OSS projects that can be used to compare OSS projects. The measure is developed according to the measurement theory in software engineering and is tested for mathematical and empirical validation.

Second, exploratory analysis of the transactional data of OSS development and maintenance is carried out to create three models of OSS performance evaluation. The availability of rich datasets enables the use of data mining techniques for model formulation. Prior work on software engineering has relied heavily on military data or experimental studies involving student subjects (Sharpe et al. 1991). There is no evidence that any of these studies could be generalized for the OSS domain.

By using a dataset from a very large population of medium and small-scale OSS projects, this study provides unique and important insights into the key issues involved in OSS development. As organizations start to use OSS projects, these measures and models will provide an effective tool for comparison of various OSS projects. It will also benefit the developer community by enabling them to monitor and control the performance of their project. In future, the results of this research may be generalized to CSS project management.

1.5 ORGANIZATION OF THIS DISSERTATION

This dissertation is organized into seven chapters. Chapter I discusses the significance of developing measures and models for OSS projects. Chapter II discusses a new multidimensional measure of project viability. Each dimension is defined and validated against a measurement framework. Chapter III discusses the steps involved in the exploratory research methodology used in this research. The datasets and the variables used for model development are also discussed in this chapter. Using these datasets and variables, three models for each dimension of viability are then developed. Chapter IV, V and VI present the results of the models for the dimensions vigor, resilience and organization respectively. The summary and conclusion of the research is discussed in chapter VII. Figure 1.1 shows the layout of the dissertation.

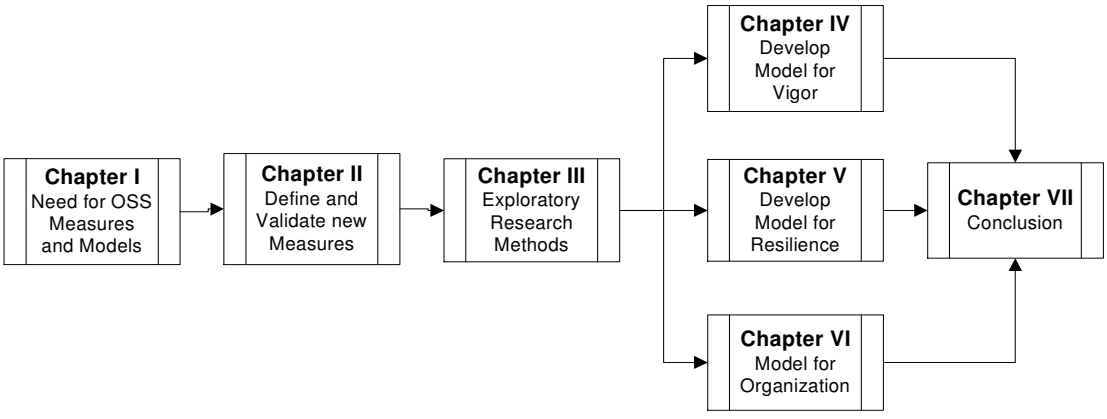

Figure 1.1: The organization of this dissertation

CHAPTER II

PROJECT VIABILITY: A MULTIDIMENSIONAL MEASURE

This chapter presents the background on need for a new OSS project performance measure. The new measure is then defined in natural language and mathematically. The theoretical and empirical framework for validating the measure is then discussed. Finally, the measure is validated according to the defined framework.

2.1 BACKGROUND

Traditionally performance of software projects has been evaluated based on conformance to budget, schedule and user requirements (Pressman 2004). All such measures hold little or no meaning for OSS projects, which are developed online by volunteer software programmers, without any defined user requirements or budget. Yet, there is a need for tangible measures that could be used to evaluate the performance of these projects. Development teams need such measures to control and improve the performance of their projects. The end users and businesses need some kind of measures to compare the OSS projects before making decisions regarding project adoption. Therefore, a new multidimensional measure of OSS project lifecycle performance was defined and validated in this research. This measure was used to develop performance models for use with OSS projects.

Software measurement is a very important component of developing an understanding of software engineering practices and processes (Fenton and Pfleeger 1991). According to measurement theory, measurement is the process through which numbers are assigned to attributes of entities in real world, so as to describe them according to clearly defined rules (Melton et al. 1990). Software metrics are quantitative measures that enable software engineers to make subjective evaluations about a project (Fenton and Pfleeger 1991). During project development, software measures can be used to spot trends and to make improvements accordingly. These measures can also be used to compare various software products and to make informed decisions regarding use of new software (Pressman 2004).

The need for new measures arises, when a domain is new and not many reliable and tangible measures exist (Briand et al. 1996; Kitchenham et al. 1995). As mentioned earlier, the existing measures of project performance are not suitable for OSS projects. Therefore, a new multidimensional measure of OSS project performance called project viability was defined and validated in this research. Project viability is a measure of survivability of a project. Ecologists describe natural systems' survivability in terms of their ability to grow, maintain structure and respond to perturbations (Costanza and Mageau 1999). OSS projects can be modeled as natural systems that grow, develop, get sick (have faults) and recover through their life cycle and die (become inactive), to be replaced by new projects. OSS projects develop like natural ecosystems; they evolve without requirements in the presence of threats[2] to their existence. The ability of a project to overcome these threats and continue

[2]The threats can be internal programming errors, or loss of team members, or other environmental factors.

to grow, determines whether a project would survive or become inactive. The quality and life expectancy of OSS projects depends on some critical characteristics. These characteristics can be used to evaluate the current performance and predict future performance.

OSS project viability is defined as the ability of a project to grow and maintain its structure in the presence of perturbations. It was developed as a 3-dimensional measure, as shown in figure 2.1, with each dimension having equal weight in the overall viability. The project viability measure can be used to monitor the performance of a single project over its lifecycle or to compare multiple projects with each other.

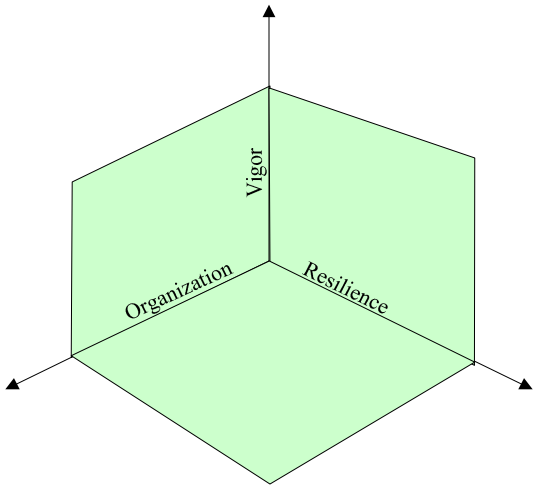

Figure 2.1: Three dimensions of OSS project viability

To create and validate a new multidimensional measure, the first step was to define each dimension of viability. In addition to mathematical definition, it is also important to have a natural language definition of a new measure (Kitchenham et al. 1995; Melton et al. 1990). For wider application of the measure, it is imperative that the measure be independent of technology and the programming environment (Churcher and Shepperd 1995). The following discussion explains each dimension and its measurement in detail for a project *P*, with viability measurements taken at time period t_n; $n = 0, 1, 2 \ldots$ (measured in days, weeks or years).

2.1.1 Vigor

Vigor refers to the ability of a project to evolve over a period of time. It is a measure of its growth or throughput. The vigor of a project will change through its lifecycle. During the development phase, vigor represents the addition of the basic functionality required for successful transition to the next phase of its life cycle. During the maintenance phase, vigor represents incremental changes incorporated to add new functionality to the project. For comparing different projects, the growth can only be compared as a function of time. The age of the projects varies and therefore the amount of functionality added can increase with the age. In order to make the new measure scale invariant, it is normalized over time. The incremental functionality can be measured by files released per unit time. The vigor of a project at time period t_n can be calculated as:

$$V = \frac{1}{t_n} \sum_{i=0}^{n} G_i \qquad \text{..... (2.1)}$$

Where;

V = Vigor at time period n; $0 \leq V \leq G_{norm}$

G_i = Number of new versions released in time period i

G_{norm} = Maximum files released per unit time [3]

The greater the number of files released per unit time, the greater will be the vigor of the project. The normalization with respect to time ensures that older projects do not have a higher value of vigor because of their age. It makes the comparison between projects of different age, valid.

2.1.2 Resilience

The resilience of a project is the ability of a project to respond to internal and external perturbations. These perturbations refer to changes in the operating environment of the project. For software projects, perturbations are typically described as occurrences of errors and bugs, which affect the operation of the system. Resilience refers to the ability of a project to remove the error and become operational again. Ideally, a project should be able to adapt to the changes in its environment quickly. The longer it takes a project to react to a perturbation; lower will be its performance. In corporate environment, such a time delay can translate into very high operational costs. Therefore, resilience is an important component of the project viability, especially for projects in the operational phase of their lifecycle.

[3] Assumption: There is finite number of file releases during the lifecycle of a project.

In OSS projects, the users hold no contractual obligations to the developers regarding commitment to the project. Therefore, if a project fails to adjust to perturbations, the end user might abandon that project. Poor resilience can also discourage programmers from contributing to that project. This can cause a reduction in the development and maintenance effort and eventually affect the project viability.

Resilience is measured in terms of response time. A shorter response time reflects higher degree of resilience. The resilience of project *P* is given by the following equation:

$$R = \frac{t_n - \frac{1}{q}\sum_{j=1}^{q} d_j}{t_n} \qquad \text{..... (2.2)}$$

Where;

R = resilience value at time period n; $0 \leq R \leq 1$

q = Total number of perturbations in time period t_n

d_j = Time taken to react to perturbation j

For a project with high resilience, the value reflects shorter response time to react to a perturbation. If on the other hand, removing an error takes a long time, then the resilience of the project will be low. The value of resilience is normalized for the total number of perturbations taking place in a project. For example, if a project has higher number of bugs reported, then dividing this number by total number of bugs for the project will normalize

resilience value. This is done to ensure that different projects with varying number of perturbations over their life cycle can be compared against their resilience levels.

2.1.3 Organization

Organization of a natural system refers to the number and diversity of interactions between its components. In terms of software projects, organization is the measure of the diversity of the interaction between the project members and the information exchanges between them. A highly organized project is characterized by a high diversity of specialized members and their corresponding specialized interactions. Organization decreases as the diversity of the members and the specialization of the information exchange decreases[4] In an OSS project, if there is a large group of specialist developers and maintainers who respond to specific problems, project organization will be higher compared to a project where there is equal number of general-purpose programmers.

In order to measure organization of a project, it is required that both diversity and magnitude of the interactions within a project be known. Information theory offers system level measurement of interactions as means to measuring system organization. One such measurement is Average Mutual Information (AMI). AMI has been adopted in many disciplines e.g. biology, engineering and ecology as a comprehensive measure of organization (Pierce 1980; Ulanowicz 1986). AMI is a validated measure of the organization

[4] Consider a natural system containing species that feed on only one or two preys and are in turn preys to only one or two other species. This system will have high organization compared to a system with the same number of generalist feeders with multiple pathways of exchange between them.

of a system (Pierce 1980). AMI refers to the amount of information that is available on an element of a system, given the value of another element. A highly organized system has a high AMI.

Consider an OSS project where users interact with N members of a project team. For any message originator *x*;

$$x \in \{ x_i \}, i \in \{ 1, 2, ..., N \} \quad \text{..... (2.3)}$$

Let $P(x_k)$ represent the probability that a message/task is originated by a member *k*, $x = x_k$.

The information content in the symbol therefore is defined as:

$$I(x_k) = \log(\frac{1}{P(x_k)}) = -\log P(x_k) \quad \text{..... (2.4)}$$

The information exchange between two entities has some interesting properties. First, if only one member sends the information, then the value of the information content vector $I(x_k)$ is 0 (Pierce 1980). Second, the information is always positive, and finally, information is additive i.e. the total information in two independent members is the sum of the information for each:

$$I(x_i, x_j) = -\log\left(P(x_i, x_j)\right) = -\log\left(P(x_i)P(x_j)\right)$$

$$= -\log P(x_i) - \log P(x_j) = I(x_i) + I(x_j) \quad \text{..... (2.5)}$$

Before discussing AMI equations, another critical measure is needed. The entropy *H(x)* of an entity *x,* with a probability of occurrence p(x), is the measure of uncertainty about the information of x and is given by the equation:

$$H(x) = -\sum_{x} p(x)\log_2 p(x) \qquad \text{..... (2.6)}$$

The joint entropy of x and y defines the uncertainty of the message being exchanged between x and y, where p(xy) is the joint probably of the occurrence of the pair (x,y)

$$H(x, y) = -\sum_{x,y} p(xy)\log_2 p(xy) \qquad \text{..... (2.7)}$$

For an information (task) exchange between team member *x* and *y*, the conditional entropy of *x* given *y* is:

$$H(x \mid y) = H(x, y) - H(y) \qquad \text{..... (2.8)}$$

Thus the conditional entropy is a measure of the average information in *x* given *y* is known (Pierce 1980) . In other words, if a task is completed by the member *y*, then *H(x| y)* is the remaining uncertainty in knowing who originated the task. The Average Mutual Information (AMI) is defined as the average information gained by *x*, when observing *y*:

$$I(x; y) = H(x) - H(x \mid y) = H(x) + H(y) - H(x, y) = I(y; x) \qquad \text{..... (2.9)}$$

In other words, the AMI is the difference in the uncertainty of *x* and the remaining uncertainty of *x*, after observing *y*. It can also be stated that AMI is the reduction in the

uncertainty of *x*, by knowing *y* (Pierce 1980). In terms of OSS projects, consider the tasks as being originated by some individuals and completed by others. Then using equations 2.3-2.9

$$AMI = \sum_i \sum_j \frac{T_{ij}}{T} \log_2 \frac{T_{ij} T}{T_{i\bullet} T_{\bullet j}} \qquad \text{..... (2.10)}$$

Where T_{ij} = Task originated by *i* and completed by *j*

$T_{i\bullet}$ = All the tasks requested by *i*

$T_{\bullet j}$ = All the tasks completed by *j*

T = Total tasks in the project

2.2 EVALUATION OF OSS PROJECT VIABILITY MEASURES

A formal measure captures the intangible aspects of a relationship to the mathematical world (Kitchenham et al. 1995; Kitchenham et al. 2002). Project viability is a three-dimensional[5] attribute of OSS project performance. Therefore, each of the three measures had to be defined, evaluated and validated to confirm that it would behave logically over the entire OSS population [6]. Initially each measure was evaluated for proper

[5]For example, referring to one OSS project being better than the other, or regarding an OSS project as successful, has no formal mathematical relationship. There is a need to quantify the measures of performance.

[6]It is also very important to distinguish between simple attributes and multidimensional attributes. The simple attributes like size are scalar, while the multidimensional attributes e.g. viability are vectors.

logical and mathematical properties. Later OSS project data was used to evaluate the and predictive validity of these measures. Logical evaluation of the measure, coupled with empirical validity, ensures that the new measure is robust and valid (Kitchenham et al. 1995).

To determine the evaluation criteria for viability, prior studies on determination of evaluation criteria and traditional measurement theory were consulted (Allison 1978; Barry et al. 2000; Chidamber and Kemerer 1994; Weyuker 1988). There has been a significant body of research on validation of software metrics (Schneidewind 1992). For validation of a new measure it is important to have a clear and intuitive natural language definition and a precise mathematical definition, so that the application is repeatable (Finkelstein and Learning 1984). There are two methods of testing the validity of a new measure: theoretical validation and empirical validation. Theoretical validation confirms that the measure does not violate any necessary properties of the elements of measurement. Empirical validation confirms that the measured attribute is consistent with reality[7] (Kitchenham et al. 1995).

2.2.1 Theoretical Measurement Validation

While creating new measures, it is critical to establish theoretical validity. Prior research in evaluation of new software measures provides a framework for theoretical and logical validation of a measure. Weyukner proposed nine axioms for validation of new measures of software complexity (Weyuker 1988). These measures have been widely adopted and used for validation of a range of software. Melton (Melton et al. 1990)

[7]For example, a project identified to have a high viability is actually a project that survives in OSS domain.

proposed a framework of validation of new measures. Kitchenham et al., summarized and selected the most general validation criteria for any software measure. This framework was adopted for theoretical validation of project viability (Kitchenham et al. 1995). The four properties of new measures suggested are:

1. For an attribute to be measurable, it must allow different entities to be distinguished from one another.

2. A valid measure must obey the Representation Condition, i.e. it must preserve our intuitive notions about the attribute and the way in which various entities are distinguished.

3. Each unit of an attribute contributing to a valid measure is equivalent,

4. Different entities can have the same attribute value (within the limits of measurement error).

Each dimension of viability was validated according to this framework.

2.2.1.1 Vigor

The vigor of an OSS project is defined in equation 2.1. Testing it against the above-mentioned four properties of the framework, it is evident that:

1. Projects are of varying sizes and have a varying pattern of file releases. The frequency of file release will vary from one project to another and will vary for the

same project during various phases of its lifecycle. As per Weyuker's explanation, there would be at least two projects, for which the vigor would be different.

2. The measure of files released per year, does obey the intuitive notion of growth and throughput. Files released and transitions of development phases have been used in other research as measures of productivity, evolution, and functionality of a project. Therefore, vigor conforms to this requirement.

3. As per the mathematical definition of vigor, all units (subsequent releases) are treated equal. This is a standard measurement practice.

4. The definition of vigor does allow different projects to have the same vigor. If the number of files released for two projects per given time are the same, then the measure of vigor would be the same for both projects in that time period.

2.2.1.2 Resilience

The resilience of a project is defined in equation 2.2. Testing it against the above-mentioned four properties of the framework, it is evident that:

1. The response time to a perturbation can vary for different projects and within the same project. Therefore, the formula allows different projects to vary in the value of resilience.

2. The response time measured as the time taken to react to a perturbation obeys the intuitive notion of resilience.

3. By averaging and normalizing the time taken to react to each perturbation, each unit is treated equal.

4. The formulation allows different projects to have different resilience.

2.2.1.3 Organization

AMI is a validated measure of system structure. The McCabe's Cyclomatic complexity measure uses the same derivations to measure the internal structural complexity of software (McCabe 1976). Testing it against above-mentioned four properties of the framework, it is evident that:

1. The organization of different projects will depend upon the magnitude and the diversity of the interaction, therefore projects could be distinguished based on the formulas discussed earlier.

2. The measure obeys Representation Condition and follows the intuitive notion of structure.

3. All interactions are aggregated with equal weight, therefore, this condition is satisfied.

4. The formulation does allow different projects to have the same AMI (within error limits).

2.2.2 Empirical Measurement Evaluation

The empirical validation of the measure establishes external validity. Like theoretical evaluation, there was the need for a framework of evaluation for the empirical evaluation. The purpose of empirical evaluation of the measure is to ensure that its mapping to a value captures the understanding of the attribute (Kitchenham et al. 2002). In this case, the measure of viability was used as a measure of the performance of OSS projects. Therefore, empirical validation involved computing the viability of various OSS projects. This was used to corroborate whether use of the measures of viability offered discrimination between projects of known high or low performance. For statistical validation, the appropriate tests and confidence levels had to be identified. It is worth mentioning here that for a new measure, it is possible that it will be valid under certain criteria and not under others. A correlation was considered weak if it was statistically insignificant ($p > 0.05$). Furthermore a correlation was considered weak if $|correlation| < 0.4$ and was considered strong if $|correlation| > 0.4$ (Kitchenham et al. 2002).

There are three types of external validity: Convergent, Discriminant and Predictive (Trochim 1999). Convergent validity could be demonstrated by comparison between the new measure and some other metric that measured the same property. Unfortunately, no validated measure of OSS project performance exists. Therefore, it was not be possible to establish this type of validity for any of the dimension of viability. Discriminant validity required that the three dimensions of viability be independent. This was achieved by demonstrating that the measures are orthogonal. This ensured that the measures were three

separate dimensions of viability (Kitchenham et al. 1995). Analyzing the correlations between each measure of viability will achieved this purpose. Testing the results of viability for various projects and then comparing the performance evaluation of known successful or failed projects demonstrated the predictive validity (Barry and Slaughter 2000; Kitchenham et al. 1995).

To test the application of the measure of viability empirically, a sample of 20 projects[8] was selected from the sourceforge.net dataset. The computation of the measure of organization was very complex and time consuming. The number of projects was kept small to facilitate the computation process. The purpose was to demonstrate the use and the validity of the new measure. A summary of the measures is in Table 2.1.

Table 2.1: Operationalization of OSS project viability dimensions

Dimension	**Measure**	**Details**
Vigor	New versions released per year	New releases are indicative of functionality growth of the project (Boehm 1987)
Resilience	Bugs fixed per unit time	Time taken to remove a bug indicates the response time of the project team to handle changes that occur in its environment (Pressman 2004)
Organization	Average Mutual Information	The structure of the maintenance process reflects the organization of the project (Pierce 1980; Ulanowicz 1986)

[8]Considering the complexity of this analysis, number of projects was kept small. This analysis is carried out to demonstrate the use of new measure and to validate the dimensions. In later chapters when new models are created, a larger dataset is used.

The sample projects used for this analysis were selected randomly. The first step was to compute the vigor of the projects. The number of files released by each group was extracted. This number alone could not be used because the total number is dependent upon the age of the project. In order to make the measure age invariant, the number of new releases per year was calculated. The date of registration for each project was extracted from the Sourceforge.net dataset. This time was available as UNIX epoch time. The UNIX epoch time is the number of seconds elapsed between 1/1/1970 and the actual time of stamp. The UNIX epoch time was converted to the Georgian calendar date through the transformation:

$$Date\_of\_registration = \frac{Unix\_Epoch\_time\_of\_registration}{86400} + 25569$$

The dataset used for this research was extracted from the Sourceforeg.net data warehouse for May 2005. Therefore, the age of the project was computed by subtracting the date of registration from the date of extraction of the dataset. This gave the age of the project. The total number of new versions released by the project was then divided by the age in years to compute the new versions released per year. This variable was used as a measure of the vigor of the project.

The measure that was considered next is resilience. Resilience is the ability to recover from a disturbance. In software project, the occurrence of errors is the most significant form of disturbance that can be measured from the available dataset[9]. The total

[9]It is realized here that there can be other types of major disturbances, e.g. change in the project team, change in the platform the project runs on, etc. However, for this analysis given the nature of the dataset, the most stable measure that can be computed in the time to fix bugs.

bugs that occur in a project were computed from the dataset. The SourceForge dataset contains data on all kinds of project artifacts in a single table. The steps followed were as follows:

```
BEGIN

P = Total number of projects
# Initialize index

p = 1;
# Initialize Sum_time

Sum_time = 0;

WHILE (p <= P)

# for all projects
DO
{
        # Extract the bug repository id for each project using the group_id
        Bug_Rep_id[p] =  Bug repository identification number for project p;

        #Compute the total number of bugs for each p
        N[p] = Total bugs for project p;

For (i = 1 to N[p])
        {
                # Extract the time bug was reported
                Time_open = bug report time;

                # Extract the time bug was closed
                Time_Close = bug closed time;

                # Compute bug fix time
                Fix_time = Time_Close - Time_open ;

                # Compute total fix time
                Sum_time = Sum_time + Fix_time;
        }

# Compute Average Time to fix bug for the project
MTTF[p] = Sum_time / N[p];

# Go to next project
  p = p+1;
}
END
```

The organization of the OSS projects was computed by AMI, which is a validated measure of system structure (Pierce 1980). The following steps were involved in the computation of AMI:

```
BEGIN

P = Total number of projects

# Initialize index
p = 1;

WHILE (p <= P)

# for all projects
DO
{
        # Compute total number of distinct maintainers
        M(Kemerer and Slaughter) = number of maintainers;


        # Compute total number of distinct reporters
        R[p] = number of bug reporters;

        # Compute total number of bugs reported
        T[p] = Compute total number of Bugs reported;

        # For each bug reporter
        For (r = 1 to R)
                Ti[p,r] = Compute number of bugs reported;

        # For each maintainer
        For ( m = 1 to M[p])
                Tj[p,m] = Compute number of bugs fixed;

        # For each reporter and maintainer
        For (r = 1 to R[p])
                For (m = 1 to M[p])
                        Tij[p,r,m] = Compute number of bugs reported by r, fixed by m;
```

```
# Compute entropy of bug reporters  H(x) = -(T_i/T) Σ_i ln(T_i/T)

        For (x = 1 to R[p])
                H[p,x] = Entropy of bug reported for each project;

        # Compute entropy of bug maintainers  H(y) = -(T_j/T) Σ_j ln(T_j/T)

        For (y = 1 to M[p])
                H[p,y] = Entropy of bug maintainer for each project;

        # Compute joint entropy  H(x, y) = -(T_ij/T) Σ_i Σ_j ln(T_ij/T)

        For (x = 1 to R[p])
                For (y = 1 to M[p])
                        Hxy[p,x,y] = Entropy of bugs reported by x, fixed by y;

        # Compute  AMI = H(x) + H(y) - H(x, y)
        For (x = 1 to R[p])
                For (y = 1 to M[p])
                        AMI[p] = H[p,x] + H[p,y] - H[p,x,y];

        # Go to next project
         p = p+1;
}

END
```

Discriminant validity was supported by lack of high correlation between unrelated measures. In order to establish the discriminant validity, the values for vigor, resilience and organization were computed for 20 projects. One project had to be rejected because the bug repository was not set up and the data on maintenance could not be retrieved for resilience computation. The results of the correlation analysis are in Table 2.2. The purpose of the analysis was to ensure that the measure is valid, i.e. it provides discriminatory power between projects of varying performance. The formulas discussed earlier were used to compute the values of vigor, resilience and structure. For multidimensional measures, it is

imperative that the measures be independent or orthogonal to each other. This means that each dimension measures a different attribute for the composite attribute and that there is no correlation between each dimension. In order to ensure this, the pair wise correlation of each dimension was computed. The results of the Pearson Correlation Coefficients for the three measures are shown in Table 2.2.

Table 2.2: Pearson correlation coefficient for the three measures of viability

Variable	**By Variable**	**Correlation**	**Sig prob**	**Result**
Vigor	Resilience	0.1197	0.6254	Reject correlation
Organization	Resilience	0.1574	0.5155	Reject Correlation
Organization	Vigor	0.4563	0.0496*	Weak correlation

The Pearson correlation coefficient values indicate a correlation between organization and vigor. Further investigation indicated that the dataset was very small and non-normal. For such small datasets, the nonparametric Spearman test is recommended (Neter et al. 2004). Therefore, the Spearman Coefficient was also computed. These results are shown in Table 2.3. The Spearman Correlation Coefficient test confirmed that the three measures of viability were uncorrelated.

Table 2.3: Spearman correlation coefficient for the three measures of viability

Variable	**By Variable**	**Correlation**	**Sig prob**	**Result**
Vigor	Resilience	0.31053	0.1957	Reject correlation
Organization	Resilience	0.08260	0.7367	Reject Correlation
Organization	Vigor	0.04569	0.8526	Reject Correlation

The next step was to establish predictive validity. This would mean that the measure itself could be used to identify the high and low performing projects. As mentioned earlier, the sample for empirical evaluation was random; therefore, there was no prior knowledge about the performance of the selected projects. The three dimensions are plotted in a 3-D plane, as shown in Figure 2.2.

From the plot, it can be seen that project # 14 is low in all three dimensions of viability. Extracting the details about this project showed that this was "Linux NFS development web site" project and was inactive. Similarly, project # 11348, another project with low values in all three dimensions was *GNOME News Applet*, which was a failed or inactive project. It can be seen that project # 84122, 23067 and 29057 have high values for all three measures. When the project description of these groups was extracted, it was found that Project # 84122 was *Azures* (a Bit Torrent Client), project # 23067 was *myPhP* (a database application) and project # 29057 was *Compeire* (an ERP/CRM solution). All three of these projects were among the top projects at the souceforge.net. Thus, the predictive validity of the measures can be confirmed.

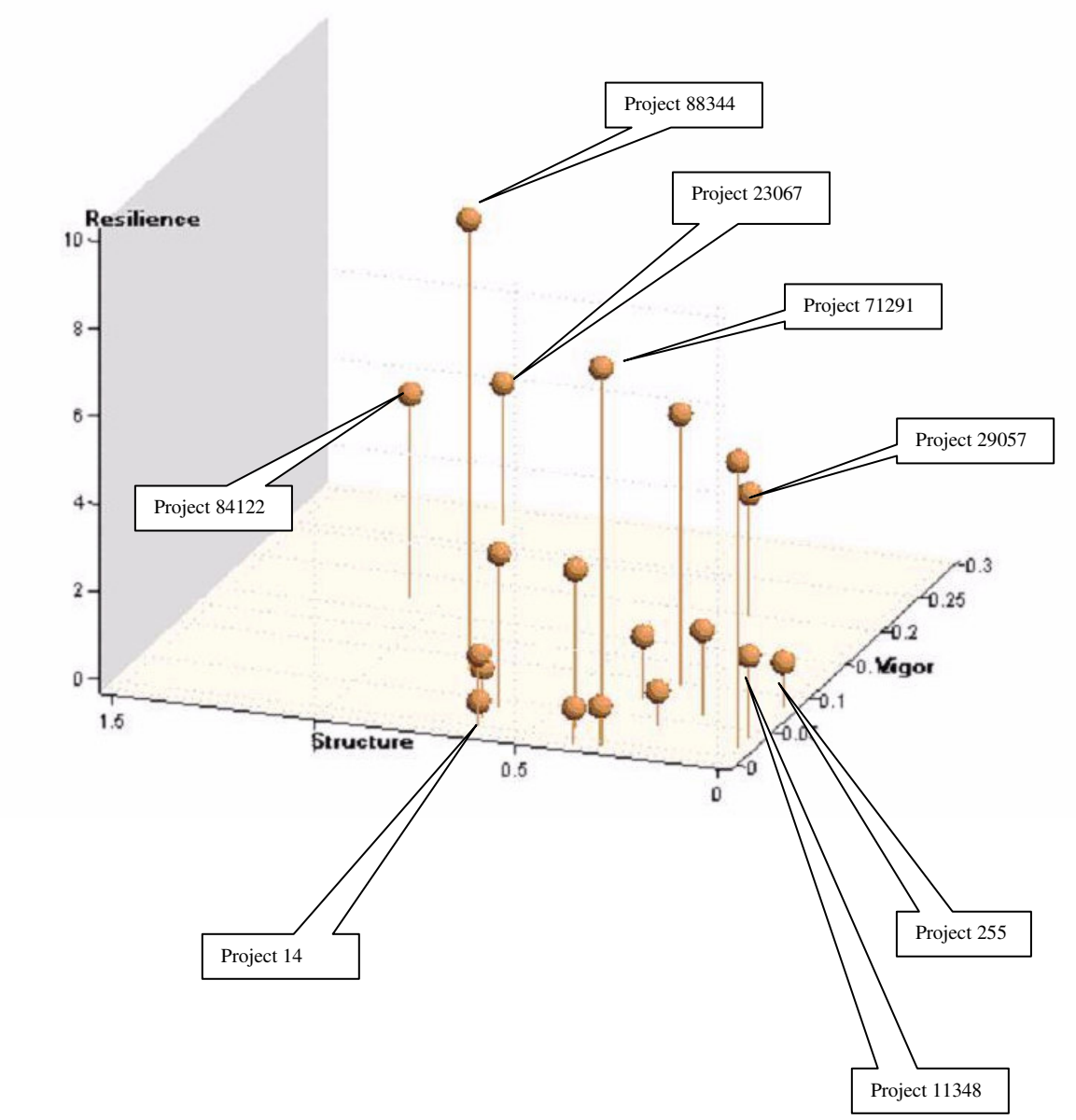

Figure 2.2: Scatter plot of the viability measures for the projects used in the empirical analysis

2.3 APPLICATION OF OSS PROJECT VIABILITY MEASURES

Using each measure of the project viability, three models are developed to evaluate the performance of OSS projects. A detailed analysis of each dimension is performed and the effects of various project variables on these dimensions are investigated. Following is a brief introduction to each model developed in this research. Later chapters discuss these models in detail.

2.3.1 Model of OSS Vigor

This model was developed to explain the factors that affect the vigor of an OSS project. Vigor refers to the growth or the throughput of a project. It is a very critical attribute for a project, especially in its development phase. Software projects transition through various phases before they become stable and operational. Prior research indicates that many OSS projects fail to grow over a period of time and sometimes become inactive (Krishnamurthy 2002). In order to identify the factors critical to vigor, OSS projects that were in their development phase[10] were analyzed. The dataset was extracted from an online OSS project development community. An initial set of factors that effect development performance was identified from existing literature. A refined model was then formulated, tested and validated using OSS project datasets. This model is discussed in more detail in Chapter IV.

.

[10]The Development Phase of software projects refers to software lifecycle phase during which a project is created, new functionality is added, and testing is conducted.

2.3.2 Model of OSS Resilience

The model was formulated to explain the dimension of resilience. Resilience is the ability of a project to react to the changes in its environment. These changes can be; removal of errors, implementation of new functionality, changes made to integrate it with the operational environment, etc. Ability of a project to react to changes during the maintenance phase[11] is a measure of its resilience. Data from OSS projects in maintenance phase, developed through an online community, was used to formulate a model of project resilience. The model is discussed in more detail in Chapter V.

2.3.3 Model of OSS Organization

Organization is the ability of a project to maintain its form and structure in the presence of perturbations[12]. This is the most complex and difficult dimension to measure. In order to investigate the changes in internal structure over a period of time, there was a need for longitudinal analysis of the source code. Therefore, a single large-scale OSS project was used for this study. The purpose of this study was to investigate how various maintenance activities affect the internal structure of an OSS project. This model is discussed in more detail in Chapter VI.

---

[11]Maintenance Phase refers to the lifecycle phase of software after it has been deployed for use. A mature project will be in maintenance phase.

[12]Here perturbations refer to changes that occur in the environment of the project. These changes could be internal e.g. errors or bugs or they could be external e.g. changes in the interfacing hardware.

2.4 CONCLUSIONS

In this chapter, a new measure for the performance of OSS projects was defined, validated and tested. A framework for validation and testing of the measure was established based on the prior literature and research in the area of software measurement. The measures were tested for mathematical, logical and empirical validation. With a measure for performance, businesses can make informed decision regarding adoption and use of OSS projects. OSS development teams can use it, to monitor and control the development and maintenance of their projects. In later chapters the factors, which affect the dimension of viability, are discussed.

CHAPTER III

EXPLORATORY MODEL DEVELOPMENT METHODOLOGY

In this chapter, the research design and methodology are discussed. First, the various characteristics of software considered in the research are discussed. It is followed by a discussion on the research methodology and the suitability of Data Mining for exploratory research. Details of the data source, data extraction and the techniques used in this research are also presented, along with a complete list of the initial variables identified for the analysis.

3.1 FRAMEWORK

Research in software project development and maintenance has focused on three types of characteristics: Process, Product and Resource. In Open Source Software (OSS), however another significant entity in the software lifecycle is the End User. OSS development is characterized by a close interaction between the end user and the development team (Feller and Fitzgerald 2002; Raymond 2001). Therefore, the conceptual framework of this research includes the following attributes:

- Product Characteristics
- Process characteristics
- Resource characteristics
- User/Client Characteristics

Each of these characteristics is discussed in detail in the following sections.

3.1.1 Product Characteristics

The product characteristics refer to the attributes of the software product. Product reliability, maintainability, portability and quality are examples of the product characteristics (Fenton and Pfleeger 1991; Pressman 2004). Affects of product characteristics on the project outcomes have been established in Closed Source Software (CSS) research (Albrecht and Gaffney 1983; Banker et al. 2003; Banker and Slaughter 1995; Eick et al. 2001; Fenton and Ohlsson 2000; Krishnan 1996; Lederer and Prasad 1998; Swanson and Dans 2000). In this research framework, various product characteristics were used to understand how these characteristics affect the OSS project development and maintenance performance.

3.1.2 Process Characteristics

The process characteristics deal with the development process of software. In the CSS domain, a Capability Maturity Model (CMM) has been proposed to measure the software development process. It has been established in prior research that process is directly related to project outcomes (Harter et al. 2000; Herbsleb et al. 1994; Swanson and Dans 2000). However, in OSS there is a lack of formal definition of organization and process. This by no means states that there is no organizational structure. In fact some research has identified the presence of structure and control in OSS communities (Jensen and Scacchi 2005; Krogh et al. 2003; Scacchi 2004a; Scacchi 2004b). In this research framework, process characteristics were used to study their effect on the OSS project development and maintenance performance.

3.1.3 Resource Characteristics

Resource characteristics refer to the nature of the team and tools employed in a project. The higher the number of team members the greater is the human effort and contribution in a project (Fenton and Pfleeger 1991). Such efforts include coding, testing, documentation and user support tasks. Prior research has indicated that resource characteristics impact the software project development and maintenance (Abdel-Hamid 1992; Barry et al. 2006; Boehm 1987). In this research framework, process characteristics were used to understand their effects on the OSS project development and maintenance performance.

3.1.4 User/Client Characteristics

User characteristics refer to the attributes of the end users of the project. The affects of user characteristics on project outcome have not been investigated in CSS research. In CSS, a project is developed for a known user with pre-defined requirements[13] (Brooks 1995) (Pressman 2004). The user does not take a direct and active part in the project development and maintenance. On the other hand, OSS projects are usually initiated by an individual programmer or group of programmers, who are trying to solve a problem that is of their own interest (Dempsey et al. 2002; Feller and Fitzgerald 2002). OSS projects do not have a predefined client or users. However, once a project is launched it is available for public use

[13]Defining user requirements is a challenge in CSS, since user requirements are often incomplete, wrong or over specified. This discussion is beyond the scope of this research.

through the internet. Thus, a user community may emerge which shares the interests of original group of project developers. Project source code is available to users, who may detect, identify and report bugs to the development team. Users are also free to propose solutions, contribute code and make function/feature requests (Feller and Fitzgerald 2002). Users of OSS projects can be developers themselves or a business entity that adopts OSS projects for its use (Feller and Fitzgerald 2002). Considering the critical role of a user in OSS projects, user characteristics were used to study how various user characteristics affect the OSS project development and maintenance performance.

The research framework identified the critical areas that were to be explored in this research. Based on this framework, an exploratory research study was developed to create the models of OSS project performance.

3.2 EXPLORATORY MODEL BUILDING METHODOLOGY

There are two main types of research paradigms: Deductive and Inductive. In deductive research, the researcher establishes a theory about a particular problem. Hypotheses are then generated based on the theory. Data is collected, using some data collection methodology and the original theory is tested to be true or false. On the other hand, inductive research is a competing methodology, which begins with broadly stating a research problem. Data is collected and examined for patterns and knowledge is generated based on the data. In this methodology, hypothesis and theories are created based on the new information (Trochim 1999).

Exploratory studies fall under the category of inductive research. Exploratory research is an important mechanism of generating knowledge, when the problem under investigation is from a new research area and when access to detailed qualitative or quantitative data is available (Hoaglin et al. 1983). Traditionally information systems research has been dominated by deductive research. Existing theories from various disciplines are used to develop hypothesis. Data is then collected through methodologies like a survey or a lab experiment to test the hypotheses. One of the reasons why researchers rely on such confirmatory research is the lack of large size data for conducting exploratory research. For exploratory research, large datasets are needed to examine patterns and to test the models. In CSS projects, access to datasets can be very difficult. Either organizations do not maintain detailed archives of project development activities or they do not provide access to such archives for strategic reasons. On the contrary, OSS community maintains data on a large number of project attributes, primarily because the development is online and detailed records of transactional data are available.

Keeping with the philosophy of free sharing of data, OSS projects provide public access to these project archives. The lack of validated theories and models and the availability of large amount of transactional data, make OSS projects an ideal candidate for exploratory research. If models were to be built and tested based on existing theories in CSS domain, there is a chance that some critical variables might be ignored. Therefore, to create models that explain the performance of OSS projects, an exploratory research through data mining techniques was conducted.

The exploratory philosophy used in the discipline of data mining is not new. Aristotle (b 384BC) and Bacon (1561-1626) advocated the collecting of large quantities of data, exploring them for patterns and then hypothesizing about these patterns. Later Galileo (1564-1642) suggested that scientists should also test these hypotheses. This was a common methodology throughout the 19th century (Press 2003). Later deductive or confirmatory research became more prevalent when the researcher collected data to support a preexisting theory. This research follows Galileo's perspective that knowledge and consequently theories are developed from data.

The following sections discuss various data mining techniques used for the analysis. The details of the data used for the study and selection of the initial variables for analysis are also discussed in detail. The chapter ends with a discussion on how the datasets are used in model formulation. This will provide a structure for the remaining chapters in which the analysis and results of model formulation are presented.

3.2.1 The Knowledge Discovery through Data Mining Process

"Knowledge Discovery through Data Mining (KDD) is the process of using data mining methods to extract knowledge according to the specifications of measures and thresholds, using databases along with preprocessing, sub-sampling and transformations of the data" (Fayyad et al. 1996). Data Mining is a component of the KDD process, which provides the means to extract and enumerate patterns from the data.

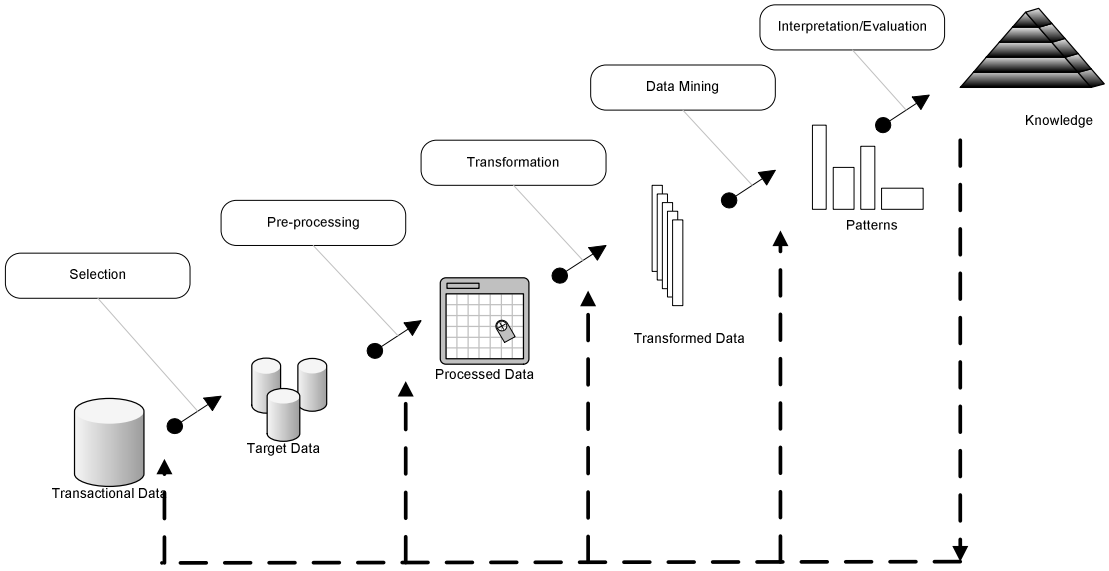

Figure 3.1: The KDD process

The basic steps involved in the KDD process are shown in Figure 3.1. These steps are:

1. *Developing an Understanding of the Application Domain*: It is necessary to have a significant understanding of the problem domain. While conducting exploratory research, the first step is to develop a research problem in a way that useful variables can be identified from the dataset to formulate models. Data Mining is not data fishing; in fact, lack of a clearly defined problem with incomplete knowledge of the domain cannot result in useful models. A researcher needs to develop a clear understanding of the domain and the

supporting areas. This ensures that the data mining process is focused on a valid research problem and that relevant results are generated.

2. *Creating Target Datasets:* Based on the research problem, a target dataset needs to be created. Variables of interest need to be identified and extracted. Too many irrelevant variables can end up delivering misleading results. Similarly too few variables can result in incomplete models.

3. *Data Cleaning and Preprocessing*: This involves operations such as removal of outliers, deciding on strategies for missing values and identifying the sample population.

4. *Data Reduction and Projection*: This involves processes like data transformation and reduction of dimensionality. For this research creation of new variables, using text mining falls in this category.

5. *Data Mining*: This step involves using various Data Mining techniques to search for patterns in data and creating models. This includes supervised methods (e.g. Regression, Neural Networks) or unsupervised methods (e.g. Clustering). The choice of a technique depends upon the nature of the research question and the dataset.

6. *Interpretation of Results*: This step involves analyzing the resulting models, interpreting the results based on domain knowledge, reporting the results and resolving conflicts with previously available knowledge.

In this research all the steps of the KDD process were performed, in order. The research framework discussed in section 3.1, was used to identify the key areas affecting the development and maintenance of projects. Factors that affect the performance of OSS projects were identified from these areas.

The following sections provide a discussion of the data source, variable identification, modeling techniques and assessment methods used in this research.

3.2.2 Data Source

Two separate data sources were used to develop the exploratory models discussed in section 23. The source of data for vigor and resilience models was an online OSS project development community, whereas the source of data for the organization model was a large-scale OSS project. The nature of the research questions warranted the need of two separate data sources. Both of these data sources are discussed below.

3.2.2.1 SourceForge.net Data Source

SourceForge.net (SF) is the world's largest OSS development web site, with the largest repository of OSS code and applications available on the internet. Owned and operated by OSTG, Inc., SourceForge.net provides free services to OSS developers. Project developers use these services to host, develop, and maintain their projects. The SourceForge.net is a database driven web site, which provides historic and status statistics on over 100,000 projects and records of over 1 million registered users' activities. OSTG has

shared certain SourceForge.net data with the research community for the sole purpose of supporting academic and scholarly research on the OSS phenomenon. The Sourceforge.net data archives starting November 1999 through May 2005 were used for this research and was accessed through a research initiative with the University of Norte Dame (Madey 2005).

The SF community hosts OSS projects of various kinds. Typically, the projects developed through this community are small and medium scale OSS projects. The number of projects hosted at SF has increased rapidly over the past few years. The increase in the number of registered projects on SF, as shown in Figure 3.2, indicates the surge in the usage and support of OSS projects.

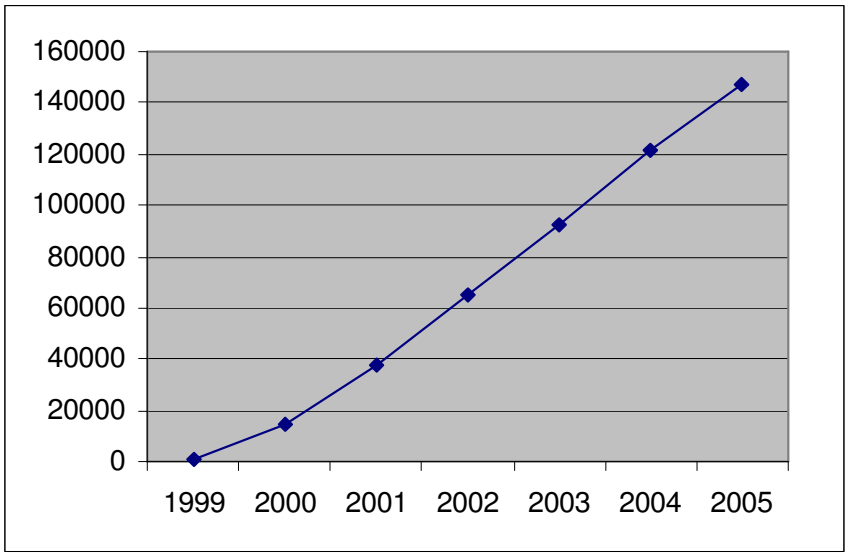

Figure 3.2: Number of projects registered at SF over time

Two datasets were created from the SF projects' data archives: Development dataset and Maintenance dataset. SF maintains information on the lifecycle phase of all the projects. This information was used to classify the projects into the two categories. The development dataset was created for the projects that were in the pre-production life cycle phase, while maintenance dataset was created for projects that were mature or in their production phase, considered as maintenance phase projects. This categorization is very critical in effective model building. The issues and the factors affecting the project outcome in these two categories are different. Research in OSS has used small random samples of projects without any consideration of project lifecycle phase. Many of these studies have failed to discover significant relationships, which could be a result of poor sampling of data (Krishnamurthy 2002; Stewart 2004) . Prior research in software engineering highlights a distinction between development and maintenance phase issues associated to software projects. Therefore, random sampling of the entire dataset is not a suitable approach.

The group identification numbers for the projects in the two categories were use throughout the analysis to extract variables from the SF warehouse. The SF data warehouse consists of over 100 tables and 1000 variables. Each table was studied in detail and the information available was decoded for investigating its usability in this study. Since the data was used from a third party, independent validation of data was performed by random verification of variables with the actual SF dataset. The results were also compared with another independent extraction of variables from the same warehouse, to verify the queries used for data extraction.

The data was imported from the SF research warehouse (hosted by the University of Norte Dame) to a local SQL server relational database. In order to import SourceForge data (available in textual format) to a local SQL server, intensive processing was required. The local SQL server was then connected to SAS Enterprise Miner 5.2 for data analysis. The required variables were extracted from the dataset by SQL queries.

A set of variables of interest was identified based on prior research in software engineering and OSS projects. The dataset contained transactional data on OSS projects hosted by SF from November 1999 until May 2005. The purpose of data analysis was to identify the key factors affecting the performance of OSS projects. Besides ensuring that the projects being analyzed were in comparable lifecycle phases, it was also necessary to ensure that projects had been available at SF for some significant time. Thus, such projects that were created less than a year ago (from the date of analysis) were not considered in the analysis. This was done to ensure that age of the project was not a confounding factor. Some additional transformations (as discussed below) were performed on the data to ensure the validity of the analysis. Extensive SQL queries were used to create the dataset and to generate the variables for each project. Details of each variable are also discussed in later sections.

3.2.2.2 Linux Data Source

In order to study the effects of maintenance activities on the internal organization of OSS projects, there was the need of a single project with sufficient data. The SF dataset consists

of thousands of small and medium scale projects. The maintenance data for these projects was not sufficient for this analysis. Therefore, the source code of project Linux, which is a large-scale OSS project, was used.

Linux maintains a record of its parallel experimental and production versions. The experimental versions are more volatile than the stable versions and tend to change more frequently. The stable versions were suited for this analysis. Linux versions 1.0 through version 2.6.5 were used in this research. Longitudinal data on structural complexity (as an indicator of internal organization) of Linux source code was also extracted for these versions for the Linux kernel released over the past 10 years. An automated tool[14] available online, was used to compute the McCabe's Cyclomatic complexity. McCabe's Cyclomatic complexity measures are the most widely accepted measures of software complexity (Fenton and Pfleeger 1991). Change in the value of McCabe's Cyclomatic measure from one software version to the next was extracted from the dataset. This extracted data was then used to analyze the affects of various maintenance activities on project organization.

Software maintenance activities in the Linux project are implemented thorough *Patches*[15]. Each Linux patch contains rich textual references to the changes it has implemented. These textual references are written by online, geographically dispersed teams

[14]RENAUD's tool, freely available online, was used for this analysis. The tool was tested and validated before use.

[15]A patch is a piece of software code that is added to the existing software, that can add, delete or update the existing functionality of the software and can be used to correct, prevent or perfect any faults that may exist in the software code.

of developers and maintainers to explain significance of the code they have added or removed (Stamelos et al. 2002). The textual information available in software patches was extracted. Although there exits a classification of maintenance activities in literature, yet there is a dearth of any formal classification of software patches. A classification scheme for patches was developed based on the type of maintenance activities performed.

3.2.3 Variable Identification

To create useful models, it is critical to identify the initial set of variables that will be used in the Data Mining process. Selection of too few variables can result in an incomplete analysis and may result in excluding critical factors from the final model. On the other hand, inclusion of irrelevant variables can adversely affect the model building process and can affect the correct identification of the significant factors. The identification of the initial set of variables for use in the Data Mining process requires domain knowledge and a deep understanding of the dataset (Fayyad et al. 1996). Prior literature in software engineering was consulted to identify the critical measures. The OSS literature was also consulted to identify the unique characteristic of the OSS paradigm. This information was used to ensure that OSS relevant features were included in the analysis. The SF data warehouse contains over a thousand variables. The data dictionary of this warehouse was examined in detail to understand the layout of the data and to extract the correct measures from it. All variables were categorized as per the framework discussed earlier. A listing of the variables along with a brief summary and operationalization detail is given in Tables 3.1-3.5.

Table 3.1: Product related variable measurement and sources identified for analysis

Variable	Summary	Measure	Source	Symbol
Functionality	Functionality refers to the number of functions being offered by the software. Functionality has been used in CSS models. Increase in product functionality is attained through new releases over project lifecycle (Boehm 1987).	Increase in features	Count feature request closed	*Cnt_Feat*
		New Modules	Count File Release	*Cnt_mod*
Maintainability	Maintainability refers to the extent to which software is maintainable. Maintainable software should not be dependent upon a small group of people who understand it. In OSS the ability of users to be able to understand and maintain code is very critical (Samoladas et al. 2004). If an OSS project is not maintainable, then detecting and removing bugs can be a problem.	Number of distinct members reporting the bugs	Count Distinct Submitted_by	*Cnt_rep*
		Number of distinct members fixing the bugs	Count Distinct user_id closed_by	*Cnt_fix*
Portability	The portability of a software project indicates the flexibility of project use. A software that runs on multiple platforms offers more flexibility to the user compared to a platform dependent software(IEEE-STD-1061 1993).	Number of platforms supported	Count Operating Systems	*Cnt_OS*
		Number of prog; languages supported.	Count prog; languages	*Lang*
License Type	OSS projects are launched under various licenses. The most common license is the OSI license. Prior research indicates that license choice can affect the development performance of OSS projects (Stewart and Ammeter 2006).	License type	OSI (Y/N)	*OSI*

Table 3.1: *Continued*

Variable	Summary	Measure	Source	Symbol
Project Type	SF classifies projects based on various aspects, e.g., games, application file transfer protocols, desktop applications, operating system, etc. This variable was used to check some types of projects are more suitable for development in OSS community compared to others.	The text Description of the project was used to create categorization for project type, using text analysis	200 word, textual description of the project	*Prj_Type*
Usefulness	Usefulness of a software project depends upon how relevant the product is to the customer. The end user determines the usefulness of software and makes a decision regarding procurement. For a free OSS product, usefulness is a measure of user community's interest in that product.	Downloads	Count downloads	*Downloads*
		Page Views	Count Page views	*Cnt_pgV*
Product Compatibility	Compatibility in OSS projects will make them more likely to receive code contributions from developers of other projects. Compatibility of OSS projects will refer to the diverse audience the project can attract.	Number of translations	Count Number of translations	*Trans*
		Number of platforms supported	Count Operating Sys;	*Cnt_OS*
Usage	When software becomes operational, the probability of finding the existing errors will be linked to the usage of the software. More software is used, the chances of finding existing errors increases. If the usage of software is low then a low number of bugs do not necessarily imply that there are no bugs. Thus software usage will be critical to its performance measurement (Delone and McLean 1992).	End user usage	Bugs reported by end user	*Bug_User*
		Usage of the project in general, that results in fault detection	Bugs Open	*Bugs_Open*

Table 3.2: Process related variable measurement and sources identified for analysis

Variable	**Summary**	**Measure**	**Source**	**Symbol**
Project Management	Use of traditional Project Management (PM) methods include a designated project manager and centralized task allocations. These activities can affect the outcomes of a software project. Though OSS projects are developed in a more informal environment, yet projects may use PM (Jensen and Scacchi 2005).	Whether an OSS project decides to have a project manager or not.	Use PM (Y/N)	*Use_PM*
Process Quality	The overall quality of development and maintenance process can affect the performance of a project. In CSS, there is evidence that process quality of a project has a positive impact on its performance. No such analysis has been performed for OSS before (Banker et al. 1998).	The response time to fix an error	Mean Time to fix a bug (MTTR)	*MTTR*
Configuration Management	In CSS literature, use of Configuration Management (CM) techniques has been linked to better performance of software (Herbsleb et al. 1997; Humphrey 1989). OSS projects use configuration management tools during development and maintenance. There is extensive use of version control tools. The effects of CM on project performance will be investigated.	Use of CVS	Use CVS (Y/N)	Use_CVS
		Number of Concurrent Version Control Systems (CVS) commits	Count CVS commits	Cnt_CVS

Table 3.2: ***Continued***

Variable	**Summary**	**Measure**	**Source**	**Symbol**
Communication Channel	Availability of various methods of communication between the developers and the users can affect the performance of the project (Herbsleb and Moitra 2001).	Number of forums	Count	*Cnt_Forum*
		Use of mail messaging	Use Mail (Y/N)	*Use_Mail*
		Use of news groups	Use News Groups (Y/N)	*Use_News*
Requirement Implementation	CSS project performance is associated to its ability to conform to user requirements (Pinto and Slevin 1987; Pinto and Slevin 1988; Schonberg 2000). In OSS, there are no predefined requirements, yet the end users can make requests for implementing new features to the projects. Therefore, this factor will be used in the models.	Response time to feature requests	Time to implement a feature	*MTIF*
Process Quality	The ability of a project to detect and remove bugs is a reflection of the quality of the maintenance process.	Ability to detect bugs	Bug repository, Count of bugs	*Bug_Cnt*
		Inability to remove problems that occur	Bugs that are not fixed	*Bug_Open*

Table 3.3: Resource related variable measurement and sources identified for analysis

Variable	Summary	Measure	Source	Symbol
Effort	The size of the project team will be indicative of how much effort is available to the development and maintenance process. There are conflicting views on the size of a team. Some researchers support the view that a large team size will have more effort available to the development and maintenance process (Abdel-Hamid 1989), while others argue that a large team size can cause a negative effect on performance(Brooks 1995) .	Number of registered developers for the project	Team Size	*Cnt_Team*
Team Communication	Conway [16] suggested that communication patterns of teams are reflected in the products they produce (Conway 1968). Effects of team communication on development and maintenance performance will be analyzed.	Frequency of development team communication	Messages posted at development forums	*Cnt_Posts*

[16] "Organizations which design systems are constrained to produce system which are copies of the communication structures of these organizations" - Conway's Law.

Table 3.4: User related variable measurement and sources identified for analysis

Variable	Summary	Measure	Source	Symbol
User Type	OSS projects are developed for various types of end users. Some projects are developed purely for a development community. Others are developed as end user applications that can be used by non-programmers too. Considering the nature of OSS development, the type of audience would affect the extent of end user involvement. If the end users were programmers, they would be able to modify the code.	Nature of the end user	Audience Programmer (Y/N)	*AUD (o= prog, 1 = non-prog)*
Activity Level of User	OSS user can be an active member of the development and maintenance community. They can also contribute to the source code and participate in the detection and removal of bugs. Anecdotal references to active user community have been made in literature, but no empirical testing has been performed to investigate its effects on project performance (Feller and Fitzgerald 2002; Scacchi). This research used the activity level of the user in project development and maintenance performance. This variable has not been used in CSS models.	Frequency of end user interaction	Forum posts by users	*Cnt_Msg*
		Number of users interacting	Number of distinct individuals posting messages, bugs or feature requests	*User_Int*
Community Size	Using the argument that the active user has an impact on the project performance, the effects of the size of the user community will also be used for the model. A larger community will imply more effort going into the development and maintenance process. This will also test the Linus law, which states, "Given enough eye balls, all bugs are shallow." (Raymond 2001).	Number of active users	Number of distinct senders of messages	*Cnt_User*

Table 3.5: Control variable measurement and sources identified for analysis

Variable	**Summary**	**Measure**	**Source**	**Symbol**
Controls	The size and of software projects differ and can affect its performance. Software engineering models typically use software size and age as a control measure to account for the affects of size on project outcomes (Banker and Slaughter 1995; Barry et al. 2006; Brooks 1995).	Size	Source Lines of Code	*Size*
		Age	Time elapsed since the start of project	*Age*

3.2.4 Data Sampling

In exploratory research, it is very important to use different samples from the dataset for model building, validation and testing (Fayyad et al. 1996). If the same data is used for model creation and validation, the resulting model will likely be biased to the sample and thus not acceptable. Therefore, both the SF datasets (for development and maintenance) were split into training, validation and testing samples. This was done to ensure that a valid model is created that would be applicable to OSS projects in general.

Initially, the training set was be used to train or build the model. Once an acceptable training model was built, the validation set was used to evaluate the model. A comparison was then made with specific diagnostics e.g. lift charts, to check how well the training model holds for the validation sample. At times, there were several iterations of re-training before a reasonable model was selected. Once a model was selected, the validation dataset can no longer be used to test the accuracy of this model. To create a robust model, the final training model was applied to the test data. The accuracy of the model on the test data gives realistic estimate of the performance of the model for OSS projects in general. Figure 3.3 describes the process of data sampling.

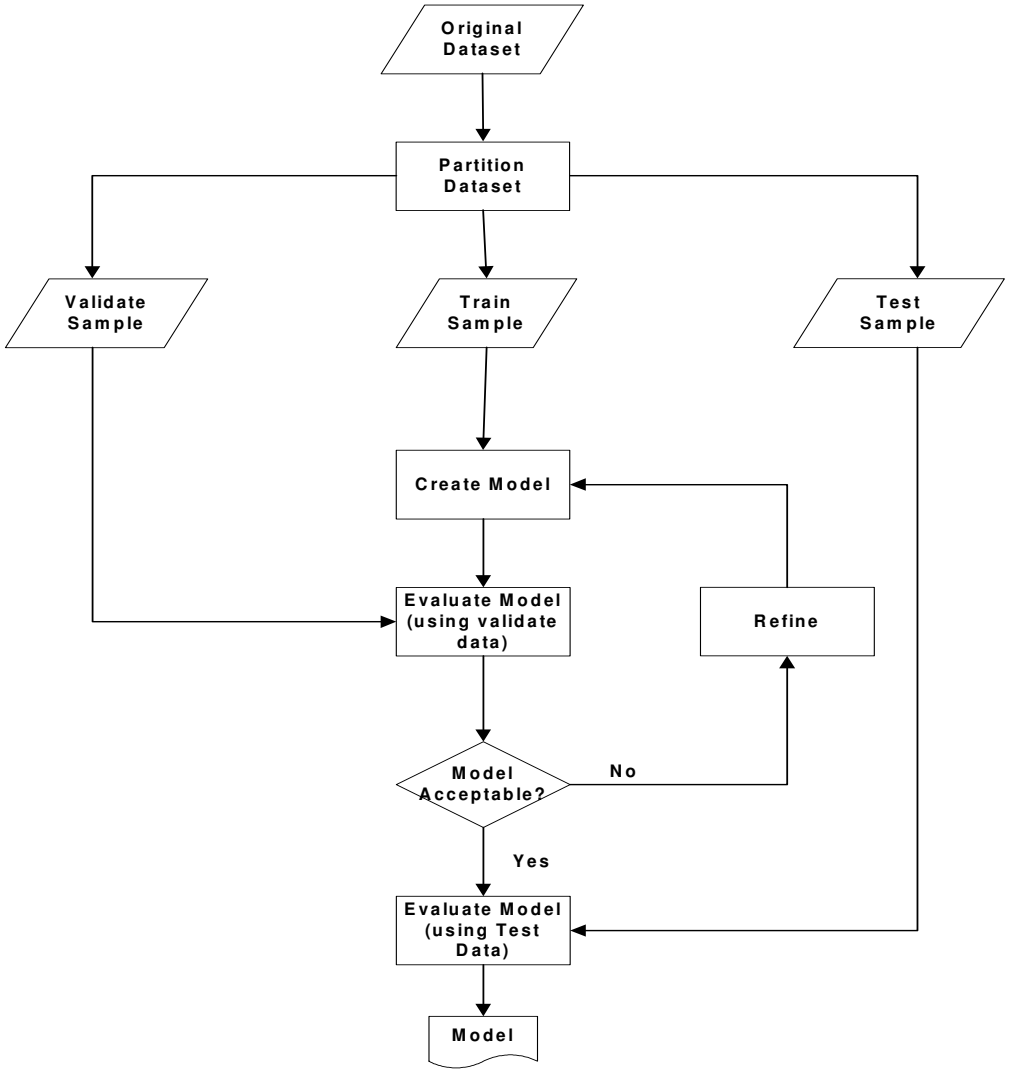

Figure 3.3: Process of data sampling used for data mining

3.2.5 Data Mining

Data Mining (DM) is the search for patterns of interest in observed data. The two main goals of DM are Prediction and Description. There are two main types of DM techniques available: Supervised Learning and Unsupervised Learning. The supervised learning methods are used when the target (or the dependent variable) is known, e.g. predicting the income of a group (target variable) based on historical information about certain variables. The most commonly used supervised DM methods are Regression, Decision Trees and Neural Networks. Unsupervised learning techniques are used to identify patterns in data with no predefined target. An example of this technique is developing taxonomies based on available variables. The most common unsupervised learning technique is cluster analysis (Berry and Linoff 2004).

In this research, three predictive DM techniques; Logistic Regression, Decision Trees and Neural Networks were used. Clustering was also performed on textual data. SAS Enterprise Miner 5.2 was used to perform the data mining tasks. These techniques are discussed below.

3.2.5.1 Logistic Regression

Logistic Regression is a predictive modeling technique that is typically used when the outcome variable is binary or dichotomous. There are a few differences between Linear Regression and Logistic Regression models and assumptions (Hosmer and Lemeshow 2000). The first significant difference is the nature of relationship between the independent

and dependent variables. If *Y* denotes the dependent variable and *x* denotes the independent variable, then the general regression equation is:

$$E(Y \mid x) = \beta_0 + \beta_1 x \ . \qquad \text{..... (3.1)}$$

Where *E(Y|x)* is the conditional mean or the expected value of *Y* given the value of *x*. In this expression *E(Y|x)* can take any values between $-\infty$ and $+\infty$. With dichotomous dependent variable, the conditional mean must be greater than, equal to zero and less than, or equal to one. Many distribution functions have been proposed for the analysis of such a variable, such as Logistics Distribution Function. When used, the conditional mean of *Y* given *x* can be represented as $\pi(x) = E(Y \mid x)$. The Logistic Regression model may be represented as:

$$\pi(x) = \frac{e^{\beta_0+\beta_1 x}}{1+e^{\beta_0+\beta_1 x}} \qquad \text{..... (3.2)}$$

A transformation of $\pi(x)$ called the *Logit Transformation* has many of the properties of Linear Regression (e.g. it can be continuous depending on the values of *x*). The Logit Transformation is defined as;

$$g(x) = \ln\left[\frac{\pi(x)}{1-\pi(x)}\right] = \beta_0 + \beta_1 x \qquad \text{……..... (3.3)}$$

Logistic regression fits a linear model to the log of the odds of the response, the logit transformation. The logit transformation forces the predicted values for the fitted model to

be between *0* and *1*, thus giving a predicted probability for the different levels of the binary variable.

The general form of the Logistic Regression equation is:

$$logit\left[\pi(x)\right] = \log\left\{\frac{\pi_h}{1-\pi_h}\right\} = \alpha + \sum_{k=1}^{t} \beta_k x_{hk} \qquad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \quad (3.4)$$

This is the log odds of one to zero of the h^{th} subpopulation.

The testing for the significance of coefficients is also different for logistics regression models. The purpose of significance testing of a variable is performed to ascertain whether adding that variable to the model helps explain the outcome more precisely or not. This is done by comparing the observed values of the outcomes, with and without the variable under test. If the addition of the variable in question improves the fit of the model, it is considered significant. The overall fit of the model is tested by goodness-of-fit tests (Hosmer and Lemeshow 2000).

3.2.5.2 Decision Trees

A decision tree is a structure that divides up a large collection of observations, into smaller sets, by applying a sequence of simple decision rules (Berry and Linoff 2004). Decision trees produce a set of rules that can be used to generate predictions for a new dataset. Decision trees are one of the most popular methods of predictive modeling for data

mining because they provide interpretable rules and logic statements, enabling more intelligent decision-making.

A decision tree partitions data into smaller segments called terminal nodes or leaves which are homogeneous with respect to a target variable. Partitions are defined in terms of other variables called input variables, thereby defining a predictive relationship between the inputs and the target. This partitioning continues until the subsets cannot be partitioned any further using user-defined stopping criteria. By creating homogeneous groups, analysts can predict with greater certainty how individuals in each group will behave. Various algorithms are available for splitting data into decision trees. One popular method is the Chi-Squared Automatic Interaction (CHAID). This technique has the advantage that the independent and dependent variables can be nominal, ordinal or interval. The splitting criteria are based on variance reduction and F-test splitting for interval targets. If the target is categorical then the CHAID or entropy reduction[17] can be used (Fernandez 2003). Generally, they fit linear and non-linear relationships.

3.2.5.3 Neural Networks

An artificial Neural Network is a network of many simple processors, each possibly having a small amount of local memory. Communication channels that usually carry numeric (as opposed to symbolic) data encoded by various means connect the units. The units operate only on local data and on the input they receive via the connections. Neural

[17] Also known as Gini

Networks are universal approximators and they can be trained for a specific application and used to extract patterns or detect trends. There are many available structures of the Neural Network. One of the popular architectures is the Multi-Layer-Perception (MLP). MLP consists of an input layer, a hidden layer and an output layer (Berry and Linoff 2004).

Neural networks are useful tools for interrogating increasing volumes of data and learning from examples to find patterns in the data. By detecting complex nonlinear relationships in the data, Neural Networks can help make accurate predictions about real-world problems. To avoid the tendency of Neural Networks to over fit the training data, model performance is constantly assessed against the validation data, and the final model is selected based on one of the several criteria that users can select (e.g., the minimal validation error, maximum total profit, etc).

Neural network models have the advantage of a high predictive power because they can fit non-liner models. The disadvantage of this technique, especially for academic research, is the difficulty in interpretation of the results. The logistic regression represents a model in the form of an equation. The significances of the regression coefficients explain the phenomenon under study.

With decision trees, the result is a sequence of English language rules that are easy to describe and understand. Neural network models on the contrary do not have a simple formulation. They provide the fit statistics and the prediction of new values. However, the individual variable significances and contributions to the model are not easy to interpret. Neural network models are recommended for use in problems where the dependent and

independent variables are well understood e.g. in credit scoring models. In exploratory research, when the factors effecting the outcome are not completely known, the use of Neural Network as the final model is not recommended (Berry and Linoff 2004).

However, in this research used all three models simultaneously in the model formulation phase. Results from decision trees are very insightful regarding the interactions between the independent variables. The neural network models can be used as a base line for detecting complex non-linear relationships. The performance of the logistic regression model can be improved by decision tree and neural network results. Chapters IV and V discuss the details of the application of this technique.

3.2.6 Text Analysis

Text Mining refers to the discovery of knowledge from text data. Text Analysis has been widely used in sociology and communication literature. It converts text into numeric form that can be used in analysis. There are many types of text analysis algorithms available.

The first step in text analysis is to identify the target dataset. It is also very critical to have a defined task e.g. clustering or categorizing. Once the data has been collected, all unique words in each document are identified. A stop list can be used to ignore the terms that are naïve and are not to be used in the analysis. Another approach is to create a start list. This is a list of the terms considered in the analysis, with all other terms ignored. Word stemming is then used to stem e.g. walk, walking, walked etc. are all treated the same. Furthermore a list of synonyms can be created, and nouns and verbs can be separated (Chiarini-Tremblay et al. 2005).

The next step is to create a Word Frequency Matrix (WFM). This term-by-term frequency matrix can be improved by utilizing weighing functions words. For example, words used more frequently have a higher weight. The words that have a high correlation to the outcome (target) variable may be assigned a higher weight. Depending upon the size of the data, this matrix can become very large. Not all the terms appear in all the documents, resulting in wastage of significant computing resources. In order to reduce the dimensionality of the WFM, Latent Semantic Indexing (LSI) is used. LSI is a technique that can transform a matrix into lower dimension form (Berry and Browne 2005). LSI uses Singular Value Decomposition (SVD) for reduction of dimensionality. The matrix is decomposed into Eigenvalues and Eigenvectors. This creates linearly independent components of the data. The smaller components can then be ignored and relationships between two documents can be determined by the remaining components (Berry and Browne 2005; Deerwester 1990).

To reduce the initial term-by-term matrix, a weighting scheme has to be implemented. There are several techniques available for this purpose. Term frequency weighing assigns a weight to each term, based on its frequency[18]. A word that has a high frequency of occurrence in a document has a high weight. One drawback is that there is no reflection on the importance factor of document discrimination. Another method is the Inverse Frequency Weighting[19] This method assigns a weight according to the frequency of

[18] $W_{ij} = Freq_{ij,}$, where $Freq_{ij}$ = number of times jth term occurs in document D_i.

[19] $W_{ij} = log_2(a_{ij} + 1)$, where a_{ij} is the frequency with which term i appears in document j.

a word in a single document compared to its occurrence in the entire collection of documents. If a word has high frequency occurrence in a single document, but has a low occurrence in the collection of documents, then the weight is high. The advantage to this scheme is that there is a reflection of the importance factor for document discrimination. However, the assumption is that the terms with low frequency in the document collection are better discriminator than those with high frequency. For unsupervised clustering tasks (like the one in this research), the best method is the entropy method (Chiarini-Tremblay et al. 2005). The formulation of this method is:

$$W_{ij} = \log(Freq_{ij} + 1.0)(1 + entropy(w_i)) \quad \text{..... (3.5)}$$

Where

$$entropy(w_i) = \frac{1}{\log(N)} \sum_{j=1}^{N} \left[\frac{Freq_{ij}}{DocFreq_j} \log\left(\frac{Freq_{ij}}{DocFreq_j} \right) \right] \quad \text{……..... (3.6)}$$

Entropy in text analysis refers to the amount of information added by the text. It is based on the Information Theory approach from Shannon's work. It is the amount of information that a word contains about the entire document, i.e. knowing one word, what is the probability that another word will occur. Entropy measures the amount of information in a random variable. It is normally measured in bits (hence the log to the base 2), but using any other base yields only a linear scaling of the results (Manning and Schutze 2002).

This research uses the entropy minimization algorithms. Primarily, text mining was used to create clustering of textual datasets. The purpose was to classify the documents into similar clusters, based on the word counts and relationships. For the model of Vigor and Resilience, the SF projects were categorized into project types, based on a textual description. Although SF provides categorization of the projects based on various attributes, there is no single classification of all projects. Use of text mining avoids the multiple classifications of the projects. It creates information from the available textual data. Each SF project maintains a 200-word description. This description was used to create a project type variable for the development phase and maintenance phase projects. Projects were split into clusters based on description terms. The terms in each cluster indicates the nature of these projects. The cluster ID was used as the *Prj_Type* variable for each project in the analyses. Later chapters discuss the results of the text mining for project type.

For the Linux dataset, text analysis was used to create a taxonomy for the OSS patches. The maintenance patches for Linux releases were used as text documents. Currently there is no available taxonomy of software maintenance patches. Therefore, a new taxonomy (discussed in chapter VI) was developed for Linux patches. The affects of these patches on code complexity were also studied.

CHAPTER IV

MODEL OF VIGOR

This chapter discusses the model for OSS projects in their development phase. First, it presents a background of the significance of vigor in the development phase. It is followed by a discussion on the model building and selection process. The selected model is then evaluated and presented. The variables significant in the final model are explained in detail.

4.1 BACKGROUND

The development phase of software projects refers to its lifecycle phase during which a project is created, new functionality is added, and testing is conducted (Pressman 2004). Successful execution of software project development has been a challenge. Researchers and practitioners have been in search of methods, which can help in managing the high failure rates of such projects. According to prior research, almost 90% of software development projects fail to complete within budget, on time, and according to customer requirements (Abdel-Hamid 1988; Boehm 1984; Boehm 1997; Pinto and Samuel. J. Mantel 1990; Pinto and Slevin 1988). Many models have been proposed which identify the factors that affect CSS project development (Godfrey and Tu 2001; Jeffery 1987; MacCormack et al. 2004; Paulson et al. 2004). These project evaluation metrics consist of measures (e.g. cost, user requirements) that have no meaning in the Open Source Software (OSS) domain. Recent reports suggest that Fortune 500 companies are considering adoption of the OSS

projects in near future. Thus, there is a need for a model to evaluate OSS project performance during its development phase. This research has formulated such a model.

OSS projects are developed as freeware through volunteers. They evolve as the development community and the functionality of the project grow (Scacchi 2002). Much of the empirical research in OSS domain has been focused on large-scale OSS projects of high quality (Godfrey and Tu 2001; MacCormack et al. 2004; Paulson et al. 2004). Ideally, projects would evolve over a period of time and develop into mature software, yet many OSS projects fail to do so (Krishnamurthy 2002). During the development phase, the most critical aspect is the ability of a project to grow in functionality so that it can transition successfully to its next lifecycle phase (Kemerer 1987). In this research, the performance of an OSS project during its development phase was measured in terms of its Vigor.

The model was built using the framework and methodology discussed earlier in Chapter III. The analysis and results of the model are presented in the following sections.

4.2 MODEL BUILDING

Data Mining techniques were used for model formulation, validation and testing. To identify the factors that affect the performance of OSS projects in their development phase, the SourceForge (SF) dataset was used. The variables identified earlier in Tables 3.1-3.5 were extracted for the SF projects in their development phase.

There are three predictive modeling techniques of Data Mining: Logistic Regression, Decision Tree and Neural Networks, each with its own strengths and weaknesses. In this research, all three techniques were used to improve the model and its interpretation. These techniques were not only used to discover models that could predict outcomes, but also to get a better understanding of the variables that affect the OSS project performance. The best technique was selected based on its performance and the ease of explanation of the phenomenon.

The dimension of Vigor was used as the outcome (target) variable. During the development phase, the biggest challenge for OSS projects is continued growth and transition through the development phases (Crowston et al. 2003). Vigor captures the growth of the project; therefore, it was selected as the outcome variable. The vigor was operationalized as the transition of a project from one development phase to the next. This data was extracted by examining the development phase for each project. Projects with multiple phases were separated. The dates on the change in phase were used to identify the projects that had evolved. At times, some projects had selected multiple phases reported at the same date e.g. alpha and pre alpha. This anomaly could be attributed to some internal definition problems. In order to remove this anomaly the highest level of development was selected for multiple entries on the same dates. Queries were used to detect multiple phase transitions. Only two projects had evolved through more than two phases while being hosted by the SF community. The dataset was validated using alternative queries and was cleaned to contain correct phase transitions.

Another research issue was to investigate the affects of end user involvement in the testing process. In the initial dataset, there were a large number of projects in the development phase, with no bugs ever reported. In OSS projects reporting of bugs is a very critical component of project development (Capiluppi et al. 2004). Upon a deeper analysis, it was revealed that these projects did not have a bug repository set up. Starting an OSS project at SF is a very simple process. Any registered user can start a project. The SF database contained a large number of projects that were never intended to be real development projects. Some registrations were not valid and had no useful data associated to them. This is the reason, why a random sample taken from the warehouse cannot generate good results and it cannot be used for model building. Therefore, data was reduced to the projects that had at least one bug reported. This meant that the project had a bug repository set up and had a working system for bug reporting. With all of the above-mentioned updates, the resulting dataset for this research contained 4931 projects.

To ensure effective model building, data cleaning is necessary. Missing data and outliers can cause serious errors in the models. Logistic Regression and Neural Networks techniques ignore the observations with missing values. In this dataset, it was also critical to analyze and understand the reason behind the missing values. The use of projects with at least one bug reported, eliminated most of the dummy projects that could have affected the validity of the analysis. The data was inspected again for missing values. Projects with no forums or no messages in the forums were also examined closely to ensure that only valid projects are included.

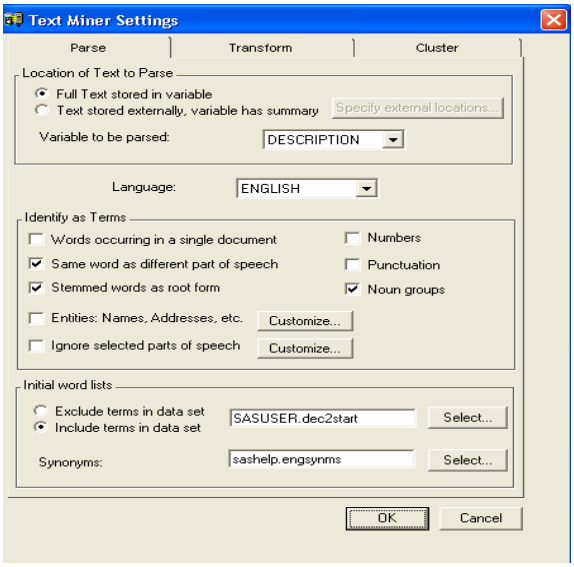

Figure 4.1: Text miner settings used for creating the new variable

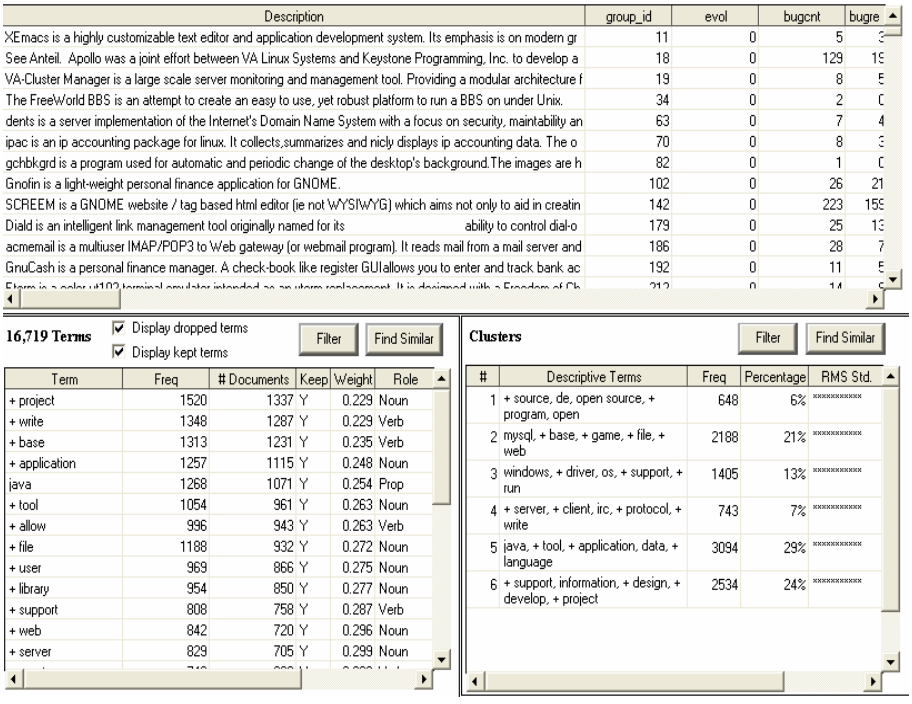

Figure 4.2: Text miner output for the project type data

Once a clean dataset was available, it was imported to the SAS Enterprise Miner 5.2. The SAS dataset was explored using the insight node. The insight node[20] provides useful features like Descriptive Statistics and Collinearity Analysis. The data was explored for missing or incorrect values. The distributions of the variables were observed, to get a better understanding of the data. The dataset was then merged with the text analysis data on project description.

The project description textual data was used to create a new variable called "project type". The dataset of project description for all the projects in the development phase was used in SAS Text Miner to create the new variable. Initially, the default stop list was used on the dataset. The stop list contains the words that are ignored while the text analysis is performed. The default list contains most commonly occurring words that do not carry information about the text being analyzed. The initial run with the default stop list resulted in generating a word frequency table. For such a large amount of data, performing an initial run with a default stop list is beneficial[21]. A new start list was created by removing the words that were not considered a project description or added no usefulness to the analysis e.g. frequent words like where, upon or abbreviation like en, dl etc. The new list with "keep terms" is saved as a new start list and is used in the final analysis.

[20] Insight node is no longer available in SAS EM 5.2. The older version SAS EM 4.3 was used for using this node.

[21] The product designer of Text Miner, Mania Mayes agrees with this approach, in fact recommends it for large datasets.

The Text Miner node was set to automatically cluster the terms (see Figure 4.1). The options were set to generate the Singular Value Decomposition (SVD) terms and to perform clustering based on the SVD dimension. A maximum number of 40 clusters were allowed. The term stemming option was set to "Yes". The frequency weighing method was "Log" and the term weighting method was "Entropy". Expectation maximization algorithm was used for clustering. This algorithm is best suited in cases where the expected number of categories is unknown.

The text miner node provided the resulting clusters . Terms identifying each cluster along with occurrence frequency and percentage among the input documents were also produced. Each project belonged to only one cluster. The result of the cluster analysis is shown in given in Table 4.1 below.

Table 4.1: Descriptive terms of the cluster analysis results of the project type data

Cluster	**Descriptive Terms**	**Frequency**	**%age**
1	Source, de, open source, program, open	648	6%
2	Mysql, base, game, file, web	2188	21%
3	Windows, driver, OS, support, run	1405	13%
4	Server, client, irc, protocol, write	743	7%
5	Java, tool, application, data , language	3094	29%
6	Support, information, design, develop, project.	2534	24%

Each of the clusters represents the type of a project, based on the project description. It can be seen from the description terms that Cluster #1 has general-purpose Open Source terms. Cluster #2 has terms referring to database and game programs, Cluster #3 has terms associated with operating systems, Cluster #4 has terms related to network communication, Cluster #5 refers to tool and application development while Cluster #6 has more description regarding project development and design.

The cluster number associated with each project was imported to the main dataset for further analysis. The Cluster_ID, defines the new variable of project type (Prj_Type)[22]. This data was used in model formulation to strengthen the model. Once the text mining results were merged with the original dataset, the new data was ready to be used for model building. The process flow of the model building process is shown in Figure 4.3.

The next step was to create the outcome variable and to perform the necessary transformations on the dataset. The transformation node was used to create the required transformations. The time dependent variables i.e. number of downloads and the total size of the files released was normalized for the project age. Since the average time to fix a single bug was being used, there was no need to normalize the bug fix time with age. Age is an important characteristic of a project and affects the performance variables. Therefore, the time dependent variables e.g. number of downloads and total file size were divided by the

[22]Each group has a Prj_TypeX, where X = Cluster_id.

age of the project. This was done to ensure that the analysis is valid and there are no confounding affects of age of the project on the performance. The value of vigor of projects that evolved was set to 1 (Vigor = 1) and of those that did not evolve was set to 0 (Vigor = 0). The resulting distribution of the target variable is shown in Figure 4.4.

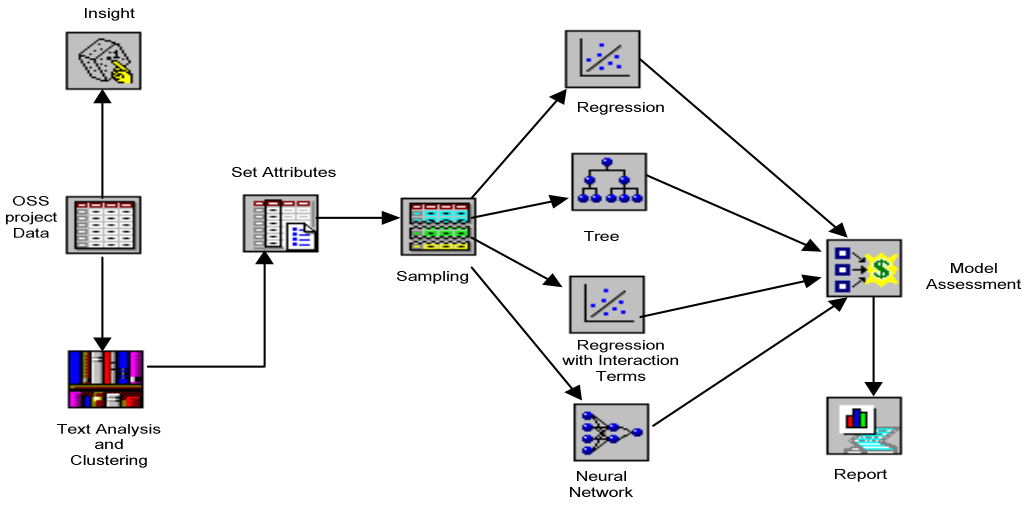

Figure 4.3: Flow diagram of model building process

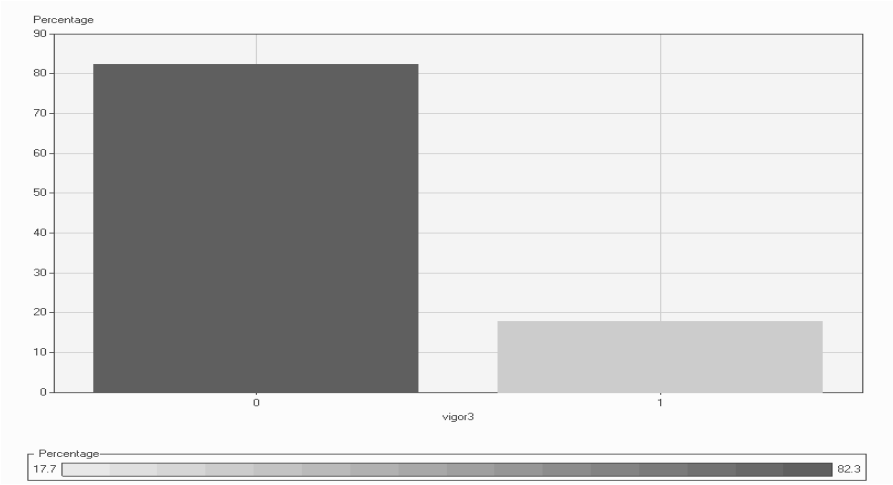

Figure 4.4: Description of target variables of vigor

As can be seen, only 17.7 % of the total projects had a high vigor. Further analysis revealed that three projects that had multiple transitions during their lifecycle[23]. The status of the variables that would be used in the analysis was set to "Use" and others, which were not to be used in the analysis (i.e., the variables used for transformations), were set to "Not use".

Next, sampling node was used to create the data splits. The dataset was split into 40% training, 30% validation and 30% test samples. The details of why and how the

[23]Data on projects that were inactive was also identified, however no considerable difference was found for these projects. There could be future research on projects that evolved and projects that failed, but it would result in a much smaller sample, therefore a greater knowledge of the process is required for this endeavor.

sampling was performed have been discussed in Section 3.2.4. Since the percentage of the observations with high vigor was very small, stratified random sampling was performed to ensure that each spilt is an accurate representation of the actual population.

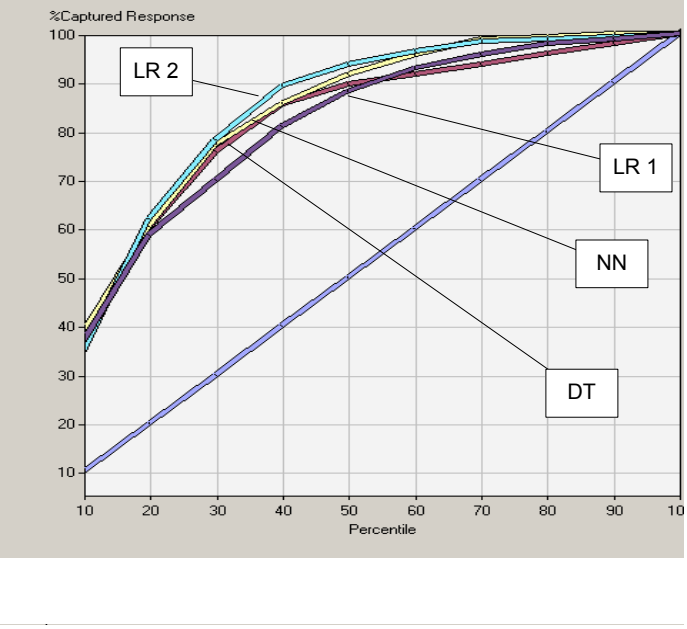

Figure 4.5: Lift Charts for all techniques used for building model for vigor

The partitioned datasets were used with predictive modeling techniques to cerate models. The initial analysis was performed using a Logistic Regression, Decision Tree and Neural Networks node. Stepwise Logistic Regression was used to select the variables in the model. Though stepwise regression is not recommended for theory testing, it is widely used in exploratory research. In exploratory research there is no a-priori assumptions regarding

the relationships between the variables, and the goal is to discover relationships (Hosmer and Lemeshow 2000; Menard 2002). The Neural Networks was used with three hidden nodes and Decision Tree was used with the default settings. The results indicated that the Logistic Regression model was the worst in performance. The Neural Networks was the best model and the Decision Tree was better than Logistic Regression. The lift charts of this analysis are shown in Figure 4.5.

Neural Networks can provide a good fitting model, but the interpretation of the model is very hard. Neural Network results do not specify any significances or coefficients for the variables used. On the other hand, a Decision Tree provides a model that can be easily interpreted as simple English rules. A Decision Tree is very effective in using the interactions, which exist between variables. The poor performance of Logistic Regression in the analysis indicated that there were missing variables in the Logistic Regression model.

In this study, the Neural Networks and Decision Tree results were used to improve the performance of the Logistic Regression node. The results of the Neural Networks and Decision Tree analysis are shown in figure 4.6 and 4.7, respectively. A better fitting of the Neural Networks node is indicative that there may be non-linearity in the relationships. The Decision Tree on the other hand could be indicative of interaction terms.

Figure 4.6: Neural Network weight plot for the model of vigor

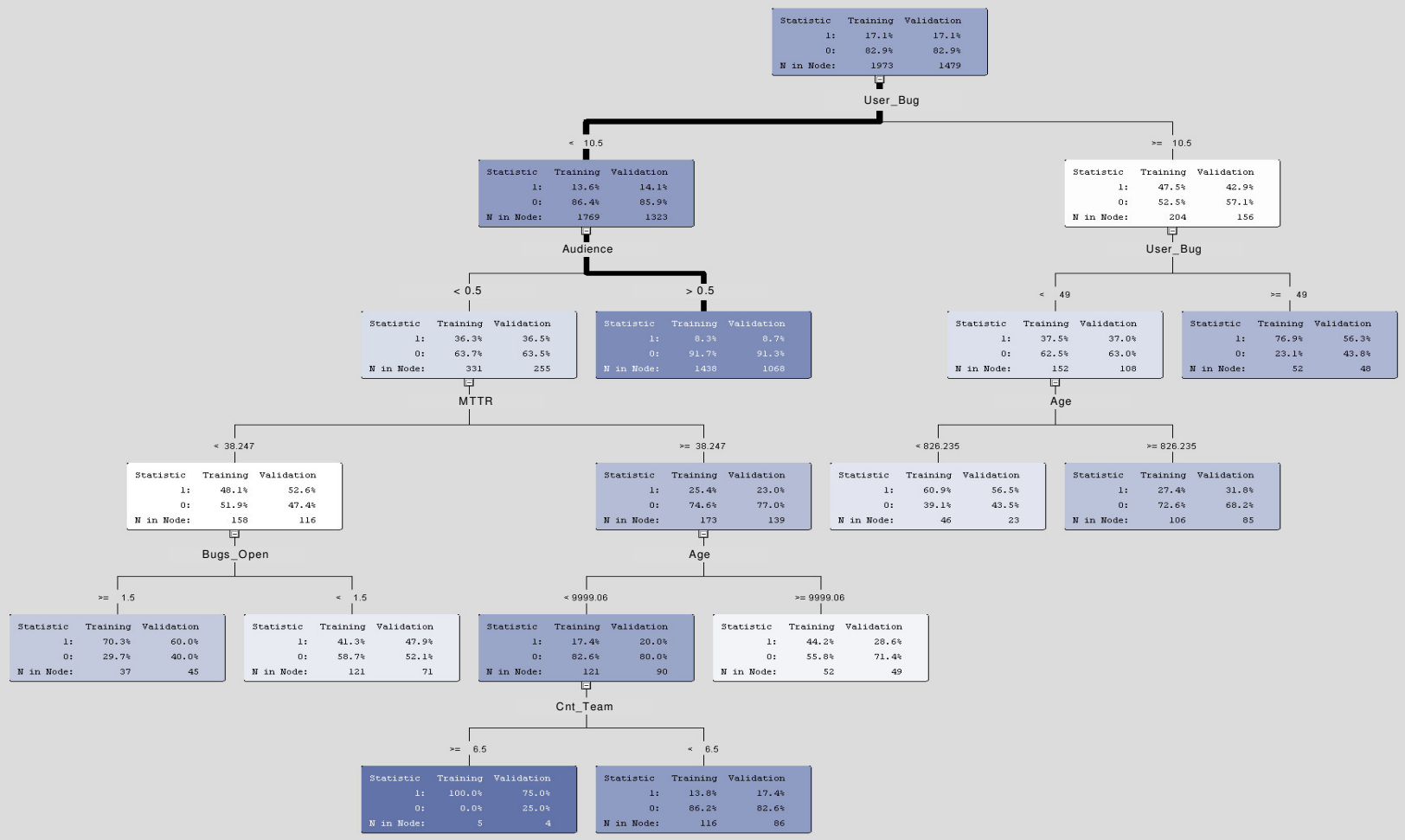

Figure 4.7: Decision Tree for the model of vigor

Further study of the Decision Tree showed that there was a persistent partitioning based on the type of audience and the bugs reported by end users. Thus interaction terms for these two variables was created for the Logistic Regression input. Additional interaction terms involving project type were also created. The Neural Network results identified a potential non-linearity in the variable age. To account for this, the variable age was binned into three categories (low, medium and high). Low age indicated that the project was started recently (from the date of analysis). High value of the variable age indicated that the project was old or was registered in the very early days of SourceForge (SF) project. The new variables and terms were used again for several Logistic Regression nodes. The best node was selected based on the values of Akaike Information Criteria (AIC) and Lift Charts. The ROC Charts for all the nodes used is shown in Figure 4.8.

The final model of the Logistic Regression provided the best fit of all three predictive modeling techniques and had the lowest AIC amongst the entire Logistic Regression models used in the analysis.

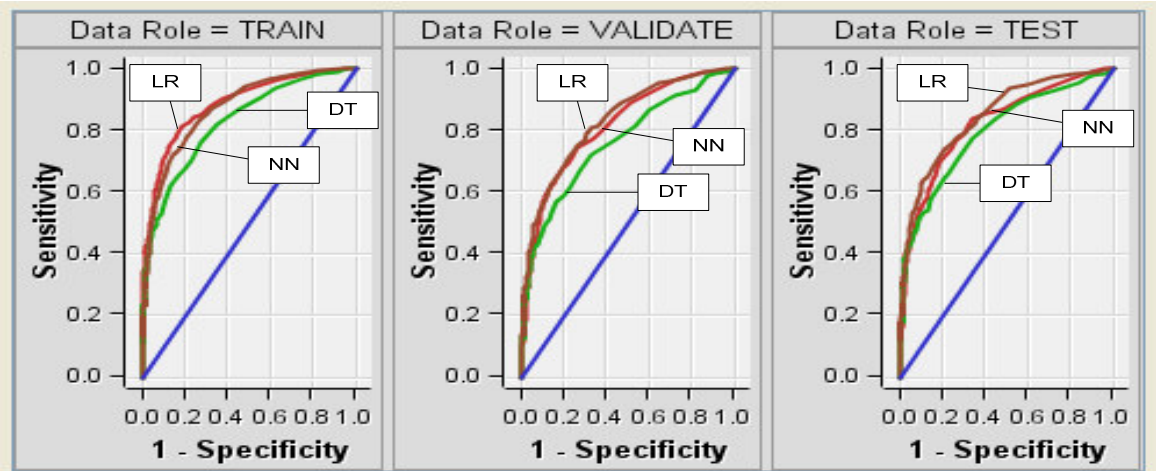

Figure 4.8: ROC Charts for model of vigor for DT, LR and NN techniques

4.3 SELECTED MODEL

The variables that were significant at 5% level were selected in the final model. The model was tested across the three data splits. The selected model statistics are for a sample of 1974 observations. The statistical profile of the variables selected for the final model is shown in Tables 4.2 – 4.5.

Table 4.2: Estimated correlation matrix of input variables for the LR model of vigor

rameter	Intercept	Age low	Age Med	Bug_Cnt	Bug_Open	Cnt_Mod	Msg_Cnt	Bug_User	Prj_Type _1	Prj_Type _2	Prj_Type _3
Intercept	1	-0.35886	-0.07084	0.00055	-0.05261	-0.25526	-0.07726	-0.05361	0.21212	-0.10423	-0.17011
Age low	0.35886	1	-0.21465	0.08916	-0.03854	0.00933	0.04344	0.02005	0.03375	-0.0337	-0.01294
Age Med	0.07084	-0.21465	1	0.01148	-0.01534	-0.00062	0.03425	0.04097	0.00716	0.0052	-0.01648
Bug_Cnt	0.00055	0.08916	-0.01148	1	-0.74164	-0.03046	-0.09245	-0.07079	0.04318	-0.0373	-0.0061
Bug_Open	0.05261	-0.03854	-0.01534	0.74164	1	-0.02408	-0.06935	-0.18253	-0.01935	0.02089	-0.03823
Cnt_Mod	0.25526	0.00933	-0.00062	0.03046	-0.02408	1	0.00021	-0.00206	0.01404	0.00792	0.01944
Msg_Cnt	0.07726	0.04344	0.03425	0.09245	-0.06935	0.00021	1	0.00995	0.03813	-0.02123	-0.06342
Bug_User	0.05361	0.02005	0.04097	0.07079	-0.18253	-0.00206	0.00995	1	0.01944	0.01669	0.02338
Prj_Type_1	0.21212	0.03375	0.00716	0.04318	-0.01935	0.01404	0.03813	0.01944	1	-0.26233	-0.25661
Prj_Type_2	0.10423	-0.0337	0.0052	-0.0373	0.02089	0.00792	-0.02123	0.01669	-0.26233	1	-0.08358
Prj_Type_3	0.17011	-0.01294	-0.01648	-0.0061	-0.03823	0.01944	-0.06342	0.02338	-0.25661	-0.08358	1
Prj_Type_4	0.05194	-0.0145	0.02384	0.01499	-0.01694	0.10756	0.05432	-0.0349	-0.26777	-0.14892	-0.13123
Prj_Type_5	0.14743	0.00027	-0.00957	0.00915	0.01618	-0.15214	-0.02566	-0.00532	-0.33476	-0.23613	-0.22258
Cnt_Team	0.43373	0.22057	0.00743	0.15462	0.01874	-0.10632	-0.02132	-0.00067	-0.04654	0.03384	0.06354
Downloads	0.07061	0.09335	0.01281	0.03217	-0.15087	-0.06174	-0.06941	-0.02163	-0.0567	0.03683	-0.00739
use_pm	0.27776	0.1444	-0.04091	-0.0572	0.03732	-0.02809	0.00416	0.00285	0.04751	0.01814	-0.15593
Prj_Type_1* use_mail	0.12992	0.0443	-0.01621	0.01477	-0.03352	0.121	0.05697	-0.01273	0.53131	-0.11653	-0.08482
Prj_Type_2* use_mail	0.01955	-0.02469	0.00655	0.00346	-0.00582	0.01161	-0.02756	0.0103	-0.13092	0.75242	-0.08134
Prj_Type_3* use_mail	0.04862	-0.02922	0.00956	0.06662	0.04754	-0.01939	-0.05191	0.01787	-0.10901	-0.07947	0.64573
Prj_Type_4* use_mail	0.01346	-0.03788	0.03043	0.01225	-0.0148	0.04025	0.03489	0.00471	-0.12647	-0.10682	-0.07512
Prj_Type_5* use_mail	0.20614	0.00357	-0.00079	0.00303	0.0261	-0.16382	-0.01763	-0.01168	-0.2338	-0.23445	-0.21058

Table 4.2: ***Continued***

Parameter	**Prj_Type_4**	**Prj_Type_5**	**Cnt_Team**	**Downloads**	**use_pm**	**Prj_Type_1* use_mail**	**Prj_Type_2* use_mail**	**Prj_Type_3* use_mail**	**Prj_Type_4* use_mail**	**Prj_Type_5* use_mail**
Intercept	-0.05194	0.14743	-0.43373	-0.07061	0.27776	-0.12992	0.01955	-0.04862	-0.01346	0.20614
Age low	-0.0145	0.00027	0.22057	0.09335	0.1444	0.0443	-0.02469	-0.02922	-0.03788	0.00357
Age Med	0.02384	-0.00957	0.00743	0.01281	-0.04091	-0.01621	0.00655	0.00956	0.03043	-0.00079
Bug_Cnt	0.01499	-0.00915	-0.15462	0.03217	-0.0572	0.01477	0.00346	-0.06662	0.01225	-0.00303
Bug_Open	-0.01694	0.01618	0.01874	-0.15087	0.03732	-0.03352	-0.00582	0.04754	-0.0148	0.0261
Cnt_Mod	0.10756	-0.15214	-0.10632	-0.06174	-0.02809	0.121	0.01161	-0.01939	0.04025	-0.16382
Msg_Cnt	0.05432	-0.02566	-0.02132	-0.06941	0.00416	0.05697	-0.02756	-0.05191	0.03489	-0.01763
Bug_User	-0.0349	-0.00532	-0.00067	-0.02163	0.00285	-0.01273	0.0103	0.01787	0.00471	-0.01168
Prj_Type_1	-0.26777	-0.33476	-0.04654	-0.0567	0.04751	0.53131	-0.13092	-0.10901	-0.12647	-0.2338
Prj_Type_2	-0.14892	-0.23613	0.03384	0.03683	0.01814	-0.11653	0.75242	-0.07947	-0.10682	-0.23445
Prj_Type_3	-0.13123	-0.22258	0.06354	-0.00739	-0.15593	-0.08482	-0.08134	0.64573	-0.07512	-0.21058
Prj_Type_4	1	-0.27994	0.01953	0.00886	-0.0236	-0.1041	-0.11036	-0.08206	0.63831	-0.24232
Prj_Type_5	-0.27994	1	-0.02126	0.01489	0.13208	-0.31379	-0.21558	-0.18368	-0.23307	0.85678
Cnt_Team	0.01953	-0.02126	1	-0.00657	0.06458	0.04654	0.0146	0.01246	-0.05131	-0.01115
Downloads	0.00886	0.01489	-0.00657	1	0.04803	0.03303	-0.00139	-0.02539	0.00775	-0.02098
use_pm0	-0.0236	0.13208	0.06458	0.04803	1	0.12505	-0.03346	-0.16471	-0.01412	0.10677
Prj_Type_1* use_mail	-0.1041	-0.31379	0.04654	0.03303	0.12505	1	-0.23003	-0.22839	-0.26723	-0.39625
Prj_Type_2* use_mail	-0.11036	-0.21558	0.0146	-0.00139	-0.03346	-0.23003	1	-0.1063	-0.15399	-0.22482
Prj_Type_3* use_mail	-0.08206	-0.18368	0.01246	-0.02539	-0.16471	-0.22839	-0.1063	1	-0.13748	-0.20446
Prj_Type_4* use_mail	0.63831	-0.23307	-0.05131	0.00775	-0.01412	-0.26723	-0.15399	-0.13748	1	-0.26923
Prj_Type_5* use_mail	-0.24232	0.85678	-0.01115	-0.02098	0.10677	-0.39625	-0.22482	-0.20446	-0.26923	1

Table 4.3: Analysis of maximum likelihood estimate of the LR coefficients

Parameter			**DF**	**Estimate**	**Std Error**	**Wald Chi-Sq**	**Pr > ChiSq**	**Std Est**	**Exp(Est)**
Intercept			1	-2.4081	0.1272	358.35	< 0.0001		0.09
Age	Low		1	1.1777	0.1052	125.44	< 0.0001		3.247
Age	Med		1	-0.3297	0.1101	8.97	0.0027		0.719
Bug_Count			1	20.8933	6.1339	11.6	0.0007	0.4168	999
Bug_Open			1	-6.5052	3.1689	4.21	0.0401	-0.1484	0.001
Cnt_mod			1	121.9	14.4809	70.81	< 0.0001	1.8517	999
Msg_Cnt			1	9.0409	3.7898	5.69	0.0171	0.1309	999
aud*Bug_User			1	0.0186	0.00753	6.09	0.0136	0.4976	1.019
Prj_Type	1		1	0.3165	0.3355	0.89	0.3455		1.372
Prj_Type	2		1	-0.3954	0.2284	3	0.0835		0.673
Prj_Type	3		1	0.1276	0.2132	0.36	0.5497		1.136
Prj_Type	4		1	0.5266	0.2593	4.12	0.0423		1.693
Prj_Type	5		1	-0.8799	0.3204	7.54	0.006		0.415
Cnt_Team			1	0.0573	0.0152	14.25	0.0002	0.1423	1.059
Downloads			1	1.16E-06	4.54E-07	6.52	0.0106	0.3694	1
use_pm			1	0.2784	0.0875	10.13	0.0015		1.321
Prj_Type*Use_mail	1	0	1	0.8497	0.3288	6.68	0.0098		2.339
Prj_Type*Use_mail	2	0	1	-0.3643	0.2257	2.61	0.1065		0.695
Prj_Type*Use_mail	3	0	1	0.00637	0.2101	0	0.9758		1.006
Prj_Type*Use_mail	4	0	1	-0.0297	0.2578	0.01	0.9082		0.971
Prj_Type*Use_mail	5	0	1	-0.6588	0.3243	4.13	0.0422		0.517

Table 4.4: Fit statistics of the train, test and validate samples of the LR model

Description	**Train**	**Validate**	**Test**
Akaike's Information Criterion	1335.931214	.	.
Average Squared Error	0.097372052	0.1081578	0.1015953
Average Error Function	0.330443795	0.4166475	0.3563485
Degrees of Freedom for Error	1957	.	.
Model Degrees of Freedom	16	.	.
Total Degrees of Freedom	1973	.	.
Divisor for ASE	3946	2958	2958
Error Function	1303.931214	1232.4434	1054.079
Final Prediction Error	0.098964236	.	.
Maximum Absolute Error	0.999995977	1	0.9999806
Mean Square Error	0.098168144	0.1081578	0.1015953
Sum of Frequencies	1973	1479	1479
Number of Estimate Weights	16	.	.
Root Average Sum of Squares	0.312044951	0.3288735	0.3187401
Root Final Prediction Error	0.314585817	.	.
Root Mean Squared Error	0.31331796	0.3288735	0.3187401
Schwarz's Bayesian Criterion	1425.328182	.	.
Sum of Squared Errors	384.2301154	319.93067	300.51876
Sum of Case Weights Times Freq	3946	2958	2958
Misclassification Rate	0.131272174	0.1372549	0.1331981

Table 4.5: Odds ratio estimates of the input variables of the LR model

Variable	Effects	Estimate
Age	Low vs. medium	7.582
Age	Medium vs. high	1.679
Bug_Cnt		999
Bug_Open		0.001
Cnt_Mod		999
Msg_Cnt		999
Aud*Bug_User		1.019
Cnt_Team		1.059
Downloads		1
use_pm	0 vs. 1	1.745

4.4 DATA INTEGRITY AND DIAGNOSTIC CHECKS

A variety of specification tests recommend for Logistic Regression models were performed on the final model (Hosmer and Lemeshow 2000; Menard 2002). The Pearson Residuals and Deviance Residuals were examined for case wise effect on the fit. No violations were detected (Hosmer and Lemeshow 2000; Menard 2002). The highest condition number of the model was 1.9, which is well within the recommended cutoff limit. The Variance Inflation Factor (VIF), of the independent variables were well below 5, suggesting that multicollinearity was not affecting the estimates (Belsley et al. 1980; Neter et al. 2004).

To test the fit of the final model, the first step is to ensure that the model contains all required variables, entered in the correct functional form. The next step is to evaluate the affectivity of the model, i.e. the goodness-of-fit. This is to ensure that knowing the values of all the independent variables in the model allows an accurate prediction of vigor, better than in case of no information in the independent variables. The next step is to evaluate how well the group of independent variables explains the vigor. In Logistic Regression models, the Log Likelihood (LL) criteria are used to select model parameters. The value of -2LL of the model with and without the independent variables was used to check the model fit. The fit of the model is determined by the reduction in the value of -2LL with and without the covariates. The results of this test are shown in Table 4.6. The results showed that the model is significant at 5% significance level ($p < 0.0001$).

Table 4.6: Likelihood ratio test for global null hypothesis: BETA=0

-2 Log Likelihood		**Likelihood Ratio Chi-Square**	**DF**	**Pr > ChiSq**
Intercept Only	**Intercept & Covariates**			
1803.960	1292.126	511.834	20	< 0.0001

The accuracy of a Logistic Regression model can be tested by the area under the ROC[24] curve. As a rule, the area under the curve indicates how well the model provides

[24]Receiver Operating Characteristics (ROC) is from the signal detection theory and it shows how the receiver operates the existence of signal in presence of noise. It plots the probability of detecting true signal (sensitivity) and the false signal (1-sensitivity) for an entire range of possible cutoff points.

discrimination between the high and low values of the target variable. The acceptable limits for the ROC curve are given in Table 4.7.

Table 4.7: Acceptable ranges of ROC value

IF ROC = 0.5	Model provides no discrimination
If 0.7 ≤ ROC < 0.8	Model provides acceptable discrimination
If 0.8 ≤ ROC < 0.9	Model provides excellent discrimination
If ROC ≥ 0.9	Model provides outstanding discrimination

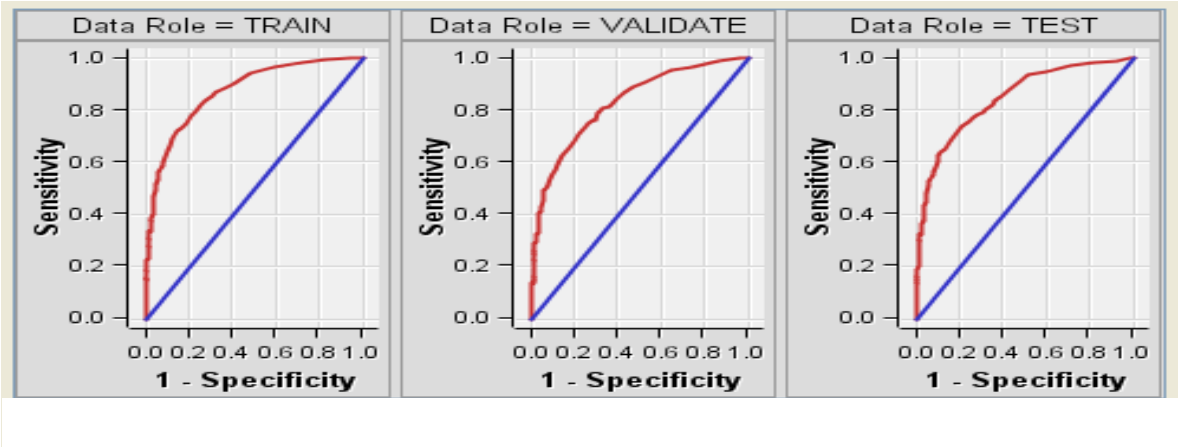

Figure 4.9: ROC for the final LR model of vigor

For the final model of vigor, the area under ROC curve was 0.834, which implies that the selected model for vigor provides excellent discrimination between the projects of high and low vigor. The ROC curve for the final model for the train, validate and test samples is shown in Figure 4.9.

Figures 4.10 – 4.12 show Cumulative Lifts, Lift Chart and percentage of the responses captured correctly. The results for all three samples of train, validate and tests are shown in the output. If the model had not been a good one, there would be significant differences between the training, validation and testing plots, indicating that the model is not robust. These charts showed that the all fits are acceptable for the final model selected and indicated a good fit that can be generalized over the population.

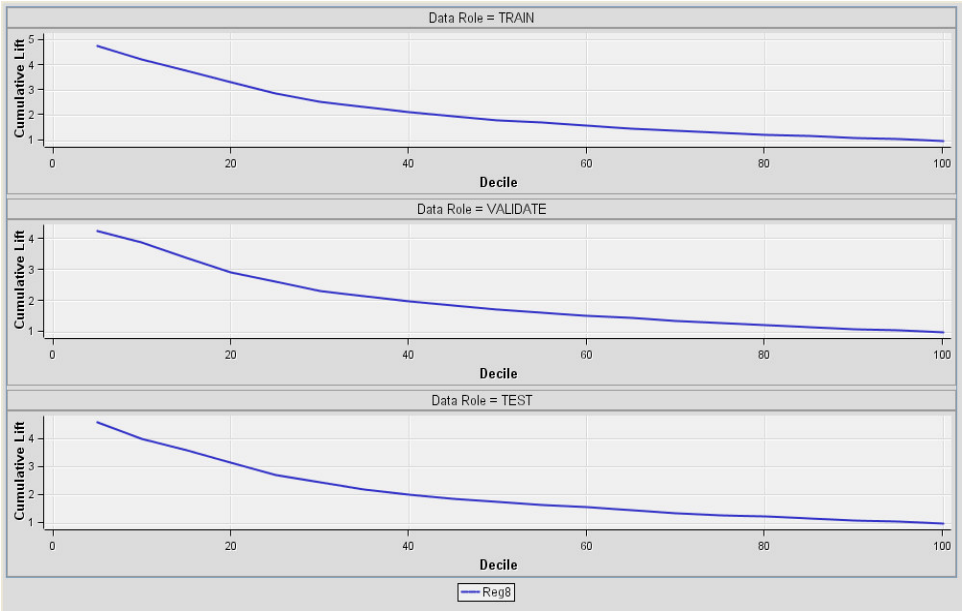

Figure 4.10: Cumulative lift for the final LR model, train, validate and test samples

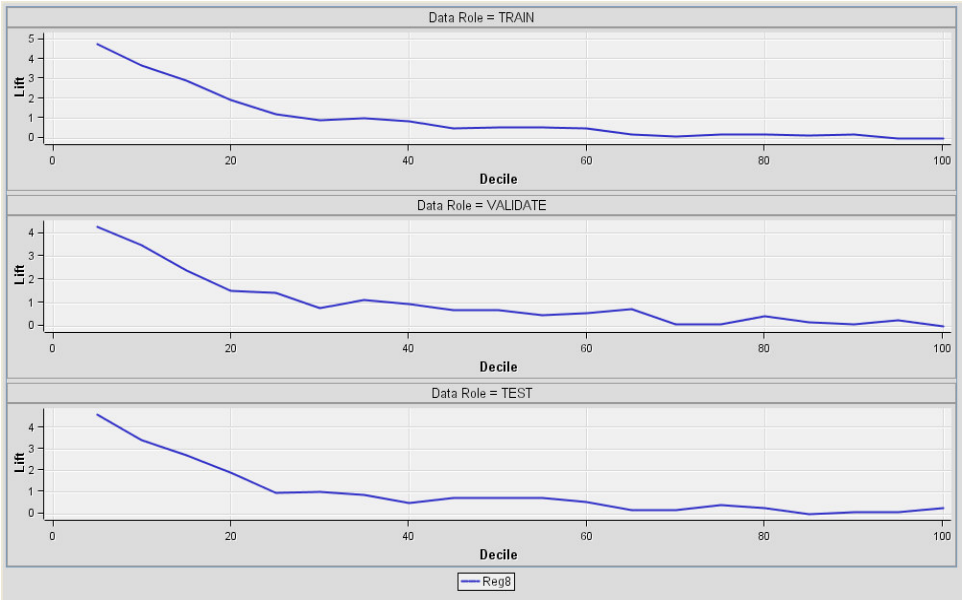

Figure 4.11: Lift of the final LR model, for train, validate and test samples

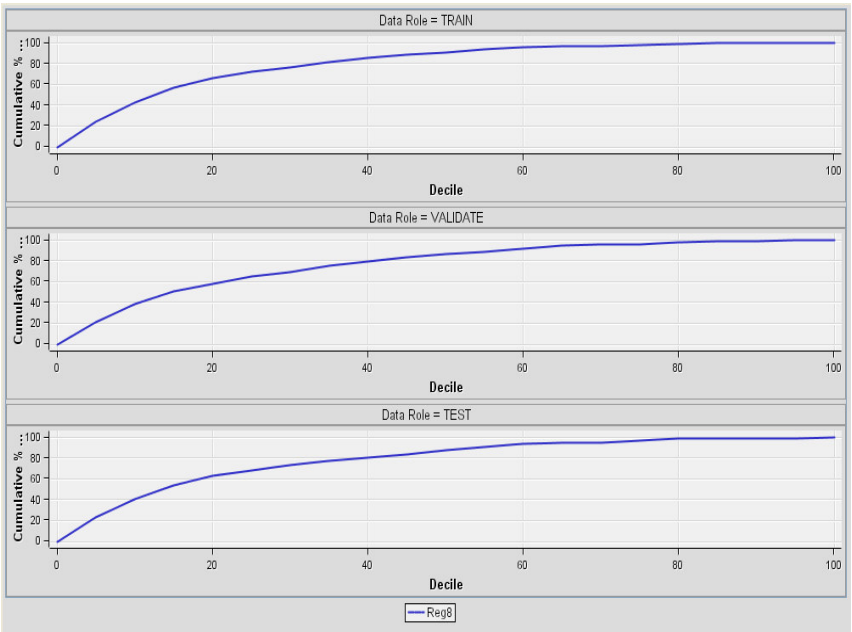

Figure 4.12: Percentage captured responses for LR model, for train, validate and test samples

Another way to summarize the results of a fitted Logistic Regression is through the classification table. The classification table is the result of cross classifying the outcome variable with a dichotomous variable whose values are derived from the estimated logistic probabilities. The classification table of the final model is shown in Table 4.8. The misclassification rate for the final model is less than 13%, which indicates a good fit.

Table 4.8: Classification table for the LR model

Target	Outcome	Target %	Outcome %	Count	Total %
0	0	87.5956	97.9829	1603	81.2468
1	0	12.4044	67.3591	227	11.5053
0	1	23.0769	2.0171	33	1.6726
1	1	76.9231	32.6409	110	5.5753

The train sample was used during model building. Validation was performed on the validate sample and the test sample was used for model testing. The formulation of the LR equation is shown below. It contains all the variables and the interactions that were significant in the final model.

$$logit(Vigor) = \mu + \beta_1(Age_{low}) + \beta_2(Age_{med}) + \beta_3(Bug\_Cnt) + \beta_4(Bug\_Open) + \beta_5(Cnt\_mod) + \beta_6(Msg\_Cnt) + \beta_7(Bug\_User) + \beta_8(Prj\_Type) + \beta_9(Cnt\_Team) + \beta_{10}(Downloads) + \beta_{11}(use\_pm) + \beta_{12}(Prj\_Type * use\_mail)$$

The final model was selected based on the best-fit statistics (see Table 4.4).

4.5 DISCUSSION

It is seen that the number of bugs (***Bug_Cnt***) in a project has a positive relationship to the probability of high Vigor. At first glance, this might appear counter intuitive. However, a deeper look into the OSS development process shows that the identification and reporting of bugs cannot be an indicator of the quality of Open Source Software. Mere reporting of bug indicates that the project is being used and the faults are being reported. In fact, the process of detecting and reporting bugs has been one of the processes of OSS that has been associated to its success (Raymond 2001). This claim however had never been empirically tested. Therefore, this research provides the empirical evidence that the process of software testing in OSS projects is improved by efficient discovery of bugs

Although the number of bugs reported has a positive influence on vigor, the number-of-bugs-open (***Bug_open***) has a negative influence. This indicates that whereas the reporting of the bugs is a contributor to vigor, the inability of the project to remove the bug leads to decline in vigor. Thus if in a project, bugs are being discovered, but not removed, the odds of project survival reduce. There can be several reasons for it, e.g., large number of bugs open can be because of the lack of enough effort available to remove them. High complexity of the code and low maintainability can also cause the number-of-bugs-open to increase. Therefore, number-of-bugs-open can be an indicator of the lack of quality

Many OSS projects use message boards as means of discussion and communication. The count of the messages (***Msg_Cnt***) on project boards has a positive impact on vigor.

Therefore, projects with high level of communication between developers and users show higher vigor.

The Count of the new modules is an indicator of the increase in functionality of a project. As a project develops and new functionality is added the number of new modules increases. Though the sample of development projects only contains projects that had been in OSS community of over a year, yet the increase in functionality can vary with time. Therefore, the functionality of the projects was normalized for time. The functionality of the project (***Cnt_Mod***) has a positive impact on vigor. Thus, a project that offers more functionality over time has a greater vigor.

Project downloads (***Downloads***) have a positive impact on vigor. Though downloads alone cannot be a measure of the project success or popularity, yet a high number of downloads indicates that more people are interested in the project. To account for different registration dates for the projects, downloads per unit time were used in the analysis.

The final model indicated that a large team size (***Cnt_Team***) has a positive impact on vigor. This finding is very interesting in terms of OSS specific research. In OSS teams work online and there is very little face-to-face working within the teams. CSS research indicates that large team sizes sometimes can have a negative effect on the project performance. Yet in OSS, the analysis indicates that larger teams can improve the vigor of a project. The reasons can be that larger teams have more resources available for managing the projects and some of the large team issues encountered in CSS are not applicable in OSS. In OSS,

the participation to a project is voluntary. Therefore, the creation of teams takes place over time, depending upon the interest and expertise of the programmers. Therefore, once a team is formed, typical issues of conflict management and task distribution in CSS are not directly valid in the OSS domain.

The number of bugs reported by the end users (***Bug_User***) of the project had a positive impact on project vigor. The end user involvement in the development and testing process is a unique characteristic of OSS projects. This is indicative of the end user involvement in the project and the usage of that software. Downloads alone do not lead to the use of a project. Since a download is free, there is no way to measure if the user ever uses the project. However, bugs reported by the end user definitely indicate that the project is being actively used. This also indicates that the user community is involved in the process and is responsible enough to report the bugs to the project team. In fact, the role of end user has been much glamorized in the OSS literature.

The bug reported by end user alone was not significant in the initial runs. However, the decision tree results revealed a potential interaction effect with the intended audience (***Aud***). As mentioned earlier, the intended audience of the OSS projects was divided into two main categories; Projects developed for other programmers (e.g. plug-ins, reusable modules) and the projects developed for non-programmers or complete applications. When the interaction term for audience type and the bugs reported by end users was added (***Aud*Bug_User***), the variable was not only significant, but it also improved the performance of the LR node.

In this research, the variable "project type" (***Prj_Type***) was developed using text mining of the project description data. SourceForge (SF) provides several categorizations of the projects based on industry, application, etc. This categorization is selected at the time of registration and one project can be placed into several categories. The existing categorization of SF was not useful for model building. Every project had to be placed in only one category and there had to be a uniform criterion for this purpose. Project description data contained significant details, which can be used to identify a category. Therefore, text analysis was performed to extract useful information regarding the nature of the project. This information was used to place each project in a cluster indicative of the project type.

Variable (***Prj_Type***) had significant impact on the project vigor. This implies that some types of projects might be better suited for development compared to others. In the final model, it can be seen that projects that were in Cluster #5 have significant negative impact on project vigor. From the analysis of keywords, it appears that these projects are typically JAVA Applications and Tools. Newer projects can be categorized into the exiting clusters by scoring them based on the project description. Therefore, the effects of the project type on the project vigor can be analyzed.

Age (***Age***)of the project was used as a control variable. Software engineering literature has used this to account for different ages of the projects or systems under investigation. In this research, preliminary results indicated that there was a pattern in the age of the project and its relationship to vigor. Binning the variable Age (computed from the

registration date of a project), it was found that there were three significant bins for the age of the projects. The projects were categorized based on their age. Recent projects had a higher probability of high vigor compared to older projects. This can be explained in two ways. The first explanation is based on literature in software evolution. Software evolution laws assert that as a software system ages, it declines in performance (Lehman and Ramil 2001; Lehman and Ramil 2002). Godfrey (2001) tested these laws in the OSS domain using Linux data. The scope of the current research does not encompass the complex questions about laws of software evolution, but variation in the patterns of vigor, based on the age, is indicative of an interesting future research question. Another explanation, which is more intuitive, is that as the OSS movement has matured so has the processes and tools of project development and management. Therefore, projects that are more recent, use advanced resources and therefore have a higher chance of surviving. Given two projects of same performance and functionality, it might be worthwhile to use the period of evolution of the project to predict its future performance.

4.6 CONCLUSIONS

The model developed and presented in this chapter adds to the body of knowledge in OSS project development. It is the first ever, empirical investigation into the affects and significance of end user involvement in OSS projects. This model not only provides a quantitative tool to compare the performance of various OSS projects, but it also identifies the factors that contribute to vigor in OSS projects. The identification of the factors is very

important to the practitioner and research community. The development teams can better monitor the performance of their projects and adjust the input variables to achieve the desired outcome. For businesses that wish to adopt OSS projects, this model provides them the ability to study the potential projects and make better decisions regarding adoption. For research community the model provides a deeper understanding of the phenomenon of OSS development and a better model to predict software project outcomes. It also identifies some new factors that have not been used in prior research and with further work can be used in developing a more generalized model for software projects. This study also highlighted the use of multiple predictive modeling techniques in effective modeling. Tables 4.9-4.13 indicate the variables that were significant in the final model.

Table 4.9: Process related variable measurement and sources

Variable	**Measure**	**Symbol**	**In final Model**
Project Management	Use PM (Y/N)	*Use_PM*	☑
Process Quality	Mean Time to fix a bug (MTTF)	*MTTF*	
Communication Channel	Forum use (Y/N)	*Use_forum*	
	Number of forums	*Cnt_forum*	
	Use Mail (Y/N)	*Use_mail*	☑
	Use News Groups (Y/N)	*Use_news*	
Req. Implementation	Time to implement a feature	*TTFT*	
Configuration Management	Use CVS (Y/N)	*Use_CVS*	
	Count CVS commits	*Cnt_CVS*	
Process Quality	Count of bugs	*Bug_Cnt*	☑
	Bugs not fixed	*Bug_Open*	☑

Table 4.10: Product related variable measurement and sources

Variable	**Measure**	**Symbol**	**In final Model**
Functionality	Increase in features	Cnt_feat	☑
	New Modules	Cnt_file	☑
Maintainability	Number of distinct members reporting the bugs	Cnt_mem	
	Number of distinct members fixing the bugs	Cnt_Usr_fix	
Portability	Number of platforms supported	Cnt_OSI	
	Number of programming languages supported	Cnt_Prg_lang	
License Type	License type	Lisc	
Project Type	Primary categorization of the project.	Pjr_type	☑
Usefulness	Downloads	Downloads	☑
	Page Views	Pg_View	
Product Compatibility	Number of translations	Cnt_tran	
	Number of platforms supported	Cnt_plat	
Usage	Usage by end user	Bug_User	☑

Table 4.11: Control variable measurement and sources

Control Varisweables	**Measure**	**Symbol**	**In final Model**
Size	Source Lines of Code	*Size*	
Age	Time elapsed since the start of project	*Age*	☑

Table 4.12: Resource related variable measurement and sources

Variable	**Measure**	**Symbol**	**In final Model**
Effort	Number of registered developers for the project	*Cnt_Team*	
Team Communication	Messages posted at development forums	*Forum_post*	☑

Table 4.13: User related variable measurement and sources

Variable	**Measure**	**Symbol**	**In final Model**
User Type	Audience Programmer (Y/N)	*AUD (o= prog, 1 = non-prog)*	☑
Activity Level of User	Forum posts by users	*Cnt_Msg*	☑
	Number of distinct individuals posting messages, bugs or feature requests	*User_Int*	
Community Size	Number of distinct senders of messages	*Cnt_User*	

CHAPTER V

MODEL FOR RESILIENCE

A model for the performance evaluation of OSS projects in maintenance is formulated and discussed in this chapter. The maintenance performance is measured as the resilience of a project. Data from SourceForge warehouse is used to develop the model using various predictive modeling techniques of Data Mining. Later the model is expanded into a two-stage model to explain the outcomes of OSS projects in the maintenance phase. This model identifies OSS maintenance phenomenon factors and their relationships. The use of the two-stage model explains how various outcomes affect each other and what factors can be controlled to improve the overall performance of a project. The results of the final selected model are discussed in details.

5.1 BACKGROUND

The maintenance phase of a software project refers to the lifecycle activities carried out once a project has been operational. Software maintenance tasks include removing and correcting errors, adding new functionality and making enhancements to improve its performance. Earlier studies have suggested that maintenance accounts for 60-90% of the software lifecycle (Bennett and Rajlich 2000; Erlikh 2000; Zelkowitz et al. 1979). Software maintenance activities consume a large portions of IT budgets and manpower (Eastwood 1993; Lientz and Swanson 1980). A substantial portion of Information Systems and

Software Engineering literature is devoted to the study of software maintenance. Considering the high cost and manpower requirements of software maintenance, there is significant research on software maintenance in Closed Source Software (CSS) projects.

Generally, CSS vendors do not allow the users to view source code or to detect locate or remove bugs. The vendor controls all source code level maintenance activity. On the other hand, Open Source Software (OSS) users have the ability to view, upgrade and modify source code. OSS users can detect bugs and report them to the project team, submit feature and support requests. Once a bug is detected, the information is provided on the project web page and users can recommend solutions. This creates a large maintenance community, which involves the end user in the bug detection and removal process (Feller and Fitzgerald 2002; O'Reilly 1999; Raymond 2001). The maintenance process in OSS projects however is not mandated as it is in the case of CSS. It is critical that the maintenance performance of an OSS project is predicable, especially from the standpoint of corporate customers, which may see the cost of software maintenance to override any advantage of free availability of the code.

The two major causes of project failure during the maintenance phase are delays in bug removal and the inability to incorporate user requests (Pinto and Samuel. J. Mantel 1990; Pinto and Slevin 1987; Seddon et al. 1999). In this research, this aspect of project performance is captured through the dimension of Resilience. In this research, measuring a project's resilience to error detection and removal was used to create a model of OSS project maintenance phase performance.

5.2 MODEL BUILDING

This research used Data Mining techniques for model formulation, validation and testing. To identify the factors that affect the maintenance phase performance of OSS projects, SourceForge (SF) dataset was used. The variables identified earlier in Tables 3.1-3.5 were extracted for the SF projects in their development phase.

The initial dataset was examined for dummy projects and missing values. As in the case of the development phase model, the projects in maintenance phase exhibited similar characteristic for idle projects. Such projects had never been operationalized and could not have been used for the analysis. Using the artifact repository, which contains data on all submitted artifacts, the valid projects were separated out of the entire dataset. As mentioned in Chapter IV, the projects that had been active on SF for at least a year (since May 2005) were used in the analysis. The dataset was validated against an independent data extraction performed by another researcher. This ensured the integrity of the data collected.

SF maintains a detailed bug repository for all the bugs reported to the registered projects. Information on time taken to fix a bug was extracted using SQL queries. The Mean Time To Repair (MTTR) was computed according to the same procedure as discussed in Chapter III. A small MTTR indicates high resilience and a high maintenance performance.

The clean dataset was imported to SAS EM 5.2. This dataset had ***4965*** observations. The data was first examined using the insight node. The distributions of the values of MTTR

were examined and a cutoff value of MTTR was selected based on the distribution of the data. The data indicated that there was a cutoff at MTTR of one day. Therefore projects with MTTR less than one day have a low MTTR, i.e. high resilience and vice versa. The distributions of the important variables are available in the Appendix B.

The transformation node was then used to make needed transformations and to create new variables. Variables were normalized with respect to time, if needed, new variables were computed (e.g. defect density = SLOC / Bugs) and the target variable was created. The cutoff value as suggested from the analysis using insight node was used to create the target variable. Projects with MTTR higher than the cutoff were labeled as “High” and the MTTR was coded as 1. The projects with MTTR lower than the cutoff were labeled as “Low” and coded as 0. Note that High and Low values are for MTTR, which is the inverse of resilience, indicating that a project with a high MTTR has a low resilience. The distribution of MTTR is shown in Figure 5.1. It can be seen that only 9.55% of the projects had a low MTTR i.e. high resilience. The rest of the projects with high MTTR were categorized as low resilience projects.

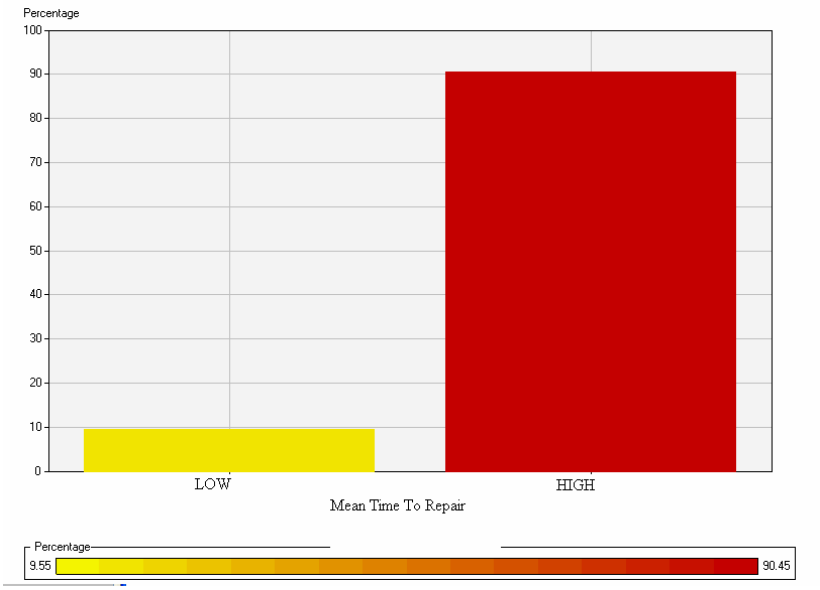

Figure 5.1: Distribution of mean time to repair (MTTR)

Next text mining of the project description data for projects in the maintenance phase was performed. There was a reason for using this categorization separately and not with the development phase project. While considering the SF dataset it was seen that the majority of the projects in the maintenance phase were launched as production projects from the start and were never in the OSS community for development purpose. Therefore in order to preserve any possible differences that may exist in the patterns of project types in the development and maintenance phases, two different text analyses were performed.

The project description textual data was used to create a new variable called “project type”. The dataset of project description for each of the project in its development phase was used in SAS Text Miner to cerate the new variable. Initially the default stop list was used on

the dataset. The stop list contains the words that are ignored while the text analysis is performed. The default list contains most common occurring words that do not carry information regarding the text being analyzed. The initial run with the default stop list provided with a word frequency table. For such a large amount of data, performing an initial run with default stop list is beneficial. A new start list was created by removing the words that were not considered a project description or added no usefulness to the analysis e.g. frequent words like where, upon or abbreviation like en, dl etc. The new list with "keep terms" is saved as a new start list and a final analysis is performed based on this list.

The Text Miner node was set to cluster the terms automatically. The option to generate the Singular Value Decomposition (SVD) terms and perform clustering based on the SVD dimension was selected. A maximum number of 40 clusters were allowed. The term stemming option was set to "Yes". The frequency weighing method was "Log" and the term weighting method was "Entropy". The expectation maximization algorithm was used for clustering. This algorithm is best suited in cases where the expected number of categories is unknown.

The observations were classified into seven clusters. The resulting clusters and the descriptive terms along with percentages and frequencies are shown in Table 5.1.

Table 5.1: Description terms, frequency and percentage of each cluster for project type

Cluster	Percentage	Freq	Descriptive Terms
1	0.150252	746	+ library, c++, + class, python, + support
2	0.035851	178	+ game, + player, + play, game, + base
3	0.091037	452	+ file, into, + will, + program, + image
4	0.082578	410	php, + easy, mysql, web, + database
5	0.090634	450	+ framework, development, + application, java, web
6	0.338771	1682	+ server, + client, + allow, + have, + tool
7	0.210876	1047	+ code, + source, + project, java, + base

Unlike the development data, the maintenance data split into seven clusters. Cluster #1 contains terms associated to programming languages e.g. C++ and python. Cluster #2 is associated to games, Cluster #3 contains terms related to image programs, Cluster #4 has terms related to databases, Cluster #5 had terms related to JAVA and web applications, Cluster #6 has terms related to networking while Cluster #7 has terms related to general OSS projects. The segment profile of the clusters is shown in Figure 5.2.

Once the text mining results were merged with the original dataset, the new data was ready to be used for model building. The process flow diagram for the model building process is shown in Figure 5.3.

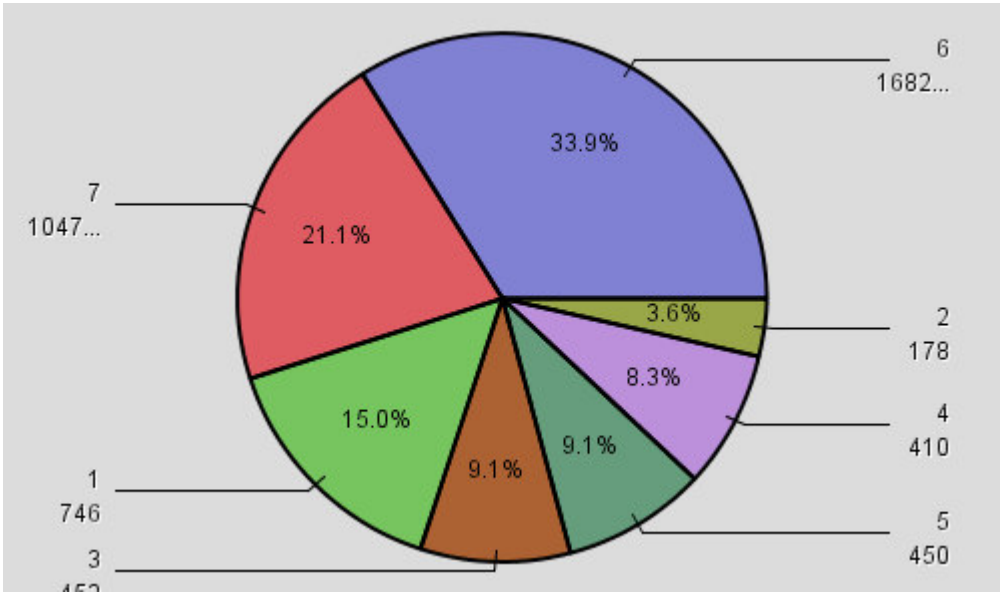

Figure 5.2: Segment profile of clusters for project type

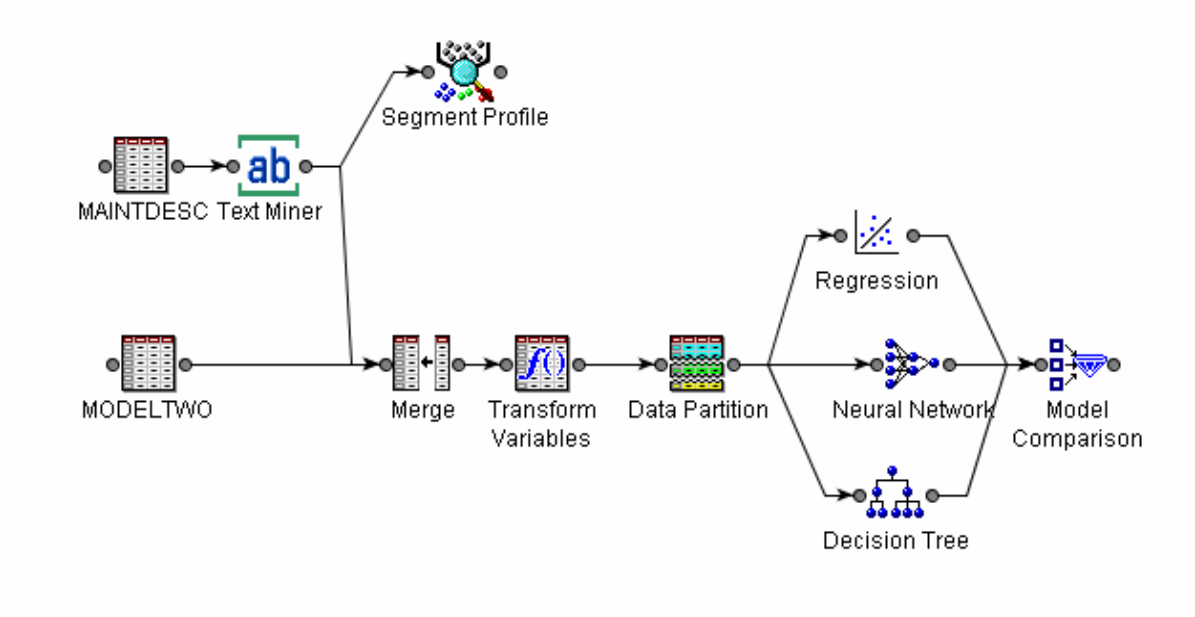

Figure 5.3: Process flow diagram for the analysis of resilience

The Sampling node was used to create the data splits. The data set was split into 40% training, 30% validation and 30% test sample. The details of why and how the sampling was performed have been discussed in Chapter III. Since the percentage of the observations with high resilience was very small, stratified random sampling was performed to ensure that each spilt is an accurate representation of the actual population. Thus, the sample used in model building is an accurate representative sample of the actual population.

Logistics Regression, Neural Networks and Decision Trees were all used to build the model. The stepwise method for variable selection was used in the Logistic Regression analysis. Stepwise Logistics Regression is recommended method of variable selection for exploratory research (Hosmer and Lemeshow 2000). It was found that Decision Trees was not a suitable technique for the given data. Several combinations of the number of terms in each leaf and splitting algorithms were used, but none gave reliable results. Therefore, the use of Decision Trees for the analysis was abandoned. The further comparisons and model building was performed using the Logistics Regression and Neural Networks. The outputs of the initial runs are shown in Figures 5.4 and 5.5.

Based on the AIC, Lift and ROC Curve values, the Logistics Regression model was selected as the final model. The model was built using stepwise regression and possible interaction affects were examined.

The model was evaluated performing diagnostic testing and Logistics Regression evaluation criteria.

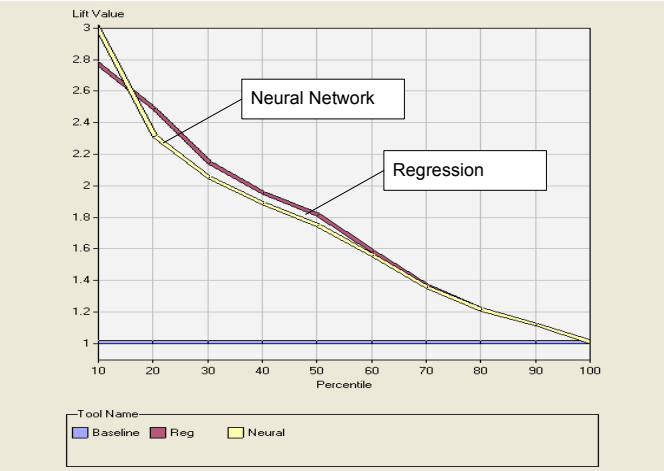

Figure 5.4: Lift values for LR and NN nodes for resilience

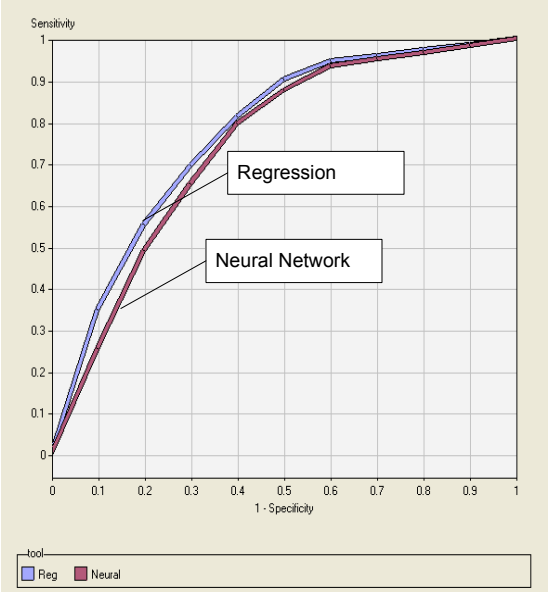

Figure 5.5: ROC curves for LR and NN nodes for resilience

5.3 DATA INTEGRITY AND DIAGNOSTIC CHECKS

A variety of specification tests recommended for Logistic Regression models were performed on the final model (Hosmer and Lemeshow 2000; Menard 2002). The Pearson Residuals and Deviance Residuals were examined and no violations were detected (Hosmer and Lemeshow 2000; Menard 2002). The highest condition number of the model was 1.56, which is well within the recommended cutoff limit. The Variance Inflation Factor (VIF), of the independent variables were well below 5, suggesting that multicollinearity was not affecting the estimates (Belsley et al. 1980; Neter et al. 2004).

To test the fit of the final model, the first step is to ensure that the model contains all required variables, entered in the correct functional form. The next step is to evaluate the affectivity of the model, i.e. the goodness-of-fit. This is to ensure that knowing the values of all the independent variables in the model allows an accurate prediction of resilience, better than the case of no information in the independent variables. The next step is to evaluate how well the group of independent variables explains the resilience. In Logistic Regression models, the Log Likelihood (LL) criteria are used to select model parameters. The values of -2LL of the model with and without the independent variables were used to check the model fit. The fit of the model is determined by the reduction in the value of -2LL with and without the covariates. The results of this test are shown in Table 5.2. The results showed that the model is significant at 5% significance level ($p < 0.0001$).

Table 5.2: Likelihood ratio test for global null hypothesis: BETA=0

-2 Log Likelihood		**Likelihood Ratio Chi-Square**	**DF**	**Pr> ChiSq**
Intercept Only	**Intercept & Covariates**			
1168.711	972.835	195.8758	4	< 0.0001

The accuracy of a Logistic Regression model can be tested by the area under the ROC curve. As a rule, area under the curve indicates how well the model provides discrimination between the high and low values of the target variable. The acceptable limits for the ROC curve are given in Table 4.7.

For the final model of vigor the area under ROC curve was almost 0.8, which implies that the selected model for resilience provides excellent discrimination between the projects of high and low vigor. The ROC curve for the model is shown in Figure 5.6.

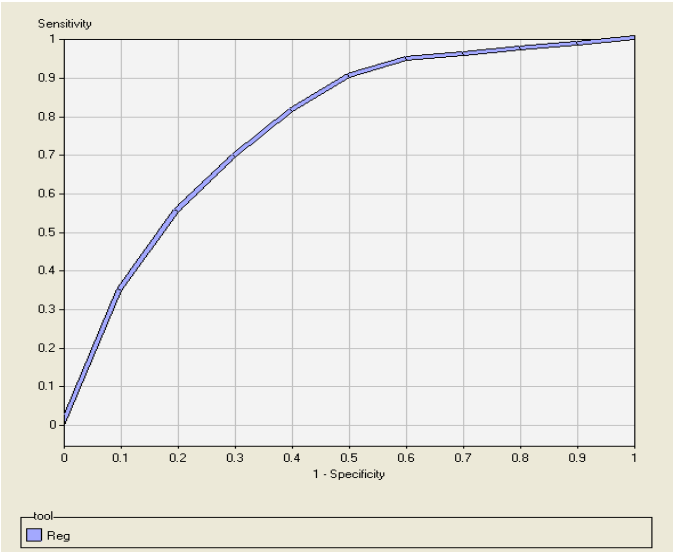

Figure 5.6: ROC curve for the final LR model of resilience

The final form of the model can be written as :

$$logit\ (R = 0) = \mu + \beta_1 (Age) + \beta_2 (Downloads\ \ ) + \beta_3 (Bug\ \_User\ ) + \beta_4 (Use\ \_mail\ )$$

Where $R = 0$ indicates low resilience.

Since target = 0 means a low time to fix an error, i.e. high resilience, therefore the intercept values are for odds of low resilience or resilience = 0. The estimates of the intercepts are shown in Table 5.3.

Table 5.3: Analysis of maximum likelihood estimates of input variables of the LR model

Parameter	DF	Estimate	Standard Error	Wald Chi-Square	Pr > ChiSq	Standardized Estimate	Exp(Est)
Intercept	1	-7.4232	2.7707	7.18	0.0074		0.001
Downloads	1	-0.0986	0.0273	13.01	0.0003	-0.348	0.906
use_mail	1	0.2498	0.0953	6.87	0.0088		1.284
Bug_User	1	-0.5531	0.0814	46.21	< 0.0001	-1.8399	0.575
Age	1	0.000482	0.000175	7.58	0.0059	0.1305	1

The bugs reported by users have a positive impact on the resilience. This means that projects with higher end user activity in reporting bugs have a greater probability of having low MTTR and high resilience. Downloads had a similar affect on the resilience of the project.

The age of the project has a negative impact on the resilience. Either this affect is indicative of the evolution of the better maintenance practices in newer projects or the decline of maintainability of a project as it ages. According to the laws of software evolution as a software system ages, it becomes more complex and harder to maintain. The increased complexity can increase the difficulty in removing bugs because of complex interfaces and spaghetti code. The affect of age is being used as a control variable in this study. Later studies can be done with focus at the affect of age on the resilience over the entire lifecycle of single project.

The model also indicates that the use of mail had a negative affect on the resilience. At first glance, this result appears counter intuitive because use of mail messages should not adversely affect the maintenance process. One of the possible explanations can be that if a project offers mail messaging, then some users and developers might end up using the mail messaging system to report bugs instead of the bug repository. This could cause potential delays in the correct reporting of bugs and therefore delay the process of bug removal. This factor alone can be studied in detail, to discover the cause of this negative affect. The fit statistics for the final model is presented in Table 5.4. The misclassification rate for this model is at an acceptable 9%.

Although the selected model provides an acceptable predictive power of project resilience, yet the bugs reported by users and downloads alone provide no useful tools to the project managers or end users to evaluate and control the project. This is because both the variables are outcomes themselves rather than being controllable events. Therefore, it was decided that further models would be created to explain the bugs reported by end users and the number of downloads. This approach was very useful in the light of a theory proposed by Delone and McLean on Information System success, which relates the success of an Information System to the system's usefulness and usage.

The number of the bugs reported by the end users indicates that the product is being used and understood by the end user. This also indicates that end user is spending time and effort in locating and reporting the errors. According to OSS literature, the reporting of

a bug is an indication of an involved end user rather than the low quality of software (supported by the first model also).

Table 5.4: Fit statistics for the LR model

Fit Statistic	Training	Validation	Test
Akaike's Information Criterion	1062.242	.	.
Average Squared Error	0.078264	0.080251	0.075512
Average Error Function	0.265048	0.277648	0.24949
Degrees of Freedom for Error	1980	.	.
Model Degrees of Freedom	5	.	.
Total Degrees of Freedom	1985	.	.
Divisor for ASE	3970	2978	2982
Error Function	1052.242	826.8363	743.9803
Final Prediction Error	0.078659	.	.
Maximum Absolute Error	0.999662	0.99916	0.96311
Mean Square Error	0.078462	0.080251	0.075512
Sum of Frequencies	1985	1489	1491
Number of Estimate Weights	5	.	.
Root Average Sum of Squares	0.279757	0.283287	0.274795
Root Final Prediction Error	0.280463	.	.
Root Mean Squared Error	0.28011	0.283287	0.274795
Schwarz's Bayesian Criterion	1090.209	.	.
Sum of Squared Errors	310.7083	238.9889	225.1777
Sum of Case Weights Times Freq	3970	2978	2982
Misclassification Rate	0.097733	0.097381	0.09725
Total Profit for AVER_05O	194	145	145
Average Profit for AVER_05O	0.097733	0.097381	0.09725

The number of downloads would be high if the OSS user community finds need for a certain software. Therefore a high number of downloads would imply that the project is considered useful by the potential user and that is the reason it is being downloaded more often. However, not every download results in usage. In fact the use of downloads alone as a measure of project performance can be very misleading. The usage and the usefulness both have a positive impact on the resilience i.e. a project that is more useful and is used more often has a high resilience (or low MTTR).

This research was conducted to identify factors that OSS project teams can manage to improve the performance of their projects. However, the resulting factors in the model of resilience cannot be controlled directly controlled. Therefore, two additional models were developed to explain the usefulness and usage of OSS projects in maintenance phase. Each of these models is discussed in detail below.

5.3.1 Model of OSS Project Usage

The role of the end user is very important in the development and maintenance of OSS projects. As mentioned earlier, the bug reporting by the end user has a positive impact on the resilience of OSS projects. The number of bugs reported by the user is indicative of the usage of the project since bug detection is related to its use.

It is not possible to detect all the problems in the software during its development phase. As the software becomes operational, its users encounter bugs and errors. The occurrence of bugs does not necessarily imply low quality. A project with low number of detected bugs can have low quality because bugs may not have been discovered due to very low usage. The bugs reported by the end users also indicate the involvement of the user in its maintenance and effectiveness of a project's bug reporting process.

All the variables identified in section 3.2.3, were used in this analysis. An additional variable was created called "defect density". Defect density was created using the total number of bugs divided by the size of the project. Defect density has been typically used as a measure for the quality of a project. In operational software system, project quality can play a vital role in the usage. If the quality is low, users will abandon the use of the project.

Three Data Mining techniques were used for the analysis; Decision Tree, Neural Network and Linear regression. Regression analysis was considered suitable since the target variable i.e. bugs reported by users, was not a binary variable. The process flow diagram for the model is shown in figure 5.7.

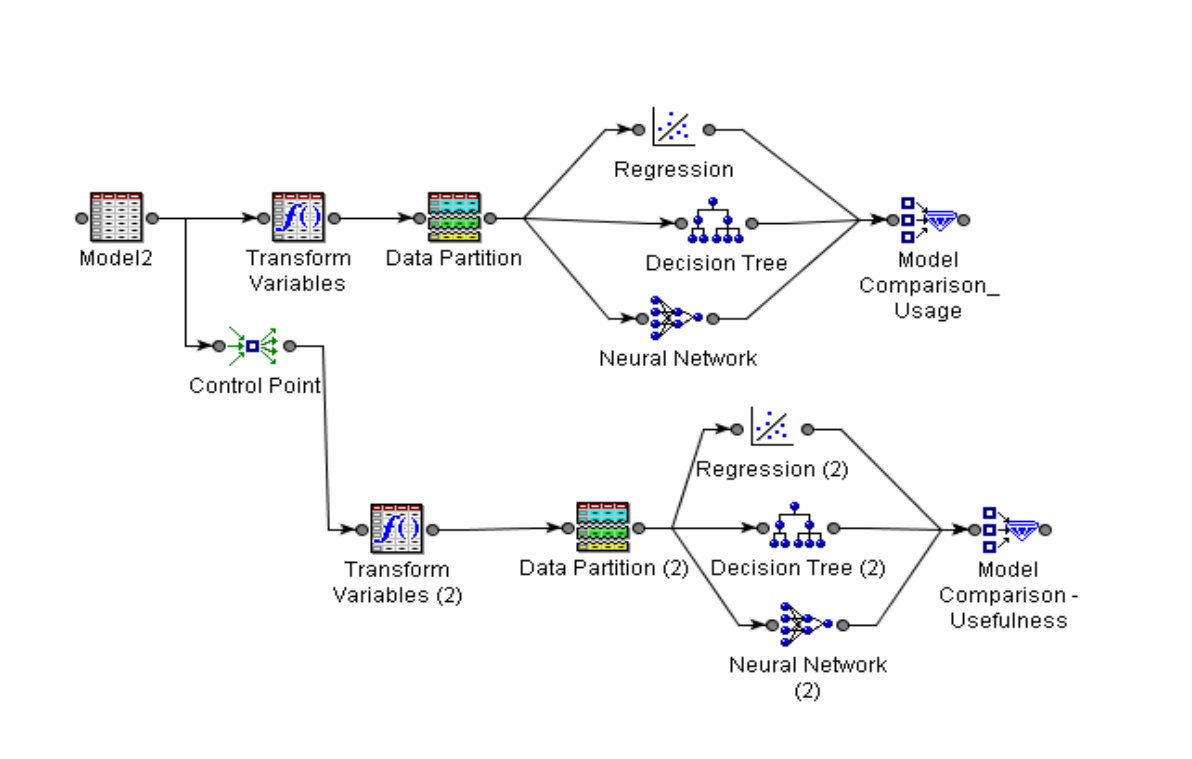

Figure 5.7: The process flow for the models of usage and usefulness

The Bugs reported by end user (***Bug_User***), was selected as the target variable. The data was split into three datasets; 40% train, 30% validate and 30% testing. For the Linear Regression analysis, Stepwise method was used for variable selection. The model was allowed to use possible interactions between the independent variables. The Linear Regression model was selected as the final model, based on the lowest value of AIC and the means of the predicted values. The score ranking overlay plot for the model is shown in figure 5.8. The details of the Decision Tree and Neural Network are available in the Appendix C.

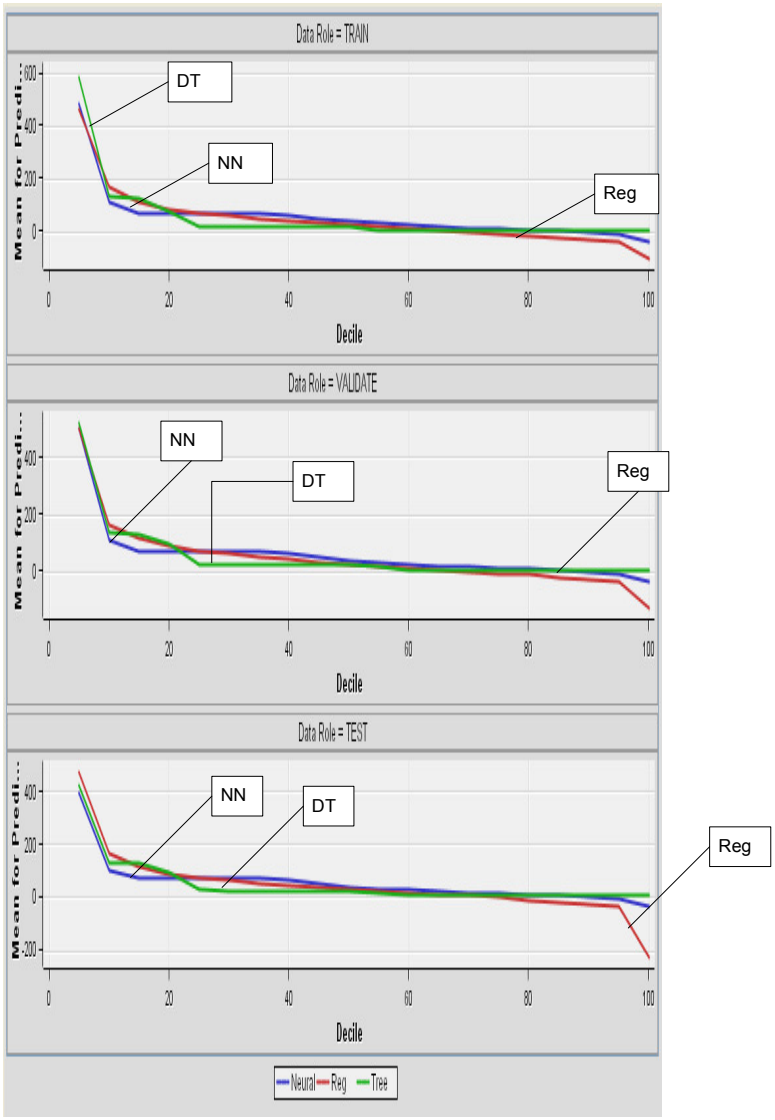

Figure 5.8: The score ranking overlay for the analysis of usage

The Linear Regression model that was selected as the final model was significant at the 5% level ($p < 0.0001$). The results are shown in Table 5.5.

Table 5.5: Analysis of variance of the Linear Regression model for usage

Source	**DF**	**Sum of Squares**	**Mean Square**	**F Value**	**Pr > F**
Model	7	34744838	4963548	133.06	< 0.0001
Error	1915	71435077	37303		
Corrected Total	1922	1.06E+08			

The adjusted R^2 of the model is 0.3248, which implies that the model explains 32.48% of the total variance in the bugs reported by users. This value is acceptable for exploratory analysis.

R-Square = 0.3272,

Adj R-Sq = 0.3248

The individual variable significances and maximum likelihood estimates of the parameters in the model are shown in Tables 5.6 and 5.7. Whereas the pair wise correlations of the variables selected in the model are shown in Table 5.8.

Table 5.6: Type 3 analysis of effects of the input variables of Linear Regression model

Effect	**DF**	**Sum of Squares**	**F Value**	**Pr > F**
Quality *Aud	1	465554.249	12.48	0.0004
Size	1	1750121.23	46.92	< 0.0001
Cnt_Mod	1	577442.348	15.48	< 0.0001
Quality	1	547127.749	14.67	0.0001
Cnt_msg	1	7655290.22	205.22	< 0.0001
Age	1	278234.645	7.46	0.0064
Cnt_team	1	7667410.8	205.54	< 0.0001

Table 5.7: Analysis of maximum likelihood estimates of the input variables of the Linear Regression model

Parameter	**DF**	**Estimate**	**Standard Error**	**t Value**	**Pr > |t|**	**95% Confidence Limits**	
						Lower	**Upper**
Intercept	1	51.1639	4.5667	11.2	< 0.0001	42.2132	60.1145
Quality*Aud	1	-0.3682	0.1042	-3.53	0.0004	-0.5725	-0.1639
Size	1	28.2055	4.1179	6.85	< 0.0001	20.1346	36.2763
Cnt_Mod	1	14.4515	3.6731	3.93	< 0.0001	7.2524	21.6505
Quality	1	0.4101	0.1071	3.83	0.0001	0.2002	0.62
Cnt_msg	1	65.1593	4.5485	14.33	< 0.0001	56.2444	74.0742
Age	1	-12.1766	4.4585	-2.73	0.0064	-20.9151	-3.438
Cnt_Team	1	70.5376	4.92	14.34	< 0.0001	60.8945	80.1806

Table 5.8: Pair wise correlation of input variables of the Linear Regression model

Parameter	**Intercept**	**AUD*Quality**	**Size**	**Cnt_Release**	**Quality**	**Forum_Post**	**Age**	**Cnt_Team**
Intercept	1	-0.13672	-0.01276	-0.03808	-0.25689	0.01219	0.05715	-0.00519
AUD*Quality	-0.13672	1	-0.00922	0.02707	0.64235	0.04702	0.00428	0.00508
Size	-0.01276	-0.00922	1	-0.24489	0.01522	-0.22701	0.01387	-0.15954
Cnt_Mod	-0.03808	0.02707	-0.24489	1	0.04779	0.03075	0.04879	-0.18576
Quality	-0.25689	0.64235	0.01522	0.04779	1	-0.05622	0.05857	0.02909
Cnt_Msg	0.01219	0.04702	-0.22701	0.03075	-0.05622	1	0.0267	-0.12277
Age	-0.05715	0.00428	-0.01387	0.04879	0.05857	0.0267	1	0.13383
Cnt_Team	-0.00519	0.00508	-0.15954	-0.18576	0.02909	-0.12277	0.13383	1

The model fit was accessed using the requirements for Linear Regression models. Various specification and diagnostics checks were performed for the estimated model. As mentioned earlier, the assumption for normality was not required considering that the sample size was large. Examination of the Besley-Kuh_Welsch diagnostic (Belsley et al. 1980) indicated that the highest condition number for the model of 4.06 which was within the recommended cutoff limit of 20. The VIF for the independent variables were all below 5, suggesting that multicollinearity was not unduly influencing the estimators (Neter et al. 2004).

The resulting model for usage of OSS projects in maintenance phase is given by:

$$E(Bug\_User\ ) = \beta_0 + \beta_1(Age) + \beta_2(Size) + \beta_3(Qual) + \beta_4(Cnt\ \_\ Msg) + \beta_5(Cnt\_Mod) + \beta_6(Cnt\_Team) + \beta_7(Qual * AUD)$$

The quality (***Qual***) of the code had a positive impact on the bugs reported by users. This means that higher quality projects will have a more involved user community. The usage of the project will increase if the quality is improved and vice versa. A good quality project would keep the users interested in using the project and would have a high level of user retention. On the other hand, if the project quality is low, the users can abandon the project. OSS projects being free and without any contractual obligations, the affect of quality of user contributions would be significant. The code quality also had an interaction affect with the end user audience (***Qual *AUD***). For projects that were targeted purely for developers, the low code quality had a negative affect on the bugs reported by end users, which can be explained by the nature of the end user. For projects developed primarily for the developer community, the end users of the project were also programmers. In such cases, users might find and fix a problem at their own end and never report it. A non-programming user however would depend upon the project team to fix problems. Therefore, a decrease in the bugs reported by end users in case of an interaction between quality and programming audience indicates the differences in the nature of the users.

The forum activity (***Cnt_Msg***) represents the end user involvement in the project. Forums are a useful resource for online communities to participate in project activities. Forums are used to discuss ideas, share experiences and provide feedback. It also indicates

how responsive the project team and the user community are. The forum activity has a positive affect the usage of the project, higher the activity on the project forums, higher the usage of the project.

The count of new file releases (***Cnt_Mod***) is a measure of the functionality of the project. Functionality in operational systems is added through new modules. The addition of new modules indicates that new features have been added to the existing system. Added functionality provides the end users additional incentive to use the project. Therefore, it has a positive impact on the usage of the project.

The team size (***Cnt_Team***) indicates the amount of effort available for project development and maintenance. The team size has a positive impact on the usage of the project. In OSS communities, the participation to a project is voluntary. The motivations of a programmer to participate in an OSS project are beyond the scope of this research. Yet the results indicate that a larger team size has a positive impact on the usage of the project.

The age (***Age)*** and the size (***Size***) of the project were used as the control variables. This ensures that projects of different sizes and ages can be used in the same analysis (Barry et al. 2006).

5.3.2 Model of OSS Project Downloads

The number of downloads of a project had a positive impact on the resilience of OSS projects in the maintenance phase. Since downloads themselves are not a factor that could be

controlled, it was decided to identify the factors that affect downloads. As discussed earlier, the number of downloads could be indicative of the usefulness of the project. If the end user finds the project useful, they will download it. In order to create a model to explain the factors contributing to the number of download of OSS projects in the maintenance phase, the same approach was used as discussed in above section.

The target variable for this model was downloads (***Downloads***). Since the target variable is continuous, Linear Regression was suitable rather than Logistic Regression. The process flow diagram is shown in figure 5.7. All the variables identified in section 3.2.3, were used in the analysis. Decision Tree, Neural Network and Linear Regression were used in the analysis.

The data was split into 40% training, 30% validation and 30% testing samples. The SAS EM 5.2 program offers the option to allow interactions between terms. The final model was selected based on the values of AIC and fit statistics. The Linear Regression model was selected as the final model. The results of the score ranking overlay for the mean predication is shown in figure 5.9. The results of the decision tree and the Neural Network are in Appendix C.

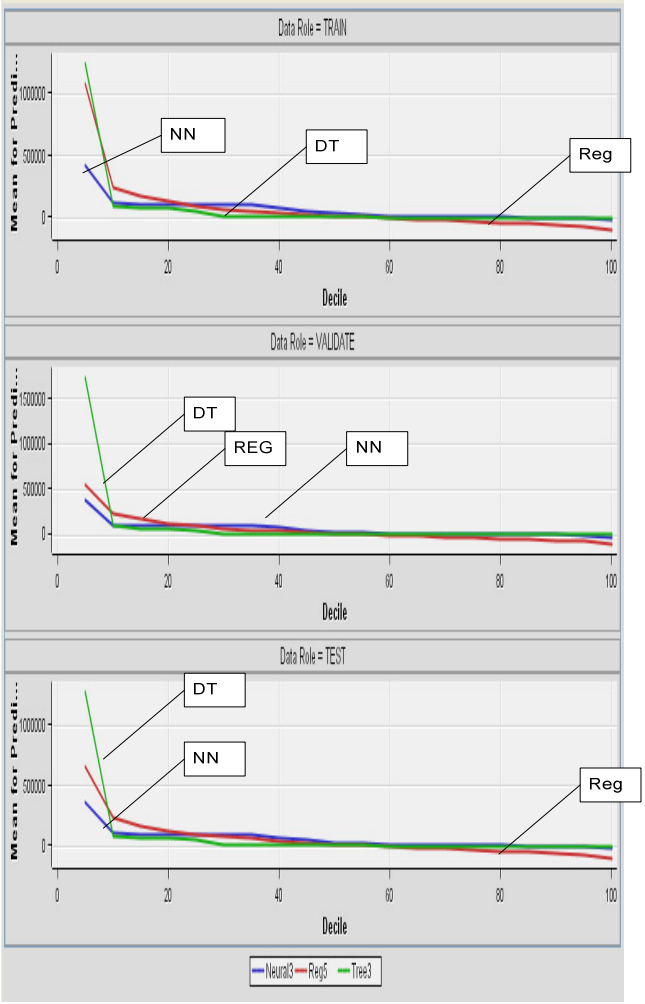

Figure 5.9: The score ranking overlay for the analysis of usefulness

The final model was the Linear Regression model. This model was significant at the 5% level ($p < 0.0001$). The ANOVA results are shown in Table 5.9, while the model fit statistics are shown in Table 5.10.

Table 5.9: Analysis of variance of the Linear Regression model for usefulness

Source	**DF**	**Sum of Squares**	**Mean Square**	**F Value**	**Pr > F**
Model	3	1215.82162	405.27387	656.48	< 0.0001
Error	1982	1223.58202	0.61735		
Corrected Total	1985	2439.40364			

Table 5.10: Model fit statistics of the Linear Regression model for usefulness

Root MSE	0.78571	**R-Square**	0.4984
Dependent Mean	-0.00452	**Adj R-Sq**	0.4977
Coeff Var	-17400		

The R^2 of the model is 0.4984 and the adjusted R^2_{adj} is 0.4977. This means that the model explains 49.77% of the variance in the values of project downloads. This value is acceptable for exploratory research. The individual significances of the independent variables are shown in Table 5.11.

Table 5.11: Significances of independent variables of the Linear Regression model of usefulness

Variable	DF	Parameter Estimate	Standard Error	t Value	Pr > |t|	Tolerance	Variance Inflation
Intercept	1	-0.00619	0.01764	-0.35	0.7257	.	0
Cnt_Msg	1	0.56765	0.01835	30.94	< 0.0001	0.91740	1.09004
Size	1	-0.15977	0.01639	-9.75	< 0.0001	0.57306	1.74502
Cnt_mod	1	0.43729	0.02086	20.96	< 0.0001	0.53746	1.86062

Table 5.12: Correlation of estimates of the input variables of the Linear Regression model of usefulness

Variable	Intercept	Cnt_Msg	Size	Cnt_mod
Intercept	1.0000	-0.0054	-0.0342	0.0124
Cnt_Msg	-0.0054	1.0000	0.1296	-0.2791
Size	-0.0342	0.1296	1.0000	-0.6511
Cnt_mod	0.0124	-0.2791	-0.6511	1.0000

The model fit was accessed using the requirements for Linear Regression models. Various specification and diagnostics checks were performed for the estimated model. The usual diagnostic checks of the regression residuals indicated no serious departures from the underlying assumptions. Examination of the Besley-Kuh_Welsch diagnostic (Belsley et al. 1980) indicates that the highest condition number for the model is 2.29 which is within the

recommended cutoff limit of 20. The VIF for the independent variables were all below 5, suggesting that multicollinearity was not unduly influencing the estimators (Neter et al. 2004). The resulting model for the number of downloads or the usefulness of OSS projects is given by the following equation.

$$E(Downloads\ ) = \beta_0 + \beta_1(Size) + \beta_2(Cnt\ \_\ mod) + \beta_3(forum\ \_\ posts)$$

The coefficient of the control variable of Age (***Age***) was not significant in the model for downloads. This could be because of the fact that the variable downloads was normalized for Age of the project in the original dataset. Therefore any affects of the age of the project on the number of downloads was already adjusted in the data. The size (***Size***) of the project was used as a control variable in the model.

The increase in functionality of the project was operationalized as the count of number of new modules or releases (***Cnt_mod***). This variable had a positive impact on the number of downloads. Therefore, if the functionality of the project increases, its usefulness (or downloads) will increase and this will have a positive impact in the end on the resilience of the project. The forum activity (***Cnt_Msg***) had a positive impact on downloads. In projects where the user and the developer community are actively engaged in discussions, the project downloads increase. The use of forums in cyberspace is very critical. Forums are not only used to discuss project related issues, but are also a good tool for advertising the project. Thus projects with active forums had higher number of downloads.

5.4 CONCLUSIONS

Projects that possess high resilience will be able to remove errors and become operational effectively. The fast response time implies that the errors were not very severe in the first place. If the errors were severe, then the ability to fix the error in small time implies that the code was well designed so that the maintainers were able to detect and remove the problem. It also indicates the effectiveness of the maintenance team. Therefore, the mean time to fix a bug is a very important characteristic of a project.

A model to explain the factors that affect the resilience of an OSS project was developed in this chapter. A two-stage model identifying the key critical components of project resilience was developed. The model is shown in Figure 5.7. It is discovered that the role of the end user of the project is very critical in the performance of OSS projects. The participation of the user in the maintenance process, improves the ability of a project to react to errors and faults. It is also discovered that the number of downloads also improve the resilience of OSS projects. The usefulness and the usage of a project had direct positive impact on the project resilience. Since both these factors were not directly in the control of the development team, it was decided to identify the factors that affected the usage and usefulness. The resulting two new models revealed that product functionality and quality had a positive affect on the usage. The end users were encouraged to participate in the maintenance process if the project offered them a high quality code with increasing functionality. An active user community also improved the usage of a project.

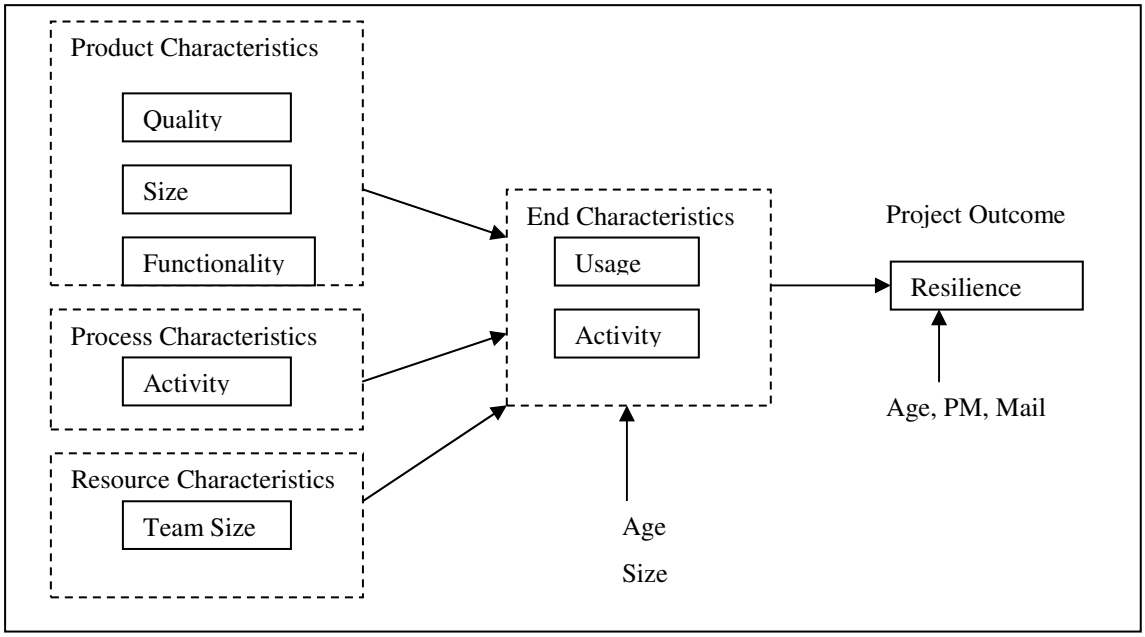

Figure 5.10: Model for resilience of OSS projects

The project functionality and the forum activity were also significant factors in the usefulness or the number of downloads of the project. If the project offered significant functionality and had active forums, the number of downloads increase. This could be attributed to the existing users continuing use of the project since they found it useful, or it could also imply that the other OSS users were more likely to download projects that offered more functionality and had active forum participation. Tables 5.13-5.18 show the variables used in the final model.

Table 5.13: Process related variable measurement and sources

Variable	**Measure**	**Symbol**	**In final Model**
Project Management	Use PM (Y/N)	*Use_PM*	
Process Quality	Mean Time to fix a bug (MTTF)	*MTTF*	
Communication Channel	Forum use (Y/N)	*Use_forum*	
	Number of forums	*Cnt_forum*	
	Use Mail (Y/N)	*Use_mail*	☑
	Use News Groups (Y/N)	*Use_news*	
Req. Implementation	Time to implement a feature	*TTFT*	
Configuration Management	Use CVS (Y/N)	*Use_CVS*	
	Count CVS commits	*Cnt_CVS*	
Process Quality	Count of bugs	*Bug_Cnt*	
	Bugs not fixed	*Bug_Open*	☑

Table 5.14: Control variable measurement and sources

Control Variables	**Measure**	**Symbol**	**In final Model**
Size	Source Lines of Code	*Size*	☑
Age	Time elapsed since the start of project	*Age*	☑

Table 5.15: Resource related variable measurement and sources

Variable	**Measure**	**Symbol**	**In final Model**
Effort	Number of registered developers for the project	*Cnt_Team*	☑
Team Communication	Messages posted at development forums	*Forum_post*	

Table 5.16: Product related variable measurement and sources

Variable	**Measure**	**Symbol**	**In final Model**
Functionality	Increase in features	Cnt_feat	
	New Modules	Cnt_file	☑
Maintainability	Number of distinct members reporting the bugs	Cnt_mem	
	Number of distinct members fixing the bugs	Cnt_Usr_fix	
Portability	Number of platforms supported	Cnt_OSI	
	Number of programming languages supported	Cnt_Prg_lang	
License Type	License type	Lisc	
Project Type	Primary categorization of the project.	Pjr_type	☑
Usefulness	Downloads	Downloads	☑
	Page Views	Pg_View	
Product Compatibility	Number of translations	Cnt_tran	
	Number of platforms supported	Cnt_plat	
Usage	Usage by end user	Bug_User	☑

Table 5.17: User related variable measurement and sources

Variable	**Measure**	**Symbol**	**In final Model**
User Type	Audience Programmer (Y/N)	*AUD (o= prog, 1 = non-prog)*	☑
Activity Level of User	Forum posts by users	*Cnt_Msg*	☑
	Number of distinct individuals posting messages, bugs	*User_Int*	
Community Size	Number of distinct senders of messages	*Cnt_User*	

CHAPTER VI

MODEL OF ORGANIZATION

In this chapter, the affects of maintenance on the project organization are discussed. A new taxonomy for maintenance patches is developed and presented. The internal organization of the project is measured in terms of complexity. The affects of patch types on complexity are statistically analyzed. The chapter ends with s discussion of results.

6.1 BACKGROUND

Software systems are constantly changing and growing throughout their useful lives (Lehman and Ramil 2002). For any software system, the majority of the lifecycle cost and effort is expended in the detection and elimination of errors or in functionality enhancements during system maintenance (Lehman and Ramil 2002; Swanson and Dans 2000). Addition of new functionality can make the task of software maintenance more difficult. It is usually accompanied by new errors, making the maintenance task even more complex (Brooks 1995) .

Software maintenance activities vary in nature, ranging from removal of faults and errors to introduction of new functionality. In software engineering literature, software maintenance activities have been categorized as corrective, perfective, adaptive and preventative. Corrective maintenance refers to the correction or removal of defects. Adaptive maintenance refers to the modifications made to accommodate changes to external environment e.g. new hardware. Perfective maintenance refers to changes that extend the

original functionality of the software and preventative maintenance refers to the tasks that are carried out to prevent or facilitate the maintenance process (Pressman 2004; Swanson and Beath 1997).

Prior literature suggests that software maintenance activities can cause a decline in the operational performance of software (Brooks 1995; Eick et al. 2001). This decline in the quality has been attributed to the increase in the structural complexity of the software associated with software maintenance. As new functionality is added, the interfaces become more complex. Often removal of old code and comments is overlooked which results in increased code complexity. The code complexity can further make maintenance tasks more difficult and thereby increasing the costs (because of increased effort) and the quality declines. The result of inefficient code and complex interfaces is that quality of the software declines.

The advocates of OSS movement attribute the quality of OSS projects to a very active maintenance process. In OSS projects, the source code is publicly available and all users can be involved in the maintenance process. Raymond proposed the famous Linus Law that states, “Given enough eyeballs, every bug is shallow”. The OSS projects have the philosophy of “release often”. This means that the maintenance activities are more rigorous in OSS projects. As OSS projects are becoming more popular, it is necessary to investigate how the software is affected by being developed and maintained through the OSS philosophy. There has been no empirical research on how the phenomenon of frequent updates affects internal characteristics of the source code. Considering the impact of

maintenance activities on software performance, there is a need to understand these activities (Greiner et al. 2003). In this model, the effects of various maintenance activities on project Viability, primarily the dimension of Organization were analyzed.

Software maintenance activities in an operational OSS project are implemented thorough Patches. OSS patches contain rich text references to the changes that are implemented through them. These textual references are made by online, geographically dispersed teams of developers and maintainers to explain the significance of the code they added or removed (Stamelos et al. 2002). Although there exits a classification of maintenance activities in the literature, there is an absence of any formal classification of software patches. For this research, textual information available in software patches was extracted, to develop a classification scheme for maintenance patches. This classification was based on the individual type of maintenance activities performed by each patch.

The purpose of this research was to analyze the affects of software maintenance on the internal structure of the source code. Complexity is a validated measure on the structure of software (Fenton and Pfleeger 1991; Simon 1994). The Cyclomatic Complexity metric is the most widely used and accepted measure of complexity. Introduced by Thomas McCabe in 1976, it measures the complexity of the software program by the number of linearly independent paths in program modules. The measure provides a single number, which can be used to compare the complexity of various programs with each other. Cyclomatic complexity is often referred to simply as program complexity or as McCabe's complexity. McCabe’s Cyclomatic is language and platform independent.

Large amount of maintenance data over a period of time was required to explore the affects of maintenance on complexity. Therefore, public archives of Linux source code were used. The reason for selecting Linux was the availability of rich lifecycle maintenance data for a single project. Linux maintains a record of its parallel experimental and production versions. The experimental versions are more volatile than the stable versions and tend to change more frequently. The stable versions were suited for this analysis. Linux versions 1.0 through version 2.6.5 were used in this research. The details of the Linux dataset were discussed in section 3.2.2.2.

6.2 TAXONOMY DEVELOPMENT OF LINUX PATCHES

The first challenge was to categorize the maintenance tasks. The available taxonomy for software maintenance had been developed at a single task level. In operational software, it is very rare that a single maintenance task would be performed. Typically, the vendor releases a new patch in which several maintenance tasks are bundled. The patch once applied to the existing software, implements the changes and the software is moved to the next version. Frequent new patches characterize OSS. Considering prior research in software maintenance, the purpose of this research was to analyze if there were differences in the maintenance activities performed through different patches. Based on the nature of the tasks performed by the patch, it would then be categorized into a new taxonomy of patches. The taxonomy would be used to analyze the affects of various types of patches on the software internal characteristics.

As discussed in Chapter II, Linux is one of the most popular and largest OSS project. Some research has been conducted on the evolution of Linux source code, but no prior study on the Linux patches and their affects on complexity have been performed. To develop taxonomy of the Linux patches the data on the patches was collected. Patch is available as a text document and is a combination of textual description of the patch and the actual code, which is embedded in the text. Programmers use comments to explain the maintenance tasks. In prior research, the nature of the maintenance tasks has been associated to the key words used to explain the task. The Linux patch data is no different. Rich comments were available on all changes introduced in the new patch.

The text data on all the versions starting from version 1.0 until version 2.6 were collected. Linux maintains parallel versions, in which all even numbered versions are stable production versions and all experimental versions are odd numbered. Only stable versions were used in this analysis. There were in all 132 patch files for stable Linux kernel from version 1-0-0 until version 2-6-13. Total time period of the release of the versions was March 9th, 1995 until March 12, 2005. The patches were available in compressed (.gz) format. They were converted to text files for this analysis.

While performing text analysis, text is used as one of the fields in the dataset. The limit of the text field is 32000 characters. However, the size of Linux patches was very large (on average a file size was 2 MB). Therefore, direct import to a single text field for every version was not possible. SAS TM 5.2 is equipped with a tmfFilter. This filter allows creating links to file locations and accessing text only during the analysis from the stored

location. The URL for the stored text is then entered in the text miner settings to perform the analysis (See Figure 6.1).

Once the text node settings were complete, the next step was to identify the key words for the analysis. As mentioned earlier, maintenance key words have been used in past CSS research to categorize maintenance activities. These analyses however were on very small datasets. Since OSS projects are purely developed without any face-to-face communication between the maintenance teams, the comment density in such code is very rich. The maintainer had to ensure that others using the code could understand any changes made and that the project is maintainable.

A preliminary list of the key words was developed using prior research in software maintenance. This list contained 33 key words. The text analysis of the Linux patch files was performed. This resulted in only two clusters. Further analysis of the patches indicated that the list of 33 words was not sufficient and a richer start list was required. Therefore, a new run using the SAS EM default stop list was performed.

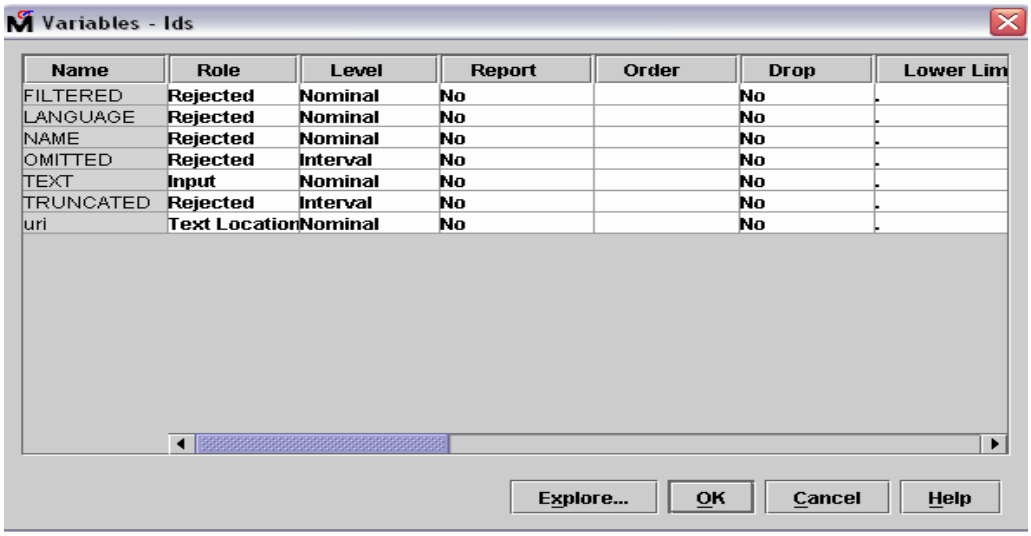

Figure 6.1: Text miner node selection of roles for the input variables

The new run resulted in seven clusters with 20,000 descriptive terms. Not all the terms in the text were related to maintenance. Many of the terms were from the actual source code itself. Therefore, these clusters could not be used for taxonomy development either.

The key terms from this run were carefully analyzed and the key words that pertained to maintenance activities were selected. To ensure that the result was not biased an independent programmer with experience in software development was assigned to identify the maintenance related key words from the list. The lists prepared by the primary researcher and the secondary coder were compared and a new start list was created. This start list had 312 terms related to software maintenance.

Table 6.1: Descriptive terms, percentage and type of the clusters of maintenance patches

Cluster	Descriptive Terms	Percentage	Patch Type
1	+ error, + buffer, + problem, + interrupt, + set	0.363636	Corrective
2	+ clear, + create, + implement, + remove, + check	0.289256	Preventative
3	+ enable, + allow, + change, + disable, + add	0.330579	Adaptive

The text miner node performing clustering had been set to allow for a maximum of 40 clusters. Yet the result was three clusters as shown in Table 6.1. From the descriptive terms, it can be seen that the terms in the first cluster are related to corrective maintenance. 36% of the patches were clustered in this category. The second cluster had words related to preventative maintenance activities. These terms indicate that maintenance tasks were performed to improve the performance of the software. The third cluster has terms related to adaptive tasks that are performed in response to changes in the environment e.g. interface or hardware etc. Based on these terms three types of patches were referred as corrective patches, perfective patches and adaptive patches.

The SAS TM 5.2 also allows creating concept links of the key terms from the analysis. The results were opened in the interactive browsing mode. Selecting a particular key word, the concept links of that term could be identified. This produces a map of how various term occurrences is related in the data set. This can allow tracing how various maintenance activities affect each other. As an example of concept links, the key term, “error” was selected. The concept links of the term are shown in Figure 6.2. The concept

map shows what terms have common occurrences of the term “error”. The thickness of the link indicates the strength of the relationship. The map gives a pictorial view of the common occurring errors in the patches. Next, the term “allocate” was selected and expanded for further mapping. This split the map to more detail and the terms associated to “allocation” were also mapped. This identifies the common allocation errors that occur.

The concept maps can be used as affective tools in patch management. As mentioned earlier, new patches could be scored to one of the three clusters defined in this research. The key terms from that patch could then be used to map the potential maintenance activities that could be related to the new patch. In case of Linux, which is an operating system, any new patch is associated to extensive testing on part of organization. They have to ensure the integrity of new releases before they can roll out the new patch. This ensures that there would be no major problems associated to the new patch. The concept links could be used to identify the potential problem areas and therefore the maintenance teams in the organizations could prepare better plans for new patch roll out.

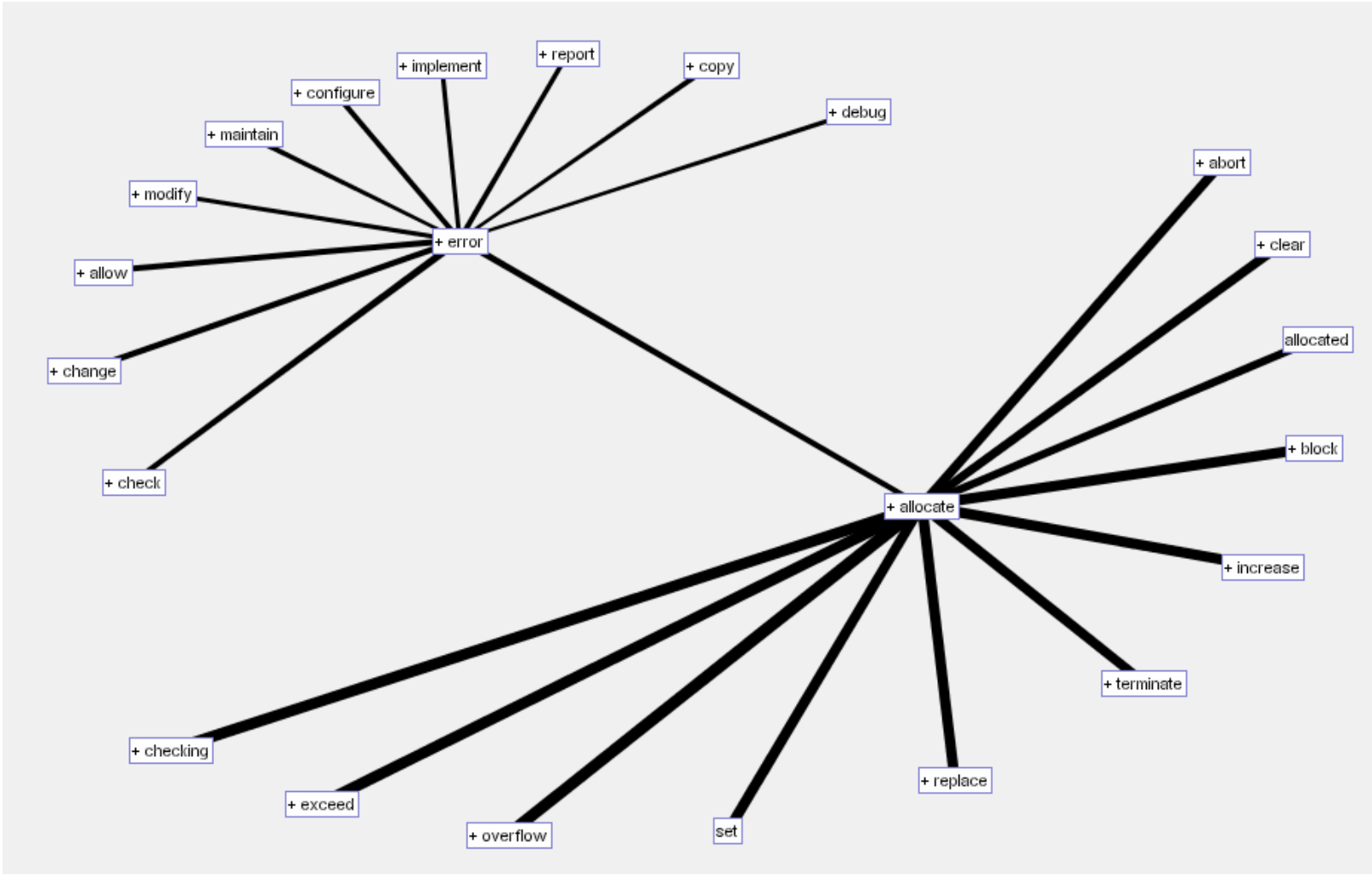

Figure 6.2: Concept map of selected terms for the patch taxonomy

This taxonomy of the maintenance patches is a new addition to the area of software maintenance and software security. Most of the software security costs and effort is expended on managing the implementation of new patches. This taxonomy provides a categorization of the patches. The affects of each patch type could now be investigated individually. Therefore, the next step in this exploratory study was to examine if there was a difference in the affects of the patch types on the existing nature of the software.

6.3 STRUCTURAL COMPLEXITY

As discussed previously, introduction to new code can cause an increase in the structural complexity of the software. As the complexity of the software increases, the quality declines and the effort needed to maintain that software increases (Brooks 1995; Eick et al. 2001; Lehman and Ramil 2002). Maintenance tasks could be designed to reduce the complexity and prolong the life of software. However, there is no empirical evidence whether there is a difference in the affects of various maintenance activities on the complexity of software. The Linux source code kernel was used, to test the affects of the patch types on the software complexity empirically.

The Linux Kernel source code is freely available to the public. It is available in compressed form and can be downloaded and uncompressed into its original directory structure. Reynolds's tool is a free online tool used for measuring software complexity. This tool was used to compute the McCabe's Cyclomatic for software. To ensure the reliability of the results, test files of known complexity were tested through the Reynaud's tool and its integrity was confirmed.

Each version of the Linux kernel composed of several modules. McCabe's complexity is a module level measure. The total complexity of a version was computed as the sum of the McCabe's complexity of each module. For a version with n modules, the complexity is given by:

$$Complexity = \sum_{i=1}^{n} Mcb_i$$

Since the data was collected over a period of time, the time series plot of complexity was generated as shown in Figure 6.3 and 6.4.

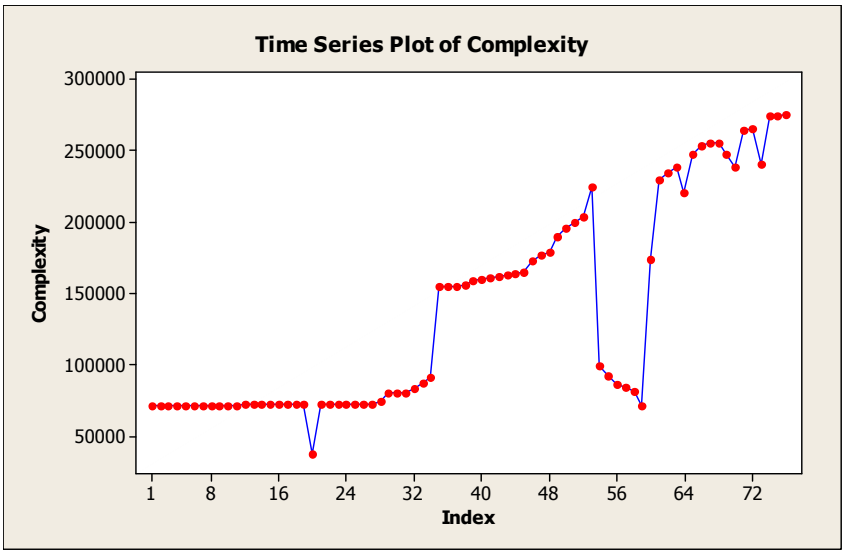

Figure 6.3: Time series plot of the Linux source code complexity

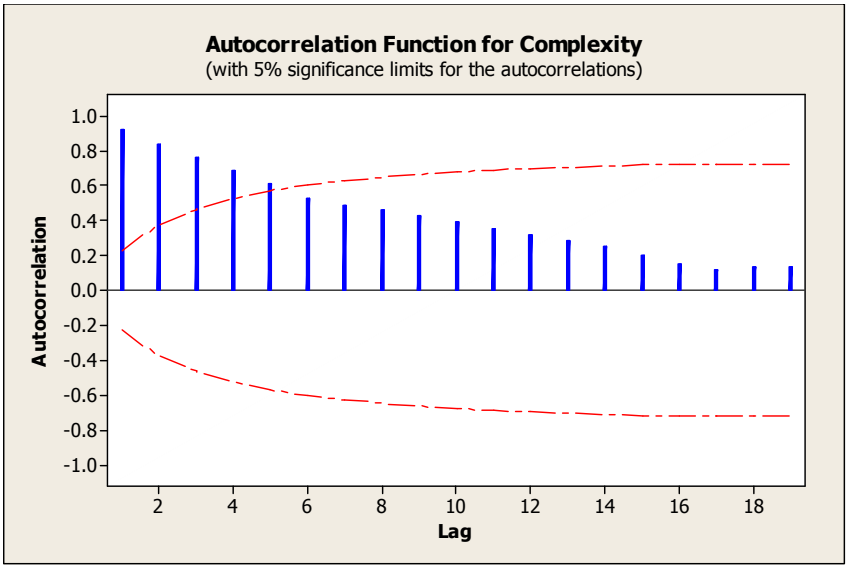

Figure 6.4: Autocorrelation function of the Linux source code complexity

Figure 6.4 indicates a trend in the data. Therefore, the difference in the complexity of two subsequent releases was taken. The variable del_complexity represents the change in complexity between two subsequent releases. The time series plot and the autocorrelation function plot of the del_complexity are shown in Figure 6.5 and 6.6 respectively. The differences indicated that the trend has been removed.

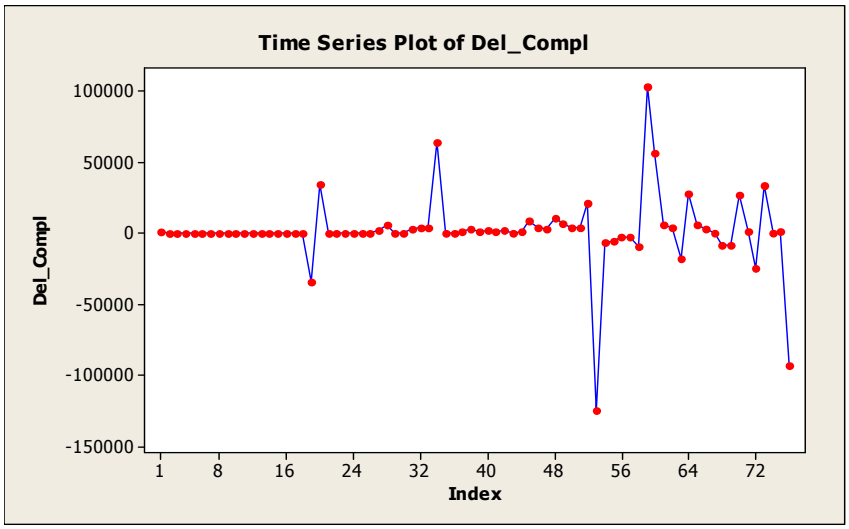

Figure 6.5: Time series plot of del_complexity

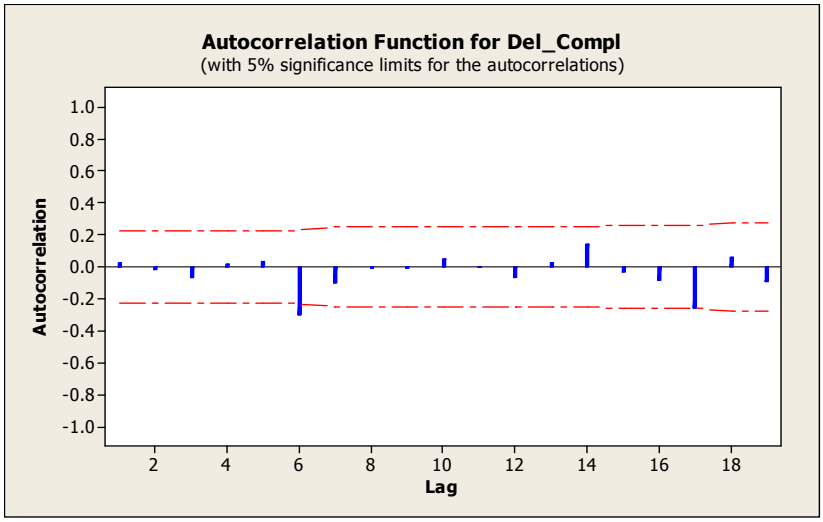

Figure 6.6: Autocorrelation plot of del_complexity

The purpose of this model was to analyze the affects of maintenance on the source code. Del-complexity is a reflection of change in complexity rather than the overall complexity; therefore, it is a better variable for the analysis.

The variable time to next release was also computed for each version. This variable indicated the number of days elapsed between two patch releases. Whenever a new patch is released, the user has to perform testing before the patch can be implemented. Sometimes the delay due to testing is translated into operational costs in terms of down time. If the patch release is too frequent, there could be significant financial impact. Therefore, for every patch the time to next release (del_time) was computed. This indicated how long the software was operational before a new major change was introduced through the patch. Frequent patch release could also imply that there is a ripple affect in the errors and the code introduced to fix the errors is causing new errors.

6.4 ANALYSIS

The cluster_id of each patch was then merged with the data on del_complexity and del_time. The objective was to examine if there was a difference in the del_complexity and del_time for the three types of patches or not. The descriptive statistics for the data is shown in Table 6.2.

Table 6.2: Descriptive statistics of the three clusters

Level of Cluster_id	N	del_comp		del_T	
		Mean	**Std Dev**	**Mean**	**Std Dev**
1	26	6927.1154	20985.4267	36.7692308	55.4179088
2	28	-10222.5357	30283.4820	57.6428571	62.9586662
3	22	7242.4545	15461.7787	19.1818182	18.6614866

It can be seen that the mean of del_complexity of the preventative patches is a high negative number, whereas the means of the corrective and adaptive patches are both positive numbers. This suggests that the preventative patches reduce the overall complexity of the software and the corrective and adaptive patches increase the complexity. It can also be seen from the means that the del-complexity of preventative patches is higher than the other two types. Higher del-time means that preventative patches have a long time between successive releases. This could be indicative of improved quality and fewer errors being introduced by the patch. To check if there is a significant difference in del-complexity and del-time between the three types of patches, an ANOVA analysis was performed. The three assumptions of ANOVA are:

- Independent observations
- Normally distributed error terms
- Equal error variance for each group

The first assumption was met by the data because it was collected for separate versions of the Linux data. The residual vs. the fitted values of the ANOVA were tested to

ensure the normality assumption. There was a random scatter about the zero reference line for each of the fitted values. Therefore, the normality assumption was met.

To test for equal variance, the Levene's Test for Homogeneity of the variance of del-complexity and del-time was performed. The null hypothesis of the test was that the variances are equal. Failing to reject the null hypothesis would imply equal variance. Table 6.3 and 6.4 show the results of the Levene's test.

Table 6.3: Levene's test for del-complexity of the Linux patch clusters

Levene's Test for Homogeneity of del_comp Variance ANOVA of Squared Deviations from Group Means					
Source	**DF**	**Sum of Squares**	**Mean Square**	**F Value**	**Pr > F**
Cluster_id	2	5.811E18	2.906E18	0.72	0.4879
Error	73	2.927E20	4.01E18		

Table 6.4: Levene's test for del-time of the Linux patches clusters

Levene's Test for Homogeneity of del_T Variance ANOVA of Squared Deviations from Group Means					
Source	**DF**	**Sum of Squares**	**Mean Square**	**F Value**	**Pr > F**
Cluster_id	2	1.5764E8	78818265	2.69	0.0749
Error	73	2.1418E9	29339769		

The p value of the F statistic was both the cases was greater than 0.05, therefore the null hypothesis of equal variance was not rejected at the 5% level. The next step was to perform ANOVA analysis to test if the means of the three types of clusters were significantly different. The results of the analysis are shown in Table 6.5 and 6.6.

Table 6.5: ANOVA analysis of the dependent variable del-complexity

Source	**DF**	**Sum of Squares**	**Mean Square**	**F Value**	**Pr > F**
Model	2	5290332788	2645166394	4.73	0.0117
Error	73	40791512489	558787842		
Corrected Total	75	46081845277			

Table 6.6: ANOVA Analysis of the dependent variable del-time

Source	**DF**	**Sum of Squares**	**Mean Square**	**F Value**	**Pr > F**
Model	2	18491.3675	9245.6838	3.53	0.0344
Error	73	191114.3167	2618.0043		
Corrected Total	75	209605.6842			

In both the cases, the value of F-statistic was significantly greater than 1 (p value of F statistic was less than 0.05); therefore the null hypothesis of equal means was rejected. These results indicate that preventative patches can significantly reduce the complexity of software. The time to next patch was also significantly higher than the rest of the types of patches. This finding can be very critical in software development and maintenance. Regular preventative maintenance can control the increase in complexity. This could improve software quality. Preventative maintenance would also remove any redundant code and make the software more maintainable.

6.5 CONCLUSION

This part of the research developed a new a new taxonomy for categorizing maintenance patches. This is the first empirical study on how individual maintenance activities are clustered together in a single maintenance patch. It used the textual data of patches through Text Mining techniques to develop the taxonomy. Use of prior literature on software maintenance identified the terms for maintenance tasks. More terms were added to the list by visual inspection of the key words. The new list was used to categorize the Linux Patches. The resulting taxonomy consisted of three types of patches: Corrective, Preventative and Adaptive.

Data on Linux source code complexity was extracted. This was done to examine the affects of various types of patches on the software internal structure. Maintaining structure is a challenge of software maintenance. Increase in the complexity can result in decline in quality. Analysis was performed to see if there was a difference in the patch types in terms

of change in complexity and the time to next release. The means of the patch types indicated that the preventative patches reduce the complexity and have a longer time to next release. The statistical significance of the differences was tested and it was significant.

CHAPTER VII

SUMMARY AND CONCLUSION

This chapter summarizes the results of this research and identifies the contribution to theory and practice. This is followed by the limitations of the study. The chapter ends with the implications for future research and conclusions.

7.1 SUMMARY OF RESULTS

The primary objective of this research was to examine the development and maintenance activities of OSS projects. A new measure for evaluating the performance of OSS projects was defined, validated and tested. Three models were developed to identify the factors that affect the development and maintenance performance of OSS projects. These models were developed using Data Mining techniques. The detailed results were presented for each model that provided a deeper insight into the OSS development and maintenance process.

The measure of software viability could be used to measure the performance of a single project over its lifecycle, or for comparison among multiple projects. It was found that the three dimensions of project viability provide the predictive power for the performance of OSS projects. Considering the nature of OSS project development, the characteristic of the end user were also used in the model in order to examine the affects on project performance. A new variable “project type”, based on the description data was used. The key findings for each model are discussed in the following sections.

7.1.1 Model 1: Development Phase Performance

This model was developed to identify the factors that affect the vigor of OSS projects. It was discovered that the end user plays a significant role in the development of the project. Projects where the end user participated in the forums and in the maintenance activities had a higher probability of having a high vigor. It was shown that the team size and the use of project management methods improve the performance of OSS projects in the development phase, whereas the inability to fix problems could have a negative impact on the project outcomes.

It was also found that some OSS project types were better suited for development compared to other projects. The type of the audience for which the project had been developed also played a role in project development.

Age was used as a control variable. It was seen that older projects had less chance of growth compared to newer projects. This could be attributed to loss of quality or age, or to improvement of OSS methods with time.

7.1.2 Model 2: Maintenance Phase Performance

The second model was developed for the resilience of OSS projects in the maintenance phase. A two-stage model was developed. The effects of bugs reported by end user (usage) and the number of downloads (usefulness) on resilience of a project was demonstrated. Both these variables were outcome variables; therefore, further models were developed to explain the factors affecting the usefulness and usage. These models identified

that the quality of the OSS project encourages users to be involved in that project. The team size and functionality affected the usefulness of the project.

7.1.3 Model 3: Affects of Maintenance on Structure

A new taxonomy for maintenance patches was developed for Linux patches. Three types of patches were defined: corrective, preventative and adaptive. The affects of these patch types on the internal structure of the software were examined. The preventative patches reduce complexity and have a longer elapsed time to the next release. Corrective and adaptive patches have an opposite affect. This indicates that suitable preventative maintenance tasks can improve the quality of a project and increase its useful life,

7.2 CONTRIBUTIONS TO THEORY

This research has contributed to the theory in a number of ways. It is the first empirical study to explore the development and maintenance of OSS projects. The study demonstrates the role of end user in the performance of OSS projects. It also provided the first empirical evidence to substantiate the argument that end user involvement improves the project performance.

The study utilizes data mining techniques for model building. This is very significant in inductive research. The public availability of large datasets will make use of these techniques suitable for conducting exploratory research. Use of multiple data mining techniques to improve the predictive power of the resulting models was also demonstrated.

In addition, a new measure of OSS project performance was developed, validated and tested. Academic research in OSS has lacked well-developed measures that could be used to evaluate OSS projects. The new measure was developed through the software measurement framework.

Text analysis was used to develop new variables of project type. This provided a new categorization for OSS projects. Using this categorization, projects were placed in only one category, unlike the existing SourceForge categorization that puts a single project in multiple categories. The use of text analysis in taxonomy development was also verified. A new taxonomy for maintenance patches was developed. This was the first body of work on empirical analysis of maintenance patches and their categorization.

7.3 CONTRIBUTIONS TO PRACTICE

This research provided the practitioner community with new tangible measure that could be used to evaluate OSS projects. Business investments in OSS projects could make use of such measures to make informed decisions regarding OSS projects. This study also provided patch taxonomy and its impact on the internal structure of software. In businesses, application of new patches can be very expensive in terms of cost, effort and risks. The new taxonomy could be used to score new software patches, before any implementation is performed. Concept links could be used to identify potential modules that could be affected. This could help in maintenance planning.

The study also indicates the significance of the role of end user in the performance of a project. This information could be generalized to CSS where software development teams could benefit user involvement. The study also indicates that OSS might be a more suitable development platform for certain types of projects. Therefore, practitioners could decide on OSS or CSS usage depending upon the nature of the problems.

For effective management of OSS projects, the development teams could use the results of this research to improve the performance of their projects. They could monitor their projects using the measures identified. They could also control the outcomes of their projects considering the factors that affect the outcomes.

7.4 LIMITATIONS OF THE STUDY

Following are the limitations of this research study. One limitation is that the dataset used for model building was from SF and Linux. Although the SF community is the largest OSS project hosting community, but is not the only community. Too many idle or inactive projects had to be removed from the analysis. Therefore, in future comparisons can be done with other communities such as Tigris.org, freshmeat.net that are more selective in project hosting. The data was provided to public use through a third party. Although data is released through the joint effort with SF, the warehouse is cleaned and maintained by University of Norte Dame. The study investigated the affects on cross-sectional data, therefore changes made to projects over a period of time, were not explained by the analysis.

7.5 IMPLICATIONS FOR FUTURE RESEARCH

Several directions for future research can be derived from this exploratory study. The findings of the factors affecting OSS performance could be tested in the CSS domain. The model could also be used to test the performance of a single project over its lifecycle.

Considering the high number of OSS projects that fail to grow and survive, it would be interesting to use techniques like survival analysis. This would give a deeper insight into how projects evolve and what accounts for their failure. A dataset of the projects that failed or became inactive has already been prepared. Another dataset for projects that evolved through multiple phases has been prepared too. These two datasets could be used to perform survival analysis and study the behavior of these projects as they fail or evolve.

The activity of the forums had a positive impact on the project outcomes. One direction could be to study the patterns of communications amongst the teams and compare them to the structure of the software code. This would mean testing Conway's Law in the software development domain. Preliminary expertise in the use of social network tools has already been developed. Some software applications that would be needed to convert the OSS data for use in social network has also been developed. These could be used in future to get a better understanding of the communication structure of OSS projects.

The use of Text Mining to improve the predictive ability of models has been demonstrated in this research. Use of additional techniques e.g. Sequence analysis and Association analysis on the maintenance documents could provide insight into how the various patches are related to each other and identify the factors that trigger new corrective patches.

The new measure of software viability can be tested for performance on CSS projects or on OSS projects in other development communities. This would result in triangulation of the results of this study. A new tool can also be developed to measure the viability of a project. This tool would accept the basic project measures as inputs and using refined models from this research compute a viability score for the projects. This can be a very useful tool for the practitioner community.

REFERENCES

Abdel-Hamid, T. "Understanding the '90% Syndrome' in Software Project Management," *Journal of Systems and Software* (8:4) 1988, pp 319-330.

Abdel-Hamid, T. "The Dynamics of Software Project Staffing: A System Dynamics Based Simulation Approach," *IEEE Transactions on Software Engineering* (15:2) 1989, pp 109-119.

Abdel-Hamid, T. "Investigating the Impacts of Managerial Turnover/Succession on Software Project Performance," *Journal of Management Information Systems* (9:2) 1992, pp 127-144.

Albrecht, A.J., and Gaffney, J.E. "Software Function, Source Lines of Code, and Development Effort Prediction," *IEEE Transactions on Software Engineering* (9:6) 1983, pp 639-648.

Allison, P.D. "Measures of Inequality," *American Sociological Review* (43) 1978, pp 865-880.

Aoki, A., Hayashi, K., Kishida, K., and Nakakoji, K. "A Case Study of the Evolution of Jun: An Object Oriented Open Source 3D Multimedia Library," International Conference on Software Engineering, Toronto, Ontario, Canada, 2001, pp. 524 - 533.

Banker, R., Datar, S., and Kemerer, C. "Software Errors and Software Maintenance Management," *Information Technology and Management* (3:1-2) 2003, pp 25-41.

Banker, R.D., Davis, G.B., and Slaughter, S.A. "Software Development Practices, Software Complexity and Software Maintenance Performance," *Management Science* (44:4), April 1998, pp 433-450.

Banker, R.D., and Slaughter, S.A. "A Field Study of Scale Economies in Software Maintenance," *Management Science* (43:12) 1995, pp 1709-1725.

Barry, E.J., Kemerer, C.F., and Slaughter, S.A. "Software Volatility: A System-Level Measure," AIS Conference of the Americas, Long Beach, CA, 2000.

Barry, E.J., Kemerer, C.F., and Slaughter, S.A. "Environmental Volatility, Development Decisions, and Software Volatility: A Longitudinal Analysis," *Management Science* (52:3) 2006, pp 448-464.

Barry, E.J., and Slaughter, S.A. "Measuring Software Volatility: A Multi-Dimensional Approach"," ICIS International Conference on Information Systems, Brisbane, Australia, 2000, pp. 412-413.

Belsley, D.A., Kuh, E., and Welsch, R.E. *Regression Diagnostics: Identifying Influential Data and Sources of Collinearity* John Wiley & Sons, New York, NY, 1980.

Bennett, K.H., and Rajlich, V.T. "Software Maintenance and Evolution: A Roadmap," International Conference on Software Engineering, ACM Press, Limerick, Ireland, 2000, pp. 75-87.

Bergquist, M., and Ljungberg, J. "The Power of Gifts: Organizing Social Relationships in Open Source Communities," *Information Systems Journal* (11) 2001, pp 305-320.

Berry, M.J.A., and Linoff, G.S. *Data Mining Techniques: For Marketing, Sales and Customer Relationship Management*, (2nd ed.) Wiley Publishing, Inc, Indianapolis, IN, 2004, p. 643.

Berry, M.W., and Browne, M. *Understanding Search Engines : Mathematical Modeling and Text Retrieval*, (Second ed.), Philadelphia, PA, 2005.

Boehm, B. "Software Engineering Economics," *IEEE Transactions on Software Engineering* (10:1) 1984, pp 4-21.

Boehm, B. "Improving Software Productivity," *IEEE Computer* (20:1) 1987, pp 43-57.

Boehm, B. "A Spiral Mode of Software Development and Enhancement," in: *Software Project Management: Readings and Cases,* C.F. Kemerer (ed.), Irwing Book Team, Chicago, IL, 1997, pp. 254-270.

Briand, L., Emam, K.E., and Morasca, S. "On the Application of Measurement Theory in Software Engineering," *Empirical Software Engineering* (1:1) 1996, pp 61 - 88.

Brooks, F.J. *The Mythical Man-Month*, (Anniversary Edition ed.) Addison-Wesley, Reading, MA, 1995.

Capiluppi, A., Lago, P., and Morisio, M. "Characteristics of Open Source Projects," Seventh European Conference on Software Maintenance and Reengineering, 2003, pp. 317- 327.

Capiluppi, A., Morisio, M., and Lago, P. "Evolution of Understandability in OSS Projects," Proceedings of Eighth European Conference on Software Maintenance and Reengineering, 2004, pp. 58 - 66.

Capliuppi, A. "Models for Evolution of OSS Projects," International Conference on Software Maintenance, IEEE, 2003, pp. 65-74.

Cearly, D.W., Fenn, J., and Plummer, D.C. "Gartner's Position on the Five Hottest IT Topics and Trends in 2005," G00125868, Gartner Group.

Chiarini-Tremblay, M., Berndt, D.J., Foulis, P., and Luther, S. "Utilizing Text Mining Techniques to Identify Fall Related Injuries," Eleventh Americas Conference on Information Systems, Omaha, NE, 2005, pp. 1497-1504.

Chidamber, S., and Kemerer, C.F. "A Metric Suite for Object-Oriented Design," *IEEE Transactions on Software Engineering* (20:6), June 1994, pp 476-493.

Churcher, N.I., and Shepperd, M.I. "Comments on 'A Metrics Suite for Object-Oriented Design'," *IEEE Transactions on Software Engineering* (21:3), March 1995, pp 263-265.

Conway, M.E. "How Do Committees Invent?," *Datamation* (14:4), April 1968, pp 28-31.

Costanza, R., and Mageau, M. "What is a Healthy Ecosystem," *Aquatic Ecology* (33) 1999, pp 105-115.

Crowston, K., Annabi, H., and Howison, J. "Defining Open Source Project Success," International Conference of Information Systems, Seatle, WA, 2003.

Crowston, K., Annabi, H., Howison, J., and Masango, C. "Towards a Portfolio of FLOSS Project Success Measures. In Collaboration, Conflict and Control:," The 4th Workshop on Open Source Software Engineering, International Conference on Software Engineering (ICSE), Edinburgh, Scotland, 2004.

Crowston, K., and Scozzi, B. "Open Source Software Projects as Virtual Organizations," *IEEE Proceedings Software* (149:1) 2002, pp 3-17.

Deerwester, E.A. "Indexing by Latent Semantic Analysis," *Journal of the American Society for Information Science and Technology* (41:6) 1990, pp 391-401.

Deklava, S.M. "The Influence of the Information Systems Development Approach on Maintenance," *MIS Quarterly* (16:3), Sept. 1992, pp 355-372.

Delone, W.H., and McLean, E.R. "Information Systems Success: The Quest for the Dependent Variable," *Information Systems Research* (3:1), 1992, pp 60-95.

Dempsey, B.J., Weiss, D., Jones, P., and Greenberg, J. "Who is an Open Source Software Developer," *Communications of the ACM* (45:2) 2002, pp 67 - 72.

Eastwood, A. "Firm Fires Shots at Legacy Systems," *Computing Canada* (19:2) 1993, p 17.

Eick, S.G., Graves, T.L., Karr, A.K., Marron, J.S., and Mockus, A. "Does Code Decay? Assessing the Evidence from Change Management Data," *IEEE Transactions on Software Engineering* (27:1), January 2001, pp 1-12.

Erlikh, L. "Leveraging legacy system dollar for E-Business," *IEEE IT Professional* (2:3), May/June 2000, pp 17-23.

Fayyad, U.M., Piatetsky-Shapiro, G., and Smyth, P. (eds.) *From Data Mining to Knowledge Discovery: An Overview*. AAAI Press, Menlo Park, CA, 1996.

Feller, J., and Fitzgerald, B. *Understanding Open Source Software Development* Addison-Wiley, London, 2002.

Fenton, N.E., and Ohlsson, N. "Quantitative Analysis of Faults and Failures in a Complex Software System," *IEEE Transactions on Software Engineering* (26:8), August 2000, pp 797-814.

Fenton, N.E., and Pfleeger, S.L. *Software Metrics: A Rigorous Approach* Chapman & Hall, New York, NY, 1991.

Fernandez, G. *Data Mining Using SAS Applications* CRC Press, Boca Raton, Fl, 2003, p. 367.

Finkelstein, L., and Learning, M.S. "A Review of the Fundamental Concepts of Measurement," *Measurement* (2:1), Jan-Mar 1984, pp 25-34.

German, D.M. "Mining CVS Repositories, the softChange Experience," 1st International Workshop on Mining Software Repositories, 2004, pp. 17--21.

Godfrey, M., and Tu, Q. "Growth, Evolution and Structural Change in Open Source Software," IWPSE, ACM, Vienna Austria, 2001, pp. 103 - 106.

Greiner, S., Boskovic, B., Brest, J., and Zumer, V. "Security Issues in Information Systems Based on Open Source Technologies," EUROCON 2003. Computer as a Tool. The IEEE Region 8 , 2003, pp. 12-15.

Harter, D.E., Krishnan, M.S., and Slaughter, S.A. "Effects of Process Maturity on Quality, Cycle Time, and Effort in Software Product Development," *Management Science* (46:4), April 2000, pp 451-466.

Herbsleb, J., Carleton, A., Rozum, J., Seigel, J., and Zubrow, D. "Benefits of CMM-Based Software Process," CMU SEI-94-TR-13, Software Engineering Institute Carnegie Mellon Institute, Pittsburgh PA.,1994.

Herbsleb, J., Zubrow, D., Goldenson, D., Hayes, W., and Paulk, M. "Software Quality and the Capability Maturity Model," *Communications of the ACM* (40:6) 1997, pp 30-40.

Herbsleb, J.D., and Moitra, D. "Global Software Development," *IEEE Software* (18:2) 2001, pp 16-20.

Hertel, G., Niedner, and Herrmann, S. "Motivation of Software Developers in Open Source Projects: An Internet-Based Survey of Contributors to the Linux Kernel," *Research Policy* (32:7) 2003, pp 1159-1177.

Hippel, E., and Krogh, G. "Open Source Software and the "Private-Collective" Innovation Model: Issues for Organization Science," *Organization Science* (14:2), March-April 2003, pp 209-223.

Hoaglin, D.C., Mosteller, F., and Tukey, J.W. *Understanding Robust and Exploratory Data Analysis* John Wiley & Sons, 1983.

Hosmer, D.W., and Lemeshow, S. *Applied Logistic Regression*, (Second ed.) John Wiley & Son Inc, New York, NY, 2000.

Huff, S.L. "Information Systems Maintenance You and The Computer," *Business Quarterly* (55:2), September 1990, pp 30-33p.

Humphrey, W.S. *Managing the Software Process* Addison-Wesley, New York, NY,1989.

Huntley, C.L. "Organizational Learning in Open Source Software Projects: An Analysis of Debugging Data," *Engineering Management, IEEE Transactions on* (50:4), Nov 2003, pp 485 - 493.

IEEE-STD-1061 "IEEE Standard for a Software Quality Metrics Methodology," Institute of Electrical and Electronics Engineers, Inc., New York, NY, p. 88.

Jeffery, D.R. "A Software Development Productivity Model for MIS Environments," *Journal of Systems and Software* (7) 1987, pp 115-125.

Jensen, B.B., Lyngshede, S., and Sondergaard, D. "A Quality Definition for Open Source Software," Second Workshop on Software Quality at the 26th International Conference on Software Engineering, 2004, pp. 30-35.

Jensen, C., and Scacchi, W. "Collaboration, Leadership, Control, and Conflict Negotiation and the Netbeans.org Open Source Software Development Community," Proceedings of the 38th Annual Hawaii International Conference on System Sciences, Hawaii, HI, 2005, p. 196b.

Kemerer, C.F. "Measurement of Development Productivity," in: *Graduate School of Industrial Administration*, Carnegie Melon University, Pittsburgh PA, 1987.

Kemerer, C.F. "Software Complexity and Software Maintenance," *Annals of Software Engineering* (1), September 1995, pp 1-22.

Kemerer, C.F., and Slaughter, S.A. "Determinants of Software Maintenance Profiles: An Empirical Investigation," *Journal of Software Maintenance* (9) 1997, pp 235-251.

Kerstetter, J., Hamm, S., and Ante, S.E. "The Linux Uprising," in: *Business Week*, 2003, pp. 78-84.

Kitchenham, B.A., Pfleeger, S.L., and Fenton, N.E. "Towards a Framework for Software Measurement Validation," *IEEE Transactions on Software Engineering* (21:12) 1995, pp 929-943.

Kitchenham, B.A., Pfleeger, S.L., Pickard, L.M., Jones, P.W., Hoaglin, D.C., El Emam, K., and Rosenberg, J. "Preliminary Guidelines for Empirical Research in Software Engineering," *IEEE Transactions on Software Engineering* (28:8) 2002, pp 721 - 734.

Koch, S., and Schneider, G. "Effort, Cooperation and Coordination in an Open Source Software Project: GNOME," *Information Systems Journal* (12:1) 2002, pp 27-42.

Koru, A.G., and Tian, J. "Defect Handling in Medium and Large Open Source Projects," *Software, IEEE* (21:4) 2004, pp Pages:54 - 61.

Krishnamurthy, S. "Cave or Community? An Empirical Examination of 100 Mature Open Source Projects," *First Monday* (7:6) 2002.

Krishnan, M. "Cost and Quality Considerations in Software Product Management," in: *Graduate School of Industrial Administration*, Carnegie Mellon University, Pittsburgh, PA, 1996.

Krogh, G.v., Spaeth, S., and Lakhani, K.R. "Community, Joining, and Specialization in Open Source Software Innovation: A Case Study," *Research Policy* (32:7) 2003, pp 1217-1241.

Lederer, A.L., and Prasad, J. "A Causal Model for Software Cost Estimation Error," *IEEE Transactions on Software Engineering* (24:2), Feb. 1998, pp 137-147.

Lehman, M.M., and Ramil, J.F. "Rules and Tools for Software Evolution Planning and Management," *Annals of Software Engineering* (11) 2001, pp 15-44.

Lehman, M.M., and Ramil, J.F. "Software Evolution and Software Processes," *Annals of Software Engineering* (14) 2002, pp 275-309.

Lerner, J., and Tirole, J. "The Open Source Movement: Key Research Questions," *European Economic Review* (45) 2001, pp 819-826.

Lerner, J., and Tirole, J. "Some Simple Economics of Open Source," *Journal of Industrial Economics* (52) 2002, pp 197-234.

Lientz, B.P., and Swanson, E.B. *Software Maintenance Management* Addison-Wesley, Reading MA, 1980.

MacCormack, A. "Red Hat and the Linux Revolution," in: *Harvard Business School Case*, 9-600-009, 2002.

MacCormack, A., Rusnak, J., and Baldwin, C. "Exploring the Structure of Complex Software Design: An Empirical Study of Open Source and Proprietary Code," in: *Harvard Business School*, 2004.

Madey, G. "SourceForge.net Research Data Archive,"www.nd.edu/~OSS/Data/data.html, 2005.

Manning, C.D., and Schutze, H. *Foundations of Statistical Natural Language Processing* The MIT press, London, England, 2002.

Markus, M.L., Manville, B., and Agres, C.E. "Virtual Organization Design- Lessons Learnt from the Open Source Movement," *Sloan Management Review* (42:1) 2000, pp 13-26.

McCabe, T.J. "A Complexity Measure," *IEEE Transactions on Software Engineering* (2:4) 1976, pp 308-320.

Melton, A.C., Gustafson, D.A., Bieman, J.M., and Baker, A.L. "A Mathematical Perspective for Software Measures Research," *Software Engineering Journal* (5:5), September 1990, pp 246-225.

Menard, S.W. *Applied Logistic Regression Analysis* Sage Publications, Thousand Oaks, CA, 2002.

Moad, J. "Maintaining the Competitive Edge," *Datamation* (61-62:64) 1990, p 66.

Neter, J., Wasserman, W., and Kutner, M.H. *Applied Linear Regression Models.*, (4th ed.) McGraw-Hill/Irwin, Boston, MA, 2004, p. 701.

O'Reilly, T. "Lessons from open source software developmet," *Communications of the ACM* (42:4) 1999, pp 32-37.

Paulson, J.W., Succi, G., and Eberlein, A. "An Empirical Study of Open Source and Closed Source Software Products," *IEEE Transactions on Software Engineering* (30:4) 2004.

Pierce, J.R. *An Introduction to Information Theory: Symbols, Signals and Noise* Dover Publications, Inc, New York, NY, 1980.

Pinto, J.K., and Samuel. J. Mantel, J. "The Cause of Project Failure," *IEEE Transactions on Engineering Management* (37:4) 1990, pp 269-276.

Pinto, J.K., and Slevin, D.P. "Critical Factors in Successful Project Implementation," *IEEE Transactions on Engineering Management* (34) 1987, pp 22-27.

Pinto, J.K., and Slevin, D.P. "Project Success:Definitions and Measurement techniques.," *Project Management Journal* (19:3) 1988, pp 67-73.

Press, S.J. "The Role of Baysean and Frequentist Multivariate Modeling in Statistical Data," in: *Statistical Data Mining and Knowledge Discovery,* H. Bozdogen (ed.), Chapman & Hall /CRC, New York, NY, 2003.

Pressman, R. *Software Engineering: A Practitioner's Approach with Bonus Chapter on Agile Development*, (6th edition ed.) McGraw-Hill Science/Engineering/Math, New York, NY, 2004.

Raymond, E.S. "Cathedral and the Bazaar," http://www.tuxedo.org/~esr/writings/cathedral-bazaar/cathedral-bazaar, 2001.

Samoladas, I., Stamelos, I., Angelis, L., and Oikonomou, A. "Open Source Software Development Should Strive for Even Greater Code Maintainability," *Communications of the ACM* (47:10) 2004, pp 83-87.

Scacchi, W. "Understanding the Requirements for Developing Open Source Software Systems," *Software, IEE Proceedings* (149:1) 2002, pp 24-39.

Scacchi, W. "Free and Open Source Development Practices in the Game Community," *Software, IEEE* (21:1) 2004a, pp 59-66.

Scacchi, W. "Understanding Open Source Software Evolution: Applying , Breaking, and Rethinking the Laws of Software Evolution," in: *Software Evolution,* N.H. Madhavji, M.M. Lehman and J.F.R.a.D. Perry (eds.), John Wiley and Sons Inc,, New York, 2004b.

Schneidewind, N. "Methodology for Validating Software Metrics," *IEEE Transactions on Software Engineering* (17) 1992, pp 253-266.

Schneidewind, N. "Measuring and Evaluating Maintenance Process Using Reliability, Risk, and Test Metrics," *IEEE Transactions on Software Engineering* (25:6) 1999, pp 769-781.

Schonberg, E. "Measuring Success," *Communications of the ACM* (43:8) 2000, pp 53-57.

Seddon, P.B., Staples, S., Patnayakuni, R., and Bowtell, M. "Dimensions of Information Systems Success," *Communications of the Association for Information Systems* (2:20) 1999.

Sharpe, S., Haworth, D.A., and Hale, D. "Characteristics of Empirical Software Maintenance Studies: 1980-1989," *Journal of Software Maintenance: Research and Practice* (3:1) 1991, pp 1-15.

Simon, H.A. *The Sciences of the Artificial*, (2nd ed.) The MIT Press, Cambridge MA, 1994, p. 247.

Stamelos, I., Angelis, L., Oikonomou, A., and Bleris, G.L. "Code Quality Analysis in Open Source Software development," *Information Systems Journal* (12:1) 2002, pp 43-60.

Stewart, K.J. "OSS Project Success: From Internal Dynamics to External Impact," Proceedings of the 4th Workshop on Open Source Software Engineering, Edinburgh, Scotland, 2004.

Stewart, K.J., and Ammeter, A.P. "Impacts of License Choice and Organizational Sponsorship on User Interest and Development Activity in Open Source Software Projects," *Information Systems Research* (Forthcoming) 2006.

Stewart, K.J., and Gosian, S. "The Impact of Ideology on Effectiveness in Open Source Software Development Teams," *MIS Quarterly* (30:2) 2006, pp 1-23.

Swanson, E.B., and Beath, C.M. "Departmentalization in Software Development and Maintenance," in: *Software Project Management: Readings and Cases,* C.F. Kemerer (ed.), Irwin Book Team, Chicago IL, 1997, pp. 539-553.

Swanson, E.B., and Dans, E. "System Life Expectancy and the Maintenance Effort: Exploring Their Equilibration," *MIS Quarterly* (24:2), June 2000, pp 277-297.

Trochim, W.M.K. *The Research Methods Knowledge Base*, (2nd ed.) Atomic Dog Publishing, 1999, p. 376.

Ulanowicz, R.E. *Growth and Development: Ecosystem Phenomenology* Springer-Verlag, New York,NY, 1986.

Weyuker, E.J. "Evaluating Software Complexity Measures," *IEEE Transactions on Software Engineering* (14:9), Sept. 1988, pp 1359-1363.

Wheeler, D.A. "More than a Gigabuck: Estimating GNU/Linux's size," http://www.dwheeler. com/sloc/redhat71-v1/redhat71sloc.html, 2003.

Wu, M.-W., and Ling, Y.-D. "Open Source Software Development : An Overview," *IEEE Computing Practices* (34:6) 2001, pp 33-38.

Zelkowitz, M.V., Shaw, A.C., and Gannon, J.D. *Principles of Software Engineering and Design* Prentice Hall Inc., Englewood Cliffs, NJ, 1979.

APPENDIX A

Table A.1: Data set for the empirical validation of project viability

Project_id	Vigor	Resilience	Structure
3	0.07182	1.123596	0.3275
14	0.018953	0.095238	0.639032
120	0.010495	1.315789	0.623444
233	0.001002	0.518135	0.370997
255	0.082788	0.724638	0
1658	0.021783	3.333333	0.404132
4236	0.003235	0.588235	0.309184
6121	0.021101	0.833333	0.634511
11348	0.03609	1.587302	0
22307	0.059434	1.639344	0.157549
23067	0.276489	2.941176	1.048
29057	0.196218	2.5	0.289
33291	0.036764	3.225806	0.620386
37089	0.095877	5.882353	0.278701
54086	0.021403	6.25	0
57621	0.038067	0.485437	0.230064
71291	0.00238	8.333333	0.305391
84122	0.167054	4.347826	1.077942
88344	0.080719	10	0.77561
106696	0.088033	100	0.290028

Table A.2: Group Names for the Projects in Analysis

group_id	group_name
3	Mesa3D
14	NFS
120	Amanda Tape Backup
233	Gnomba
255	Ethereal
1658	libppd
4236	The Insidious Big Brother DataBase
6121	PhpWiki
11348	GNOME News Applet
22307	MySQL for Python
23067	phpMyAdmin
29057	Compiere ERP + CRM Business Solution
33291	jTDS - SQL Server and Sybase JDBC driver
37089	Open Media Lending Database
54086	Video4LinuxGrab
57621	PHPeclipse - PHP/SQL/HTML Eclipse-Plugin
71291	ESSTP
84122	Azureus - BitTorrent Client
88344	open ArchitectureWare
106696	Simple PHP Blog

APPENDIX B

Additional Results for the LR model of Vigor

Table B.1: Summary of Stepwise Selection

Step	Effect	DF	Score Chi-Square	Chi-Square Pr > ChiSq
1	Age	2	126.2512	<.0001
2	Bug_Count	1	194.8111	<.0001
3	File_Size_age	1	48.8165	<.0001
4	Cnt_Team	1	14.7037	0.0001
5	use_pm	1	13.0996	0.0003
6	Downloads	1	11.4321	0.0007
7	Aud* Bug_User	1	5.8314	0.0157
8	Msg_Cnt	1	4.7238	0.0297
9	Prj_Type	5	12.9602	0.0238
10	Bug_open	1	3.9932	0.0457
11	Prj_Type*use_mail	5	11.4942	0.0424

Table B.2: Frequency Distribution of Input Class Variables

Class	**Value**	**1**	**0**	**Total**
Age	01:low -749.08	184	395	579
	02:749.08-1377.04	96	708	804
	03:1377.04-high	57	533	590
audience1	1	157	760	917
	2	180	876	1056
clstdec2	1	19	99	118
	2	69	405	474
	3	68	272	340
	4	34	127	161
	5	56	385	441
	6	91	348	439
osi	0	257	1284	1541
	1	80	352	432
use_cvs	0	42	163	205
	1	295	1473	1768
use_forum	0	96	347	443
	1	241	1289	1530
use_mail	0	58	250	308
	1	279	1386	1665
use_news	0	35	123	158
	1	302	1513	1815
use_pm	0	101	334	435
	1	236	1302	1538

Table B.3: Descriptive Statistics for Continuous Variables

Variable	Vigor	Mean	Deviation	Minimum	Maximum
Bug_Count	1	0.027559	0.079889	0	0.693147
	0	0.004144	0.013196	0	0.216517
	Total	0.008143	0.036187	0	0.693147
Bugs_Open	1	0.032536	0.082577	0	0.693147
	0	0.010902	0.02414	0	0.30973
	Total	0.014597	0.041368	0	0.693147
Bugs_Rep_User	1	0.017673	0.065842	0	0.693147
	0	0.001905	0.009423	0	0.27368
	Total	0.004598	0.029112	0	0.693147
DwnLds_per_age	1	0.00637	0.039762	0.000001566	0.693147
	0	0.000597	0.005637	0	0.158411
	Total	0.001583	0.017334	0	0.693147
File_size_per_age	1	0.01852	0.064017	0.000008541	0.693147
	0	0.000988	0.004637	0	0.120045
	Total	0.003982	0.027562	0	0.693147
Msg_Cnt	1	0.01404	0.057237	0	0.693147
	0	0.002305	0.011641	0	0.175149
	Total	0.004309	0.026269	0	0.693147
Cnt_OS	1	2.700297	2.037396	0	11
	0	2.361858	1.709799	0	11
	Total	2.419665	1.774069	0	11
Cnt_Team	1	5.442136	6.793975	1	48
	0	3.047066	3.745364	1	66
	Total	3.456158	4.506444	1	66
Cnt_Forum	1	2.136499	0.855168	0	7
	0	2.097188	0.888784	0	8
	Total	2.103903	0.883044	0	8

Table B.4: Type 3 Analysis of Effects

Effect	**DF**	**Wald Chi-Square**	**Pr > ChiSq**
Age	2	125.8052	<.0001
Bug_Cnt	1	11.602	0.0007
Bug_Open	1	4.214	0.0401
File_size_age	1	70.8149	<.0001
Msg_Cnt	1	5.6911	0.0171
Aud*Bug_User	1	6.0876	0.0136
Prj_Type	5	15.3941	0.0088
Cnt_Team	1	14.2526	0.0002
Downloads	1	6.5237	0.0106
use_pm	1	10.1275	0.0015
Prj_Type*use_mail	5	11.2668	0.0463

Table B.5: Neural Network Weights for the Model of Vigor

N	Parameter	Estimate	Gradient Objective Function
1	age1_H11	-0.15958	0.001227
2	bftdays_H11	-0.10034	0.000909
3	bugcnt_H11	-0.25133	-0.00093
4	bugopen_H11	0.011565	-0.00064
5	bugreportedbynu_H11	0.545236	-0.00092
6	cntOS_H11	-0.27152	-0.00048
7	countusr_H11	-1.03598	-0.00063
8	downloads_H11	-0.34233	-0.00035
9	filesizeperage_H11	1.191523	-0.0009
10	formcnt_H11	-0.6407	0.000702
11	msgcnt_H11	-0.39956	-0.00022
12	age1_H12	0.363	0.014659
13	bftdays_H12	0.019895	0.007423
14	bugcnt_H12	1.192353	-0.0043
15	bugopen_H12	-0.65737	-0.00582
16	bugreportedbynu_H12	1.714098	-0.00281
17	cntOS_H12	-0.08592	0.006333
18	countusr_H12	-0.06469	-0.00996
19	downloads_H12	2.085465	-0.00181
20	filesizeperage_H12	4.885095	-0.00342
21	formcnt_H12	0.102449	0.003126
22	msgcnt_H12	0.19836	-0.00112
23	age1_H13	-0.46009	-0.00539
24	bftdays_H13	0.451149	0.00032

Table B.5: *Continued*

N	Parameter	Estimate	Gradient Objective Function
25	bugcnt_H13	-0.13137	0.001875
26	bugopen_H13	0.476282	0.00441
27	bugreportedbynu_H13	0.348048	0.002213
28	cntOS_H13	-0.78658	0.002557
29	countusr_H13	-0.98162	0.001373
30	downloads_H13	-0.02118	-0.00069
31	filesizeperage_H13	-0.97195	0.00139
32	formcnt_H13	-0.16572	0.000223
33	msgcnt_H13	-0.20365	-0.00305
34	osi0_H11	0.133397	-0.00107
35	use_cvs0_H11	-0.52265	0.000124
36	use_forum0_H11	-0.05251	0.000368
37	use_mail0_H11	0.463156	0.001218
38	use_news0_H11	0.646571	0.001088
39	use_pm0_H11	-0.12975	0.000362
40	osi0_H12	-0.07868	0.013723
41	use_cvs0_H12	0.053672	-0.02116
42	use_forum0_H12	0.001287	-0.02707
43	use_mail0_H12	0.060688	-0.01141
44	use_news0_H12	-0.10019	-0.02054
45	use_pm0_H12	-0.02032	-0.02781
46	osi0_H13	-0.55232	-0.00532
47	use_cvs0_H13	-0.04961	0.006308
48	use_forum0_H13	0.164206	0.005009
49	use_mail0_H13	0.540126	0.0048
50	use_news0_H13	-0.66791	0.00698
51	use_pm0_H13	-0.51902	0.005667
52	audience11_H11	0.182248	0.000558
53	clstdec21_H11	-0.57774	0.000478
54	clstdec22_H11	-1.21725	0.000761

Table B.5: *Continued*

N	Parameter	Estimate	Gradient Objective Function
55	clstdec23_H11	0.554575	0.000115
56	clstdec24_H11	0.555508	0.000135
57	clstdec25_H11	-0.43886	-1.3E-05
58	audience11_H12	0.113019	0.009852
59	clstdec21_H12	0.052073	-0.00433
60	clstdec22_H12	0.142524	0.003256
61	clstdec23_H12	-0.25367	-0.00322
62	clstdec24_H12	0.07382	-0.00021
63	clstdec25_H12	-0.03134	-0.00319
64	audience11_H13	0.83985	0.003969
65	clstdec21_H13	0.371032	0.001081
66	clstdec22_H13	0.56016	-0.00206
67	clstdec23_H13	-1.07177	0.000927
68	clstdec24_H13	-0.67624	0.001032
69	clstdec25_H13	0.368155	0.000564
70	BIAS_H11	-0.53743	-0.00133
71	BIAS_H12	0.902622	0.016607
72	BIAS_H13	0.262889	-0.00617
73	H11_vigor31	0.552582	-0.00387
74	H12_vigor31	3.386263	-0.00112
75	H13_vigor31	-1.02827	0.000548
76	BIAS_vigor31	-2.2215	0.002881

APPENDIX C

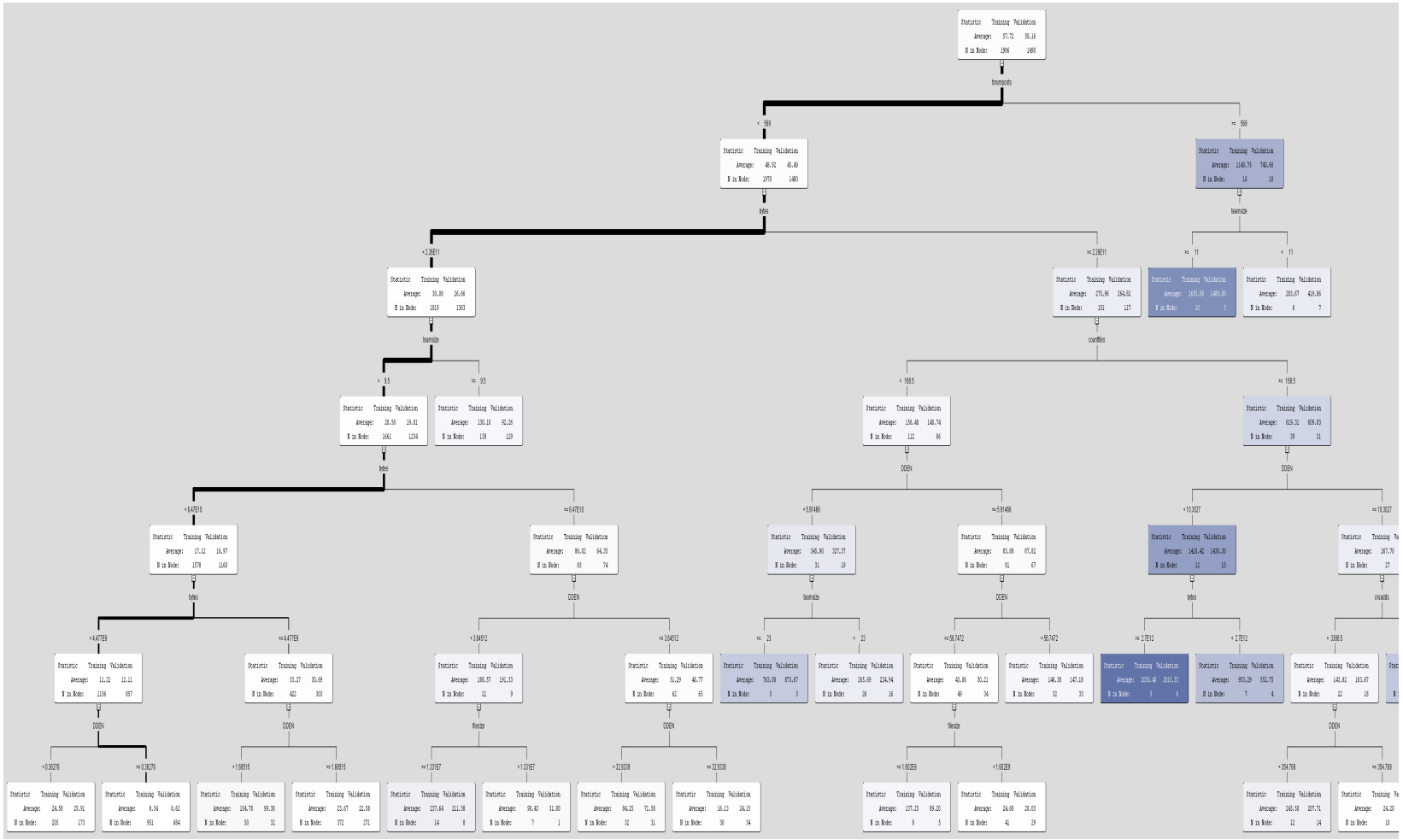

Figure C.1: Decision tree for the model of usage

187

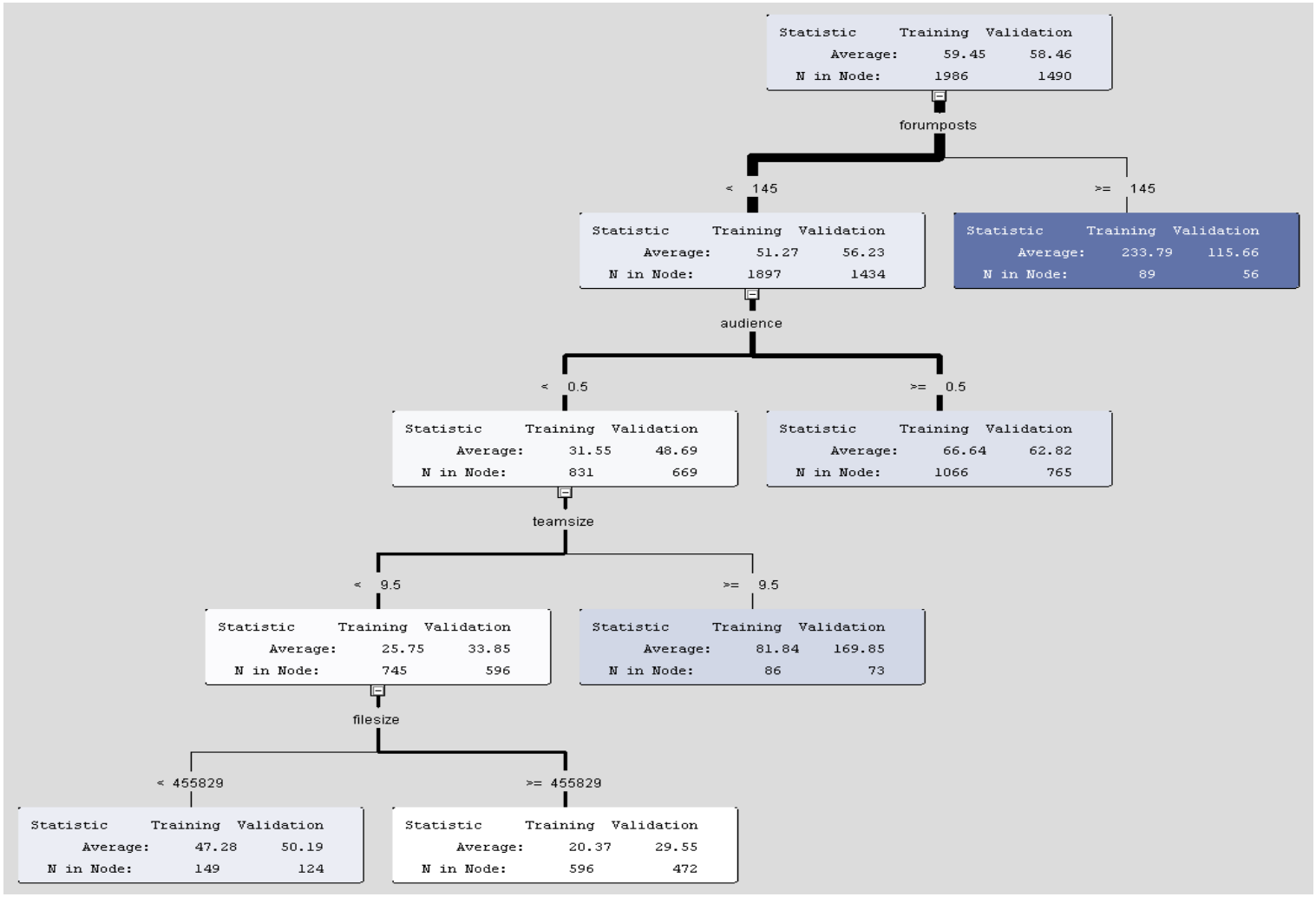

Figure C.1: Decision tree for the model of usefulness

VITA

Name:	Uzma Raja
Permanent Address:	Box 870223 Tuscaloosa, Alabama 35487-0223
Telephone:	(205) 348-7443
Email:	URaja@cba.ua.edu
Education:	B.Sc. Electrical Engineering, 1993 University of Engineering and Technology Lahore, Pakistan
	M.S. in Management Information Systems, 2002 Texas A&M University College Station, Texas
	Ph.D. in Information Systems, 2006 Texas A&M University College Station, Texas