12-13-2003

Assessment of Open-Source Software for High-Performance Computing

Gayatri Rapur

ASSESSMENT OF OPEN-SOURCE SOFTWARE FOR HIGH-PERFORMANCE COMPUTING

By

Gayatri Rapur

A Thesis
Submitted to the Faculty of
Mississippi State University
in Partial Fulfillment of the Requirements
for the Degree of Master of Science
in Computer Science
in the Department of Computer Science and Engineering

Mississippi State, Mississippi

December 2003

ASSESSMENT OF OPEN-SOURCE SOFTWARE FOR HIGH-PERFORMANCE COMPUTING

By

Gayatri Rapur

Approved:

Edward B. Allen Assistant Professor of Computer Science and Engineering (Major Professor)	Anthony Skjellum Professor of Computer Science and Engineering (Committee Member)
Thomas Philip Professor of Computer Science and Engineering (Committee Member)	Susan Bridges Professor of Computer Science and Engineering Graduate Coordinator Department of Computer Science

A. Wayne Bennett
Dean of the James Worth Bagley College of Engineering

Name: Gayatri Rapur

Date of Degree: December 13, 2003

Institution: Mississippi State University

Major Field: Computer Science

Major Professor: Dr. Edward B. Allen

Title of Study: ASSESSMENT OF OPEN-SOURCE SOFTWARE FOR HIGH-PERFORMANCE COMPUTING

Pages in Study: 121

Candidate for Degree of Master of Science

High quality software is a key component of various technology systems that are crucial to software producers, users, and society in general. Software application development today uses software from external sources, to achieve software implementation goals. Numerous methods, activities, and standards have been developed in order to realize quality software. Nevertheless, the pursuit for new methods of realizing and assuring quality in software is incessant. Researchers in the software engineering field are in pursuit of methods that can be on par with changing technology. Assessment of open-source software can be supported by a methodology that uses data from prior releases of a software product to predict the quality of a future release. The proposed methodology is validated using a case study of MPICH — an open-source software product from the field of high-performance computing. A quantitative model and a module-order model have been developed that can predict the modules that are expected to have code-churn and the amount of code-churn in

each module. Code-churn is defined as the amount of update activity that has been done to a software product in order to fix bugs. Further validation of the proposed methodology on other software and development of classification models for the quality factor code-churn are recommended as future work.

DEDICATION

This work is dedicated to my parents and my major professor.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my major professor and thesis director Dr. Edward Allen for his inspiration, constant encouragement, and guidance during the course of this research work and my studies. I sincerely thank Dr. Thomas Philip for his guidance throughout the thesis work. I am grateful to Dr. Anthony Skjellum for his guidance and support through out the thesis. I take this oppurtunity to thank Kumuran Rajaram for his valuable suggestions and reviews on this work. This work was supported in part by grant CCR-0132673 provided by the National Science Foundation.

I would like to thank Dr. Ewing Lusk of Argonne National Laboratory for providing us with MPICH source code and related information. I take this oppurtunity to thank the High-Performance Computing laboratory at Mississippi State University, for their valuable comments and suggestions during the course of the research. I thank Dr. Anna L. Dill for her review on the document. I especially thank the Empirical Software Engineering research group for all their valuable suggestions in making this work a success. Last but not the least I am grateful to my family and friends whose blessings, love, and faith always bring the best out in me.

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

TABLE Page

LIST OF FIGURES

LIST OF SYMBOLS, ABBREVIATIONS, AND NOMENCLATURE

AAE**:** Average Absolute Error

ARE**:** Average Relative Error

*FilComGlbNbr***:** Number of comment sections in the global scope of a file (thus excluding the comments inside routine or class scopes within the file)

*FilComGlbVol***:** Size in characters of all the comments in the global scope of a file (without considering the comments inside routine or class scopes within the file)

*FilComTotNbr***:** Total number of comment sections in the file (considering the comments inside routine or class scopes within the file)

*FilComTotVol***:** Size in characters of all the comments in the file (considering the comments inside routine or class scopes within the file)

*FilDecClaNbr***:** Number of classes declared within the file

*FilDecGncTypNbr***:** Number of generic classes (template class declaration) declared within the file

*FilDecGndTypTotNbr***:** Total number of generated classes (template class instances) declared within the file

*FilDecStruNbr***:** Number of struct types declared within the file

*FilDecObjExtNbr***:** Number of `extern` objects declared within the global scope of the file

*FilDefObjGlbNbr***:** Number of global variables/objects defined within the global scope of the file

*FilDefRtnNbr***:** Number of functions/routines defined within the file

*FilIncNbr***:** Total number of files included by the current file

*FilIncDirNbr***:** Total number of files directly included by the current file

*FilLnsNbr***:** Number of lines of the file

*FilLnsSkpSum***:** Number of full or partial lines skipped in the file, due to syntax errors

FilStxErrNbr: Number of syntax errors

MPICH: Message Passing Interface—Argonne-Mississippi State Model Implementation

MPI: Message Passing Interface

PCA: Principal Components Analysis

RtnArgXplSum: Sum of the explicit argument numbers (actual parameters) passed to other function by all the explicit function calls made in the routine

RtnCalXplNbr: Number of explicit function/method calls in the routine

RtnCastXplNbr: Number of explicit type casts in the routine

RtnComNbr: Number of comment sections in the routine scope (between the routine brackets $\{\ldots\}$)

RtnComVol: Size in characters of all the comments in the routine, without considering the comments within nested classes or routines

RtnCplCtlAvg: The mean control predicate complexity

RtnCplCtlMax: The maximal control predicate complexity

RtnCplCtlSum: Total (sum) complexity of the control predicates (test expressions) composing the decision and loop statements within the routine

RtnCplCycNbr: Cyclomatic number of the routine

RtnCplExeAvg: The mean executable statement complexity

RtnCplExeMax: The maximal executable statement complexity

RtnCplExeSum: Total (sum) complexity of the executable statements within the routine

RtnStmDecRtnNbr: Number of function/routine declaration statements within the routine

RtnStmDecObjNbr: Number of variable/object declaration statements in the routine

RtnStmDecPrmNbr: Number of parameters of the routine

RtnStmDecTypeNbr: Number of type/class declaration statements in the routine

RtnLnsNbr: Number of lines of the routine

RtnLnsSkpSum: Number of full or partial lines skipped in the routine, due to syntax errors

RtnScpNbr: Number of scopes within the scopes of the routine

RtnScpNstLvlAvg: Average nesting level of the scopes in the routine

RtnScpNstLvlMax: Maximal nesting level in the routine

RtnScpNstLvlSum: Sum of nesting level values for all scopes in the routine

RtnStmCtlBrkNbr: Number of break statements in the routine

RtnStmCtlCaseNbr: Number of C-language case-like statements in the routine

RtnStmCtlNbr: Number of control-flow statements in the routine

RtnStmCtlThwNbr: Number of throw (cast) statements in the routine

RtnStmCtlCtnNbr: Number of continue statements in the routine

RtnStmDecNbr: Number of declarative statements in the routine

RtnStmCtlDfltNbr: Number of default statements in the routine

RtnStmExeNbr: Number of executable statements in the routine

RtnStmCtlGotoNbr: Number of `goto` statements in the routine

RtnStmCtlIfNbr: Number of `if` statements in the routine

RtnLblNbr: Number of label statements in the routine

RtnStmCtlLopNbr: Number of loop statements in the routine

RtnStmNbr: Number of statements in the routine

RtnStmNstLvlAvg: Average nesting level of statements in the routine

RtnStmNstLvlSum: Sum of nesting level values of each statement in the routine

RtnStmCtlRetNbr: Number of return statements in the routine

RtnStmCtlSwiNbr: Number of C-language switch-like constructs in the routine

RtnStmXpdNbr: Expanded statement number: size (in number of statements) of the routine after expansion (limited loop unfolding operation)

RtnStxErrNbr: Number of syntax error that occurred while parsing the routine

CHAPTER I

INTRODUCTION

As indicated by the title, *Assessment of Open–Source Software for High-Performance Computing*, the goal of this thesis is to propose a methodology for assessment of open-source software that supports high-performance computing. "Open-source" software is typically defined as software whose source code is available to anyone ("open"), free or almost for free, with few license restrictions [2]. The high-performance computing field is defined as application of computing resources to achieve computational rates suitable for numerically intensive computations, such as advanced scientific and engineering modeling. The software process of development of open-source software is often unstructured and the quality assurance that is done in the process may not be rigorous.

Quality is a broad topic. There are many factors that affect quality. It is seen in a different light for different products. For software, factors such as correctness, maintainability, and integrity contribute to quality. Such factors cannot be measured directly from the software. Each of these factors in turn have indicators that can be measured from software. Let us consider the factor correctness, which is often quantified as the number of defects per line of code. In order to fix defects, changes are made to source code. In certain situa-

tions code is tuned to achieve optimization and performance goals. The amount of activity to change the source code in order to fix bugs and fine tuning is called “code-churn” [19].

Suppose a user has an open-source software system that needs an upgrade. If the user is considering an update, the question is,“Does the risk of bugs in a new version outweigh the benefits of its improvements?” The proposed methodology develops a model that can predict which modules are likely to be code-churn-prone and also predict the amount of code-churn for each module. Thus, the user can have prior identification of those modules that have a high probability of getting updated in the future release. Thus, the user can decide whether to upgrade now or not.

In this thesis, a methodology is proposed that uses data from the past releases of a software product to predict the amount of the quality factor “code-churn” for a future release of the software. Measurements of various software product attributes and the amount of reuse are collected for the software under study. Using the methodology, a model is built that can predict the amount of code-churn for a future release of the software.

As a case study, the proposed methodology is applied to Message Passing Interface–Chameleon (MPICH), which is an open-source software product for high-performance computing. The methodology was used with four versions of MPICH software released during the years 1994 and 1995. These were the first complete versions of MPICH. A model was built for the quality factor “code-churn” [18]. The model presents a rank-ordering of the most churn-prone modules for the “next” version of MPICH.

1.1 Hypothesis

The research hypothesis is as follows:

> By studying the data available in the form of past versions of MPICH, a methodology can be developed that will enable quality assessment of the software.

The hypothesis is supported by applying the proposed methodology using four versions of MPICH. All of the four versions are past releases, data from the versions APR/94, SEP/94, and 1.0.5 were used to build the model that can predict code-churn of a future release and data from versions 1.0.5 and 1.0.6 were used to validate the model. The results from this research provides software quality engineers and users with a methodology that can be used to obtain quality assessments of a such software products.

1.2 Research Questions

The following are the research questions that have been formulated for the current research.

1. How can the quality of the open-source software MPICH be assessed using changes in source code between consecutive versions?
2. What are software attributes derived from software metrics that are indicators of the quality of MPICH ?
3. Is a quality model feasible that gives an understanding of quality and is suitable for the open-source software product MPICH ?

The answers to the above research question will provide evidence for or against the stated hypothesis.

1.3 Relevance

The primary research objective is to develop ways to assess the quality of open-source software for high-performance computing. The current thesis proposes a methodology with which a quality model can be developed, which can provide insight into a future release. This research will help developers gauge quality more effectively and will enable prospective scientific and engineering users to evaluate whether or not to rely on the software [2]. More broadly, the long-term goal of this research is to contribute to research and education in empirical software engineering [2].

1.4 Overview

The document is divided into seven chapters. Chapter II provides a literature survey, done as background work for the thesis. Chapter III discusses the methodology and some measurement issues that were encountered when measuring the MPICH software. Chapter IV discusses the various tools and scripts that were used for measurement and data collection. Chapter V provides details on the subject of the case-study, the statistical procedures, the model, and the results of evaluation of the model. Chapter VI provides an evaluation of the research questions and some of the risks associated with the methodology and the model. Chapter VII presents the conclusions drawn from the work and makes some suggestions for future work.

CHAPTER II

RELATED WORK

This chapter discusses open-source software, giving a brief history of its inception, the pros and cons of open-source software that a user should have knowledge of before adopting, and factors that can make the open-source software more dependable. A part of the chapter introduces empirical research, software metrics, the current research, and its significance to the field of software engineering. Some recently researched techniques for reliability modeling and analysis of open-source software concludes the chapter.

2.1 An Overview of Open-Source Software

Open-source software can be defined as a software product that is open (available) to any user. The user has the freedom to make modifications of his/her choice to the software and to include it into projects. It implies freedom to redistribute the software under certain liberal policies [2]. The software be readable, modifiable, and redistributable, making application development an evolutionary process [2]. The conception of the open-source idea dates back to 1984, when Richard Stallman started the Free Software Foundation with the goal of ensuring that software be available to an end user for free modification and redistribution [30]. Later, Linus Torvalds used the GNU General Public License when

he released the first version of Linux software, making open-source software more popular and worth pondering. Open-source software moved into the enterprise market when the data-center managers grasped the value of cheap, malleable software [7].

In recent years the open-source movement has crept into the technological vendor market including IBM, Oracle, Sun Microsystems, and Apple Computer. Open-source software is on par with the commercial software in terms of functionality, capability, efficiency, speed of execution, organizational standards, and preferences [7]. The low cost associated with acquiring open-source software is a major reason for its popularity. In addition, the software is configurable, and the cost associated with application bug correction is reduced with a community proactively searching out and patching bugs [7].

Open-source software is developed by programmers from around the world who participate in an accepted process for open-source software development [2]. Since project developers are scattered across the globe, coordination of effort is important. Developers need to agree on a version control system in order to avoid development chaos. The most widely used version control models include `diff` and `patch`, Revision Control System, and Concurrent Versions System [30]. The conception of an open-source product was a result of personal itch than the need for a scientific or commercial application [7].

The software under study in the current research has both research and software development goals. The research goal is to narrow the gap between the programmer of a parallel computer and the performance of the hardware [12]. The software project goal is to promote adoption of the Message Passing Interface (MPI) standard by providing the users

with a free, high-performance implementation on a diversity of platforms, while aiding vendors in providing their own customized implementations [12].

The principle that forms the core of the open-source software movement is that the software source code should be available to the user so that both understanding and modifying the code is possible [8]. Some commonly used open-source licenses, which identify with that principle include the GNU General Public license (GPL), Lesser GPL, Berkeley Software Distribution, Mozilla Public License, and Netscape Public License [30]. The source license models fall mainly into three categories: Free, Copyleft, and GPL–Compatible. According to the Free license model the program can be freely modified and redistributed. Under the Copyleft license model the owner gives up intellectual property ("copyright") and private licensing rights. Other licenses that are GPL–Compatible are legally linked to GPL licensing [30].

Open-source software still needs to pass muster in terms of scalability, reliability, and security in order to gain market share and user confidence. There are many considerations that users are wary of when adopting an open-source software product in a given enterprise [7]. The main consideration that hinders user acceptance is the absence of dedicated resources for maintaining and enhancing the product, and the absence of a guaranteed level of support and trust, which is available for commercial software. For an open-source software candidate to be considered operationally robust and highly reliable, a user should pay attention to the usage history of the software, whether it has been operational in a large number of applications, and whether its performance has been evaluated and reviewed [7].

Other considerations include frequent updates, either for bugs or feature add–ons, and ease of use [7] . The potential of the open-source approach to contribute to aspects of dependability is also research issue. The term dependability covers reliability, security, safety, and availability [21].

Some of the research topics currently being investigated regarding open-source software are the quality and maintainability, the replicability and portability, software engineering processes and tool kits, the stability and sustainability of developer and user communities, and the viability and profitability of business models [9].

2.2 Empirical Research in Software Engineering

Empirical study in a broad sense involves an investigator gathering data and performing analysis to determine what the data mean for the purpose of discovering something unknown or testing a hypothesis [3]. The field of empirical studies in software engineering has only recently achieved significant recognition in the broader software-engineering community. Part of this new interest comes from practitioners, who have seen advances gained by adopting the research results [17].

Empirical studies can be classified as observational, formal experiments, and surveys [20]. Experiments are characterized by controlled behavior and execution. Surveys are indirect observations where information is collected through questionnaires and similar instruments [20]. Observational studies may be historical studies, where information is gathered from archives. This kind of empirical study has the advantage of being non-

intrusive. The observer does not need to be on-site or to communicate with people on active projects to gather information [20]. However, sometimes archival data may not be complete. The current research is a historical study of changes in source code in several released versions of MPICH.

The main element in any empirical study is the use of data to address a research question [17]. The data might define individual or group behavior, it might be quantitative or qualitative and it could be gathered over a short or long period of time [27]. The main obstacle is the difficulty in obtaining the data.

In any kind of empirical research either quantitative or qualitative, techniques for data collection can be adopted. Qualitative data is depicted more by words and pictures than numbers, whereas quantitative data mainly consists of numbers that can be directly and easily analyzed by calculations [27]. In the current study, changes in source code from consecutive versions of MPICH are being analyzed quantitatively. DATRIX is a tool that measures routine, class and file level metrics from preprocessed source code written in the C, C++, or Java languages [4]. Statistical analysis techniques are applied to obtain further information from the data [27].

Other data collection techniques include participant observation, interviews, and coding of observations [27], which are not relevant to the current situation. Other data analysis techniques include constant comparison method and cross-case analysis. Factors such as involvement of people and other human factors, the complexity of the research being conducted, and delivery of results in time to be relevant in a rapidly changing context make

empirical software engineering difficult [27]. Empirical research is always dependent on the maturity of knowledge being developed and tested. There will always be issues of internal and external validity, which are the basis for credible results [27].

It is important to know and keep in mind the ethical issues in empirical studies of software engineering. It is the obligation of an empirical researcher to be aware of the ethical issues that might come up during a research project. It is the responsibility of the empirical research unit to protect the participants in a study from any kind of harm. Singer and Vinson [29] discuss ethical issues raised by empirical research in software engineering and emphasize the need to deal with these issues. Research projects that include human subjects or information that can lead to identification of individuals are emphasized. Empirical research involving metrics, workplace studies, and process studies are a few good examples of such projects. Knowledge of the applicable ethical codes can help empirical researchers to protect their subjects and subjects' employers or managers. This will mitigate the risk of losing access to subjects, which is pivotal to empirical research. Singer and Vinson [29] provide four high-level ethical principles: informed consent, scientific value, contribution, and confidentiality. Examples illustrate these mentioned ethical principles. Informed consent is a primary ethical principle that consists of the following parts: disclosure, comprehension, competence, voluntariness, actual consent, and the right to withdraw from the experiment [29].

2.3 Software Metrics and Measurement

The following is background on measurement and its significance and relevance to software engineering and software metrics.

Measurement is acknowledged to have potential benefits to the field of software engineering, but it is rarely used on actual large-scale development products [24]. Measurement describes ways to characterize products, processes, or resources, and can be applied to assess the state of affairs, understand relationships among process elements, predict likely events, evaluate efficiency and effectiveness, and improve software development [24].

Software metrics can be defined as quantitatively determining the extent to which a software process, product, or project possesses a certain attribute [6]. Software metrics have become integral to improving software development.

The choice of the software metrics for a project depends on its definition of success. Depending on the goals, the right metrics must be chosen [11]. Goal-oriented measurement ensures such practicality, because it provides not only metrics definitions, but also the context of interpreting their values. Engineers and managers are then able to use them for making decisions [6]. Grady [11] suggests that the validity of inputs to any model for estimation or prediction is important and that the goals of any objective should be measurable. The definition of goals and objectives is important, because metrics must be defined with their intended use in mind. One of the important factors is cost-effectiveness, in the

sense that there is a positive return on investment. One should also ensure that changes to metrics have meaningful interpretations [11].

Metrics can be classified into process metrics, product metrics, and project metrics. The audiences for these metrics are software users, software managers, software engineers, software process engineers, and software quality assurance [6]. The needs of metrics users may include definition of metrics, training and consulting, automation including data collection tools, analysis, or feedback on development processes. The different levels in the organization at which metrics can be applied are company level, product level, project level, and component level [6].

The failure of some metrics programs can be attributed to the lack of clear definition of the purpose of the program, resistance because of the perception of it being a negative commentary on personnel performance, the data collection burden added to an already burdened staff, the failure to generate management action, and the withdrawal of management support, because the program seemed problematic and generated no-win situations [16],

Humphrey et al. [15], based on their experiment at Hughes Aircraft, state that all the significant data should be stored in a single central repository. A uniform data definition process should be established across projects. The process definitions should include key measures and analyses required at each major project milestone.

The selection and successful implementation of improvements depends on many variables such as the current process maturity, the available skills base, and business issues,

such as cost and schedule risk. Other barriers include difficulty getting started with the implementation of the improvements, staff turnover, lack of dedicated resources, and lack of management support. Also, difficulty finding time to work on software process improvement because of other extreme commitments to deliver customer products [23] is considered to be another barrier. In our current research none of the above problems are expected since the goal is not process improvement in an organization [23], instead we are using software metrics to develop a model for quality assessment of open source software.

Our goal is to analyze whether a proposed methodology identifies pointers for assessing quality of open-source high-performance computing software. Special attention will be given to product metrics such as the class, routine, and file metrics and information about how much has each source file changed over time in each version.

2.4 Statistical Modeling Techniques

Statistical models have become an important factor in evaluating complex systems [28]. Relevant information regarding such systems can only be obtained by means of statistical models. Complex systems include large computers, biological systems, and also weapon systems. Statistical models include models for measurement, empirical models, models for testing assumptions, and models for analysis of systems [28]. For the current research, we are interested in empirical models. Empirical models are models that fit a given set of data, rather than models derived from the mechanism under study [28]. The choice of estimation tools is based on the type of fundamental metrics being considered.

The credibility of the results from the model ultimately depends on the quality of the data [22]. For example, some of the concerns for estimating a regression model include whether the most important variables are included, and the model's predictive power [28].

To build a robust model, it is important that multivariate data be transformed into variables that are not correlated. Principal components analysis is a data reduction technique that transforms variables that are correlated into ones that are not correlated [19].

In an experimental work we may wish to find out how the values of one variable depends on other variables. For example, let a dependent variable be denoted by y and let one independent variable be denoted by x. If the relationship between x and y in our data appears to be reasonably linear we can postulate a linear equation for the line which best fits the sample data of the form $y = a + bx$ where a is the intercept and b is the slope of the line. The above equation is a regression line. It is an example of a linear statistical model [26]. In a multiple linear regression model there are several independent variables and one dependent variable.

$$y = a_0 + a_1x_1 + a_2x_2 + ... + a_nx_n \tag{2.1}$$

where $x_1, x_2, \ldots, x_n$ are the independent variables, y is the dependent variable and $a_1, a_2, \ldots a_n$ are parameter estimates of the model.

In this thesis, code-churn is the dependent variable and product metrics and reuse variables are the independent variables. A multiple linear regression model is built in which code-churn is a linear function of the product metrics and the reuse variables.

A module-order model [18] has an underlying quantitative software quality model that is used to predict the rank-order of modules according to a quality factor, such as reliability. With a predicted rank-order in hand, one can select as many modules from the of the list for reliability enhancement as resources will allow [18]. Khoshgoftaar and Allen [18] talk about module-order models and give methods to evaluate them. Case studies on two software systems are given, for which module-order models have been built to predict those modules that are fault prone and not fault-prone. In the first case study, the quality of fit of the underlying regression model was satisfactory and model was accurate. The module-order model for this case study was both useful and robust, even though the underlying model was less then ideal. For the second case-study the quality of fit for a linear regression was low, and the absolute and relative error for the test dataset indicated that the model was not accurate in predicting the code-churn. A module-order model was built based on this quantitative model. It was found that the module-order model's accuracy did not depend on how many modules would actually be given reliability enhancement. In this thesis, a module-order model is built to predict those modules that are code-churn prone.

2.5 Reliability Modeling of Freely Available Internet-Distributed Software

Reliability of software can be said to be the probability of fault-free operation of a computer software component for a specified time in a specified environment [10]. To determine the reliability of the software we need to have fault data. Reliability modeling includes using records of fault data occurring per testing hour to fit some parameters of

a numerical model. This model is used to predict reliability levels at a future point in time. Acquiring the fault data are feasible for commercial software, but for software that is available freely on the Internet and open-source, obtaining the fault data may not be directly feasible. The technique for modeling reliability data requires fault metrics data. The technique described by Fink [10] in his paper, shows that fault data can be determined from non-traditional sources such as email message archives. However, in the current research, fault data are not relevant. Data in the form of source programs are available and the changes in source code between consecutive versions are being measured instead of using email message archives.

CHAPTER III

METHODOLOGY

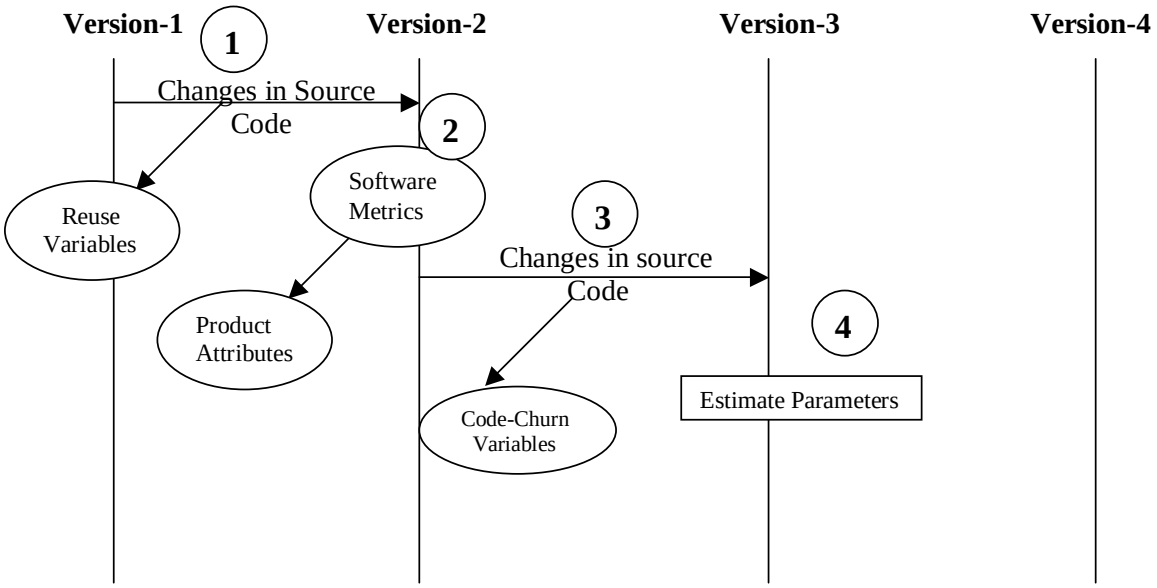
This chapter presents the methodology that has been adopted for the research. The organization of the chapter is as follows. Section 3.1 gives details of the methodology for the research. Section 3.2 discusses some of the measurement issues that are encountered on application of the methodology, during the data collection step. The details on how these issues can be dealt with are given.

3.1 Methodology

The methodology proposed in this thesis is based on the idea that from the information obtained from the prior releases of a software product a quality model can be built. The predictions from the quality model can provide useful insights regarding a future version. For the methodology we need four versions of a software product. The pictorial depiction of the methodology is represented in Figure 3.1 and Figure 3.2.

The methodology can be split into the following steps: Data are collected, statistical analysis is performed, and the model is evaluated.

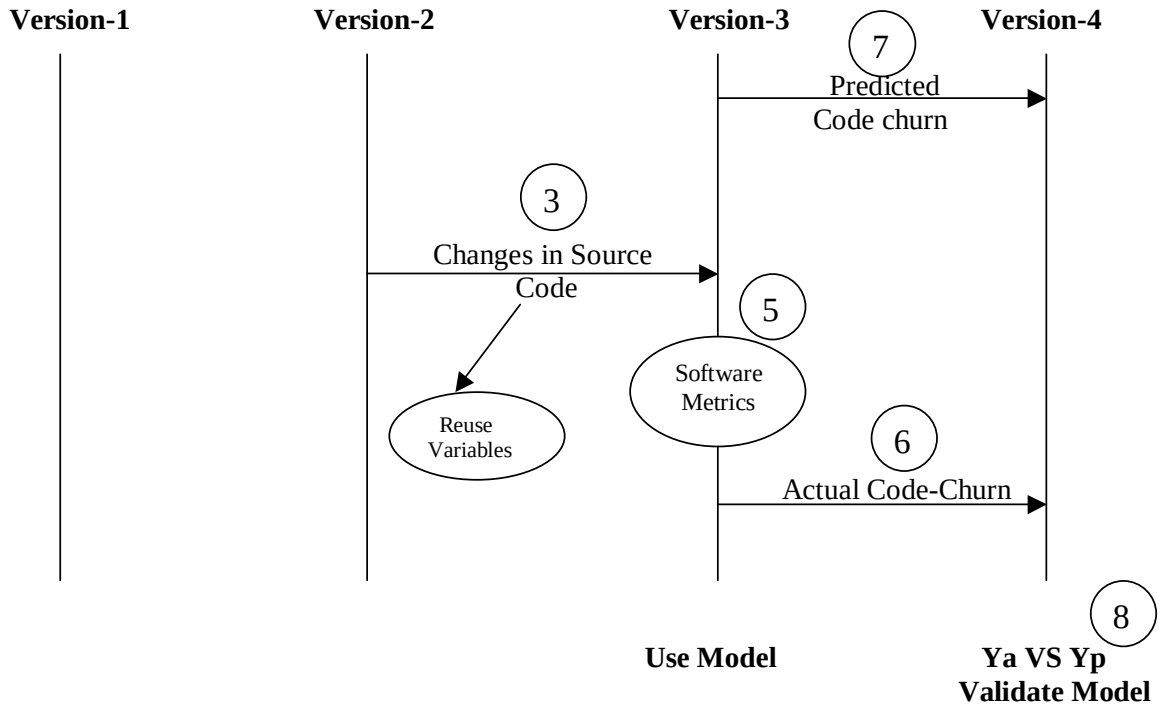
1. Data collection
    (a) Measure the changes in source code of the first two versions to obtain reuse variables. Refer to step 1 in Figure 3.1 The measurement of reuse variables is further discussed in the Section 3.2.



Build Model

$$y_i = a_0 + a_1 x_{i1} + a_2 x_{i2} + a_3 x_{i3} + \cdots\cdots a_p x_{ip}$$

Figure 3.1 Build Model



$$y_i = a_0 + a_1 x_{i1} + a_2 x_{i2} + a_3 x_{i3} + \ldots\ldots a_p x_{ip}$$

Figure 3.2 Use and Validate Model

(b) Using a software metrics analyzer such as DATRIX [4], obtain the routine and file metrics from the second version, shown as step 2 in Figure 3.1.

(c) Measure the changes in source code of the second and third versions from which the code-churn variables are obtained, shown as step 3 in Figure 3.1. The code-churn variables are summed to obtain the total code-churn, the dependent variable of the quality model.

(d) In a similar manner, reuse measurements are obtained from the second and third versions shown as step 3 in Figure 3.2. Product attributes are obtained from the third version shown as step 5 in Figure 3.2. Code-churn measurements are obtained from third and fourth versions, shown as step 6 in Figure 3.2.

(e) Assemble fit data and test data. Fit data consists of data to be used for calibration of the model's parameters and test data will be used for evaluation of the model. The fit dataset comes from the first, second, and third versions. The test dataset comes from the third and fourth versions.

2. Perform statistical analysis, data reduction, and transformation techniques

(a) Descriptive Statistics: Calculate the mean, standard deviation, maximum and minimum value of each raw metric, reuse variable and code-churn measurements in the datasets

(b) Correlation coefficients: Calculate details on those metrics that are highly correlated.

(c) Principal Components Analysis: Reduce the dimensionality of the data.

(d) Regression Fitting: Uses Least Square method for parameter estimation, this method is used at step 6 in the Figure 3.1. Use the stepwise procedure for the selection of independent variables in the model. In Figure 3.1 and Figure 3.2 the equation at the bottom, illustrates the form of the linear statistical model that was built.

3. Validation

(a) Use the model on a subsequent historical release, shown as step 7 in Figure 3.2. This simulates using the model on a future release.

(b) Evaluate the model by calculating error values. The error percentages of the model are obtained by comparing the actual and predicted values of code-churn.

In the equation given at bottom of the Figure 3.1 and Figure 3.2, y_i is dependent variable "code-churn" and $x_{i1}, x_{i2}, x_{i3} \ldots x_{ip}$ are the independent variables i.e, product attributes

and reuse measurements. $a_1, a_2, a_3 \ldots a_p$ are the parameter estimates. Parameter estimates are calculated from the known values of dependent and independent variables. When the methodology is applied to other software, it is not necessary that exactly the same techniques be used. Other techniques and statistical tools that are suitable for the data at hand should be used.

3.2 Measurement Issues

This section discusses measurement issues that were encountered while measuring the independent variables, that is product attributes and reuse measurements, and the dependent variable, that is code-churn. The reuse variables and code-churn were measured from changes in source code. In order to obtain the changes in source code, any tool or utility that can give the number of changed, new, and deleted lines between a pair of files can be used, for example `diff`. Product metrics are collected using a tool that can measure a broad range of attributes of source code.

3.2.1 Reuse Measurement

In this section, the measurement of reuse variables and code-churn from the changes in source code of the source files of a software product is discussed. For the purpose of measuring the changes in source code in this thesis, the CVS `diff` utility was used. The following are the reuse variables that can be measured from the changes in source code:

1. *Rchange*: The total number of lines in the second file (current release) that have been changed from the prior release.

2. *Rnew*: The total number of lines that are present in the second file (current release) and are not in the first file (prior release).

3. *Rdeleted*: The total number of lines that are not present in the second file, but were present in the first file.

The following list here summarizes the issues that were encountered during the meausrement of changes in source code. For each issue ecountered, an explanation is also given regarding how it was dealt with in this thesis:

1. When a pair of files are given as input to the `diff` utility, it gives the changes in source in the first file and also the second file. In this thesis, changes in source code from the second file were counted.

2. Every source file contains header information given by the configuration management system, such as the path of the file, the date and time of entry into the configuration management system, given by the configuration management system keywords such as `Header`, `Date`, `Id`. `diff` gives changes in lines differing on this information among a pair of files. This inclusion of configuration management system header data is true for all files that are stored in it. In this thesis, we did not adjust the measurement for this. Changes in these lines were also counted, which resulted in slight inflation of the total number of changed lines.

3. Changes in comment lines in the source files were also counted.

4. Changes in blank lines, and inclusion of new blank lines were also counted.

5. A source file present in a prior release and not present in the next consecutive version was considered deleted. All the lines in that file were counted as deleted lines.

6. A source file introduced in the current release and not present in the prior release was considered new. All the lines in that file were counted as new lines.

7. For a file present in a prior release and deleted from the next consecutive version. Lines of the file are counted as deleted. The code-churn for that file was zero.

8. A file not present in a prior release and present in the consecutive version was considered as new and all the lines in the file are counted as new. The reuse variables for that file are zero.

3.2.2 Code Churn

Code-churn is the measure of change done to source code in order to fix bugs or to fine tune the product's performance. It is a surrogate measure for the amount of defects. Code-churn variables were measured the same way as reuse variables. The same issues that came up during measurement of changes in source code for obtaining reuse variables were encountered here. These issues were handled in the same way. The following are the code-churn variables that were measured:

1. *Churnchange*: The total number of lines in the second file (current release) that have been changed.
2. *Churnnew*: The total number of lines that are present in the second file (current release) and are not in the first file (prior release).
3. *Churndeleted*: The total number of lines that are not present in the second file, but were present in the first file.

The release notes of a version should be studied in order to clear the ambiguity on the issue of whether code-churn is because of functional enhancement or bug fixing and fine tuning. The code-churn is the sum of the total number of changed lines, added lines and deleted lines.

$$code\text{-}churn = churnchange + churnnew + churndeleted \tag{3.1}$$

It should be noted that same set of changes in source code measured between the second and third versions, are present in the fit and test data. These values are used as churn variables (fit dataset) when building the model. The same values are used as reuse

variables in the test dataset for validation of the model. Chapter V gives more details on the fit and test sets in the case-study.

3.2.3 Aggregation of Product Metrics

This section discusses the issues that were ecountered during the product attribute measurement and how they were handled in this thesis. During the implementation of a software product, modularity is one of the primary design concerns. Consequently, there will be files with single and multiple routine definitions. When a software metrics analyzer is used to collect metrics from such source files, it will collect file level metrics and routine level metrics. For all the files containing a single routine, the tool will collect one single set of file level metrics and a single set of routine level metrics. For all those files containing multiple routine definitions, the tool will collect one set of file metrics and as many sets of routine metrics as the number of routines defined in the file. In order to make the collected data consistent we had the following alternatives:

1. Disaggregation of metrics
2. Aggregation of metrics

In disaggregation, the file containing multiple routine definitions could be split into multiple files. For instance, if a file had four routines defined in it then the file could be split into four different files. For such a case, the measurement of reuse variables and churn variables would be time consuming and would entail understanding the underlying code.

In aggregation, the metrics analyzer collects as many sets of routine metrics as there are routines defined in the file. Metrics that are counts or a sum of some source code attribute are added to obtain one value for the file. Metrics which define the maximum value of a source attribute measured over all the routines in the file is considered. Similarly, for metrics defining the average, the average value of the metric calculated over all the routines is computed. In this thesis, aggregation was performed over the routine metrics for those files that had multiple routine definitions. For the purpose of aggregation of metrics, a script can be used. The script that was used for this purpose is given in the Appendix.

In this chapter we discussed the methodology and the measurement issues that can come up when the methodology is applied. We also discussed how to handle issues during reuse and code-churn measurement and product attribute measurement.

CHAPTER IV

TOOLS

This chapter describes the various tools and scripts we have used at various stages in order to collect and format data. The chapter is organized as follows. Section 4.1 gives a discussion on the configuration management system that we have used in the study. Section 4.2 gives details on the metric tool that we have used to collect the product attributes. Section 4.3 is a discussion on the various scripts that we have used to aid in the process of data collection. The final section 4.4 gives a brief description of the statistical analysis tool we have chosen in this thesis.

4.1 Concurrent Version Management Tool

Concurrent Version Management System (CVS) [5] is used in many software projects to record and maintain a history of source files. CVS enables easy retrieval, maintenance, and recording of source files. It provides many commands to aid in configuration management. Taking advantage of the features of CVS, we setup the CVS repository in order to store the selected four versions of MPICH. CVS also supports the `diff` utility, which gives the changes in source code between two files. CVS was also utilized to store all the scripts that have been developed in the study.

The following is the `diff` command used to obtain the changes between a pair of files: `diff -c -C0 file1 file2`. The `-c` option gives the output in context format. The context format produces a listing of differences with three lines of context. With this option, output begins with identification of the files involved and their creation dates, then each change is separated by a line with a dozen asterisks. The lines removed from `file1` are marked with an initial hyphen (-); those added to `file2` are marked with an intial plus (+). Lines that are changed from one file to the other are marked with an intial exclamation point (!). For the purposes of this thesis we have set the number of context lines to be output to zero, using option `C0`. This resulted in a listing of just the lines that have changed in both the files to be output. An example of the context output format of `diff` with the number of lines of context to be output set to zero is given in the Appendix.

4.2 DATRIX Tool

DATRIX is a proprietary tool of Bell Canada. It is a code assessment tool. Datrix can obtain product metrics for C, C++, and Java programming languages. We are using this tool to obtain the source code file and routine level metrics from C source code files of MPICH. In order to obtain the metrics, the tool uses the preprocessed file `*.i` of the source code `*.c`. The set of the product attributes that are collected by DATRIX from the source code at the routine and file level is given in the glossary [4].

4.3 Scripts

A script performs a set of repetitive steps over different inputs. Scripts are introduced at those places where certain steps are to be performed iteratively, thereby conserving on time and reducing human error. In this thesis, scripts have been written to aid in the process of data collection, data formatting, and aggregation of raw metrics. We are using the Practical Extraction and Reporting Language (PERL) [1] and shell scripting language. Figure 4.1 gives the pictoral depiction of the steps to obtain the total data file which contains formatted, aggregated metrics of all the directories under study along with the reuse variables and the code-churn variables.

The left hand side of Figure 4.1 gives the steps involved to collect product attributes and the right hand side gives the scripts that were involved to obtain the reuse variables. Starting from the left hand side, the first step is to obtain product metrics. The source code `*.c` is preprocessed using the gcc compiler for C, and then DATRIX is run separately on the output obtained from the preprocessing step. In the Figure 4.1, this step is represented by the box “Collect Metrics”.

The next step is to format the obtained routine metrics according to a format compatible with SAS. The following is an example of the comma-separated format of a file containing routine metrics.

```
"FILE NAME","FUNCTIONAME","RtnArgXplSum","RtnCalXplNbr","RtnCastXplNbr"
"allgather.c","MPI_Allgather__FPviP13MPIR,  22.000,5.000,   0.000,
"allgatherv.c","MPI_Allgatherv__FPviP13MPIR,  23.000,5.000,   0.000,
"allreduce.c","MPI_Allreduce__FPvPviP13MPIRM,  24.000, 5.000,   0.000,
"alltoall.c","MPI_Alltoall__FPviP13MPIR,  30.000,   7.000,4.000,
```

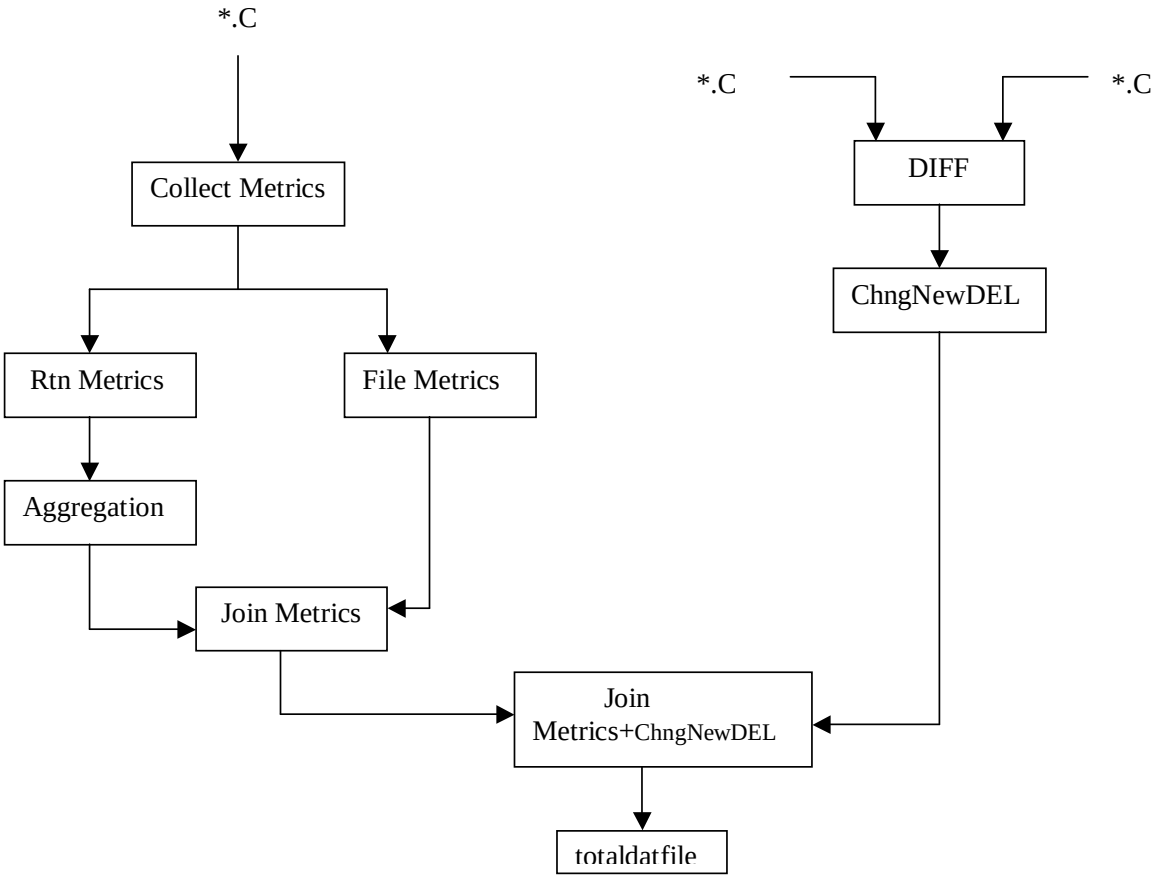


Figure 4.1 Flow Diagram for Data Collection

All the strings should be enclosed in quotes. Every numeric and string value should be separated by a comma. The first line consists of the column names (fields) of the data collected by DATRIX. SAS interprets a number separated by commas as one individual reading. This step of formatting the routine and file metrics is achieved by the scripts `RtnMetrics` and `FileMetrics`, in Figure 4.1.

The next step is to aggregate the metrics. This is achieved by the script `Aggregation` in Figure 4.1. The need for aggregation was discussed in the Section 3.2.3. The step of formatting and aggregating the metrics is done separately for the routine and file metrics. Using the `join` command of Unix, we join the two files to obtain one file containing the formatted and aggregated file and routine metrics.

The right hand side of Figure 4.1 gives the steps for collecting the reuse and churn variables. The first step is to find the changes in source code between two source files using the CVS `diff`. The script `ChngNewDel` counts the number of changed lines, added lines from the second file, and deleted lines from the first file. From the script, we find the reuse variables. The final step is to join the two files containing the raw metrics and the file containing the reuse variables, which gives the total data file. The code of the above mentioned scripts are in the Appendix.

4.4 Statistical Analysis Tool

Statistical Analysis System (SAS) [14] is a statistical analysis package that we have chosen for the purpose of running different statistical procedures on the collected raw

data. SAS provides various procedures that can be run on that raw data to obtain results for further analysis. The following are the procedures that we were interested in from SAS:

1. `Proc Means`
2. `Proc Corr`
3. `Proc Factor`
4. `Proc Reg`
5. `Proc Score`
6. `Proc Plot`

Details on the results obtained from the above procedures are discussed in Chapter V. In addition to the above statistical procedures, SAS also provides data modification statements. We have used the SAS `data` statement to split up the total data file which contains both the C source file data and the header file data into data sets containing only C source file data and only header file data.

CHAPTER V

CASE STUDY

This chapter presents details on MPICH and the results obtained from the statistical analysis step. The organization of the chapter is as following. Section 5.1 gives details on the case study. Section 5.2 gives the details of the directories of MPICH from which data were collected. Section 5.3 gives summary statistics of the fit dataset and observations from these results. Section 5.4 gives the principal components analysis and regression results. Section 5.5 gives details on validation of the model.

5.1 Subject of Case Study

MPICH is an open-source, portable implementation of the Message Passing Interface (MPI) standard. The implementation of MPICH began at the same time as the MPI definition process itself, in order to provide early feedback on decisions being made by the MPI Forum. The goal of the implementation was to include all the systems capable of supporting the message-passing model [13].

MPICH came into being quickly, because it could build on stable code from existing systems. These systems had already prefigured in various ways the portability, performance and some of the other features of MPICH. The following are the precursor systems

on which the MPICH was built. P4 is a third generation parallel-programming library, including both message-passing and shared memory components. It is portable to many parallel computing environments including heterogeneous networks. Chameleon is a high-performance portability package for message passing on parallel supercomputers. Zipcode is a portable system for writing scalable libraries. It contributed several concepts to the design of the MPI standard, such as contexts, groups, and communicators [13].

The following is the configuration of MPICH that we are interested in for the current study:

1. `ch_p4` device, distributed memory architecture, Unix version.
2. The following versions have been chosen for the current study: APR/94, SEP/94, 1.0.5, and 1.0.6.

5.2 Data Collection Details

Table 5.1 gives the details on the data collected and whether they constituted the fit or the test dataset.

From the release notes of the above versions we concluded that most of changes the done to the source code because of bug fixing and fine tuning the functions. No major enhancements in functionality were done to directories under study. The following are the names of the seven directories belonging to MPICH from which the data for the study was collected. The lines of code of all the files in the seven directories summed to $16,691 LOC$. LOC is the acronym for lines of code.

1. `Coll`

Table 5.1 Versions, Datasets and Variables.

Version	Measurement	Data Set	Variables
APR/94			
SEP/94	Changes since 1.0.1.1	Fit	Reuse
SEP/94	Software metrics	Fit	Product attributes
1.0.5	Changes since SEP/94	Fit	Code churn
		Test	Reuse.
1.0.5	Software Metrics	Test	Product attributes
1.0.6	Changes since 1.0.5	Test	Code churn.

2. `Context`
3. `Env`
4. `Pt2pt`
5. `Topol`
6. `Util`
7. `Dmpi`

The product metrics were measured from the source files in these seven directories of the versions of MPICH shown in Table 5.1. The total data file contained the file metrics, routine metrics collected from the `*.C` source files and header files, and the reuse and churn variables. The total data file was then split into two datasets containing observations from C source files and header files separately. The following section gives a detailed description of each statistical procedure run separately on the total dataset, the dataset containing only C observations and the dataset containing only header observations.

5.3 Descriptive Statistics

The `Means` procedure gives the mean, minimum, maximum, and standard deviation of each metric collected. The SAS code of the procedure that was used to obtain the summary statistics is given in the Appendix. Table 5.2 shows the descriptive statistics of the raw metrics for the total data file from the fit dataset. The following are the observations that are drawn from the table:

- There were 175 files, including the `*.c` and `*.h` files in the total dataset, from which the product metrics, reuse variables and code-churn variables where collected.
- The file metrics *FilComGlbNbr*, *FilComGlbVol*, *FilComTotNbr*, *FilComTotVol*, *FilDecClaNbr*, *FilDecGncTypNbr*, *FilDecGndTypTotNbr*, *FilLnsSkpSum*, and *FilStxErrNbr* had mean and standard deviation of zero.
- The routine metrics *RtnComNbr*, *RtnComVol*, *RtnLblNbr*, *RtnLnsSkpSum*, *RtnStmCtlGotoNbr*, *RtnStmCtlThwNbr*, and *RtnStxErrNbr* had mean and standard deviation of zero.
- The above metrics were eliminated from further analysis, because they were constant over all files measured.

Table 5.3 shows the descriptive statistics of raw metrics collected from C source files belonging to the fit dataset. The following are the observations that are drawn from the table:

- There were 158 `*.c` files, which constitute the C dataset, from which the product metrics, reuse variables and code-churn variables where collected.
- The file metrics *FilComGlbNbr*, *FilComGlbVol*, *FilComTotNbr*, *FilComTotVol*, *FilDecClaNbr*, *FilDecGncTypNbr*, *FilDecGndTypTotNbr*, *FilLnsSkpSum*, and *FilStxErrNbr* had mean and standard deviation of zero.
- The routine metrics *RtnComNbr*, *RtnComVol*, *RtnLblNbr*, *RtnLnsSkpSum*, *RtnStmCtlGotoNbr*, *RtnStmCtlThwNbr*, and *RtnStxErrNbr* had mean and standard deviation of zero.

Table 5.2 Summary Statistics of Total Dataset

Variable	Mean	StdDev	Minimum	Maximum
FilComGlbNbr	0.000	0.000	0	0.000
FilComGlbVol	0.000	0.000	0	0.000
FilComTotNbr	0.000	0.000	0	0.000
FilComTotVol	0.000	0.000	0	0.000
FilDecClaNbr	0.000	0.000	0	0.000
FilDecGncTypNbr	0.000	0.000	0	0.000
FilDecGndTypTotNbr	0.000	0.000	0	0.000
FilDecObjExtNbr	0.429	4.10	0	52.000
FilDecStruNbr	0.212	1.061	0	8.000
FilDefObjGlbNbr	0.932	5.801	0	77.000
FilDefRtnNbr	1.457	2.343	0	17.000
FilIncDirNbr	1.490	0.857	0	6.000
FilIncNbr	17.190	7.249	0	26.000
FilLnsNbr	95.383	139.291	0	1166.000
FilLnsSkpSum	0.000	0.000	0	0.000
FilStxErrNbr	0.000	0.000	0	0.000
RtnArgXplSum	19.029	29.123	0	273.000
RtnCalXplNbr	6.663	12.552	0	113.000
RtnCastXplNbr	4.131	16.493	0	208.000
RtnComNbr	0.000	0.000	0	0.000
RtnComVol	0.000	0.000	0	0.000
RtnCplCtlAvg	11.204	11.120	0	47.000
RtnCplCtlMax	23.263	23.530	0	84.000
RtnCplCtlSum	66.235	118.877	0	1226.000
RtnCplCycNbr	8.903	23.810	0	278.000
RtnCplExeAvg	8.250	5.354	0	39.250
RtnCplExeMax	16.777	13.662	0	63.000
RtnCplExeSum	178.860	420.327	0	4417.000
RtnLblNbr	0.000	0.000	0	0.000
RtnLnsNbr	59.983	113.089	0	1049.000
RtnLnsSkpSum	0.000	0.000	0	0.000
RtnScpNbr	10.240	27.223	0	302.000
RtnScpNstLvlAvg	1.783	1.104	0	5.685

The number of source files was 175

Table 5.2 Summary Statistics of Total Dataset (continued)

Variable	Mean	StdDev	Minimum	Maximum
RtnScpNstLvlMax	2.732	2.325	0	13.000
RtnScpNstLvlSum	32.423	117.412	0	1264.000
RtnStmCtlBrkNbr	1.150	9.214	0	116.000
RtnStmCtlCaseNbr	0.943	8.282	0	104.000
RtnStmCtlCtnNbr	0.040	0.224	0	2.000
RtnStmCtlDfltNbr	0.10	1.068	0	14.000
RtnStmCtlGotoNbr	0.000	0.000	0	0.000
RtnStmCtlIfNbr	4.743	8.056	0	56.000
RtnStmCtlLopNbr	1.657	8.352	0	104.000
RtnStmCtlNbr	10.394	24.104	0	278.000
RtnStmCtlRetNbr	2.686	2.789	0	19.000
RtnStmCtlSwiNbr	0.120	1.090	0	14.000
RtnStmCtlThwNbr	0.000	0.000	0	0.000
RtnStmDecNbr	10.570	23.485	0	284.000
RtnStmDecObjNbr	10.550	23.470	0	284.000
RtnStmDecPrmNbr	4.980	6.905	0	51.000
RtnStmDecRtnNbr	0.011	0.106	0	1.000
RtnStmDecTypeNbr	0.011	0.151	0	2.000
RtnStmExeNbr	17.640	40.648	0	378.000
RtnStmNbr	38.606	85.106	0	940.000
RtnStmNstLvlAvg	1.103	0.786	0	3.545
RtnStmNstLvlSum	77.470	279.303	0	3332.000
RtnStxErrNbr	0.000	0.000	0	0.000
RtnStmXpdNbr	49.62	119.86	0	1238.000
Rchange	19.766	52.342	0	612.000
Rnew	14.343	56.470	0	669.000
Rdeleted	7.690	42.420	0	517.000
churnchange	5.863	14.954	0	142.000
churnnew	3.303	12.442	0	116.000
churndeleted	1.246	11.525	0	152.000

The number of source files was 175

- The above metrics were eliminated from further analysis, because they were constant over all files measured.

Table 5.4 discusses the descriptive statistics of the raw metrics of the header files from the fit dataset. The following are the observations that are drawn from the table:

- We see that the number of observations of header files were only 14.
- Only file metric data values were collected. We can understand from this that functions were not defined in header files and hence routine level metrics were not collected by DATRIX.
- Similar to the other datasets, the file metrics *FilComGlbNbr*, *FilComGlbVol*, *FilComTotNbr*, *FilComTotVol*, *FilDecClaNbr*, *FilDecGncTypNbr*, *FilDecGndTypTotNbr*, *FilLnsSkpSum*, *FilDefRtnNbr* and *FilStxErrNbr* had mean and standard deviation of zero.
- The above metrics were eliminated from further analysis because they were constant over all files measured.

The next step was to obtain the correlations among the raw metrics. From this step, we analyzed whether there were any metrics that are highly correlated with each other in the given dataset. If so, we could use either of the metrics in further analysis, that were found to be highly correlated. This reduced the redundancy in the independent variables of the model. The following is the mathematical formula to compute a correlation coefficient r between a pair of metrics. Here x_i and y_i are raw metrics for observation i [26]:

$$r = \frac{\Sigma x_i y_i - \frac{\Sigma x_i \Sigma y_i}{n}}{\sqrt{[\Sigma [x_i]^2 - \frac{(\Sigma x_i)^2}{n}][\Sigma {y_i}^2 - \frac{(\Sigma {y_i}^2)}{n}]}} \tag{5.1}$$

Table 5.5 lists the metrics that where found to be highly correlated among the metrics in the total data file. From the table we see that the routine level metrics *RtnStmCtlCaseNbr* and *RtnStmCtlBrkNbr* were almost perfectly correlated, hence either one of these metrics

Table 5.3 Descriptive Statistics of C Dataset

Variable	Mean	Std Dev	Minimum	Maximum
FilComGlbNbr	0.000	0.000	0	0.000
FilComGlbVol	0.000	0.000	0	0.000
FilComTotNbr	0.000	0.000	0	0.000
FilComTotVol	0.000	0.000	0	0.000
FilDecClaNbr	0.000	0.000	0	0.000
FilDecGncTypNbr	0.000	0.000	0	0.000
FilDecGndTypTotNbr	0.000	0.000	0	0.000
FilDecObjExtNbr	0.006	0.080	0	1.000
FilDecStruNbr	0.101	0.690	0	7.000
FilDefObjGlbNbr	1.013	6.106	0	77.000
FilDefRtnNbr	1.614	2.415	0	17.000
FilIncDirNbr	1.570	0.720	0	3.000
FilIncNbr	18.835	5.281	0	26.000
FilLnsNbr	96.740	141.3279	0	1166.000
FilLnsSkpSum	0.000	0.000	0	0.000
FilStxErrNbr	0.000	0.000	0	0.000
RtnArgXplSum	21.076	29.950	0	273.000
RtnCalXplNbr	7.380	13.010	0	113.000
RtnCastXplNbr	4.576	17.304	0	208.000
RtnComNbr	0.000	0.000	0	0.000
RtnComVol	0.000	0.000	0	0.000
RtnCplCtlAvg	12.409	11.0457	0	47.000
RtnCplCtlMax	25.766	23.425	0	84.000
RtnCplCtlSum	73.360	123.028	0	1226.000
RtnCplCycNbr	9.860	24.963	0	278.000
RtnCplExeAvg	9.137	4.859	0	39.250
RtnCplExeMax	8.582	13.157	0	63.000
RtnCplExeSum	198.101	438.142	0	4417.000
RtnLblNbr	0.000	0.000	0	0.000
RtnLnsNbr	66.437	117.228	0	1049.000
RtnLnsSkpSum	0.000	0.000	0	0.000
RtnScpNbr	11.342	28.44	0	302.000
RtnScpNstLvlAvg	1.974	0.984	0	5.686

The number of source files was 158

Table 5.3 Descriptive Statistics of C Dataset (continued)

Variable	Mean	Std Dev	Minimum	Maximum
RtnScpNstLvlMax	3.025	2.258	0	13.000
RtnScpNstLvlSum	35.911	123.094	0	1264.000
RtnStmCtlBrkNbr	1.272	9.691	0	116.000
RtnStmCtlCaseNbr	1.044	8.713	0	104.000
RtnStmCtlCtnNbr	0.044	0.235	0	2.000
RtnStmCtlDfltNbr	0.114	1.123	0	14.000
RtnStmCtlGotoNbr	0.000	0.000	0	0.000
RtnStmCtlIfNbr	5.253	8.323	0	56.000
RtnStmCtlLopNbr	1.835	8.774	0	104.000
RtnStmCtlNbr	11.513	25.119	0	278.000
RtnStmCtlRetNbr	2.975	2.785	0	19.000
RtnStmCtlSwiNbr	0.133	1.146	0	14.000
RtnStmCtlThwNbr	0.000	0.000	0	0.000
RtnStmDecNbr	11.709	24.452	0	284.000
RtnStmDecObjNbr	11.683	24.434	0	284.000
RtnStmDecPrmNbr	5.5126	7.062	0	51.000
RtnStmDecRtnNbr	0.012	0.112	0	1.000
RtnStmDecTypeNbr	0.012	0.159	0	2.000
RtnStmExeNbr	19.538	42.354	0	378.000
RtnStmNbr	42.759	88.592	0	940.000
RtnStmNstLvlAvg	1.222	0.734	0	3.545
RtnStmNstLvlSum	85.804	292.809	0	3332.000
RtnStxErrNbr	0.000	0.000	0	0.000
RtnStmXpdNbr	54.956	125.008	0	1238.000
Rchange	20.690	54.770	0	612.000
Rnew	14.101	58.500	0	669.000
Rdeleted	7.335	43.1618	0	517.000
churnchange	5.500	13.782	0	142.000
churnnew	2.430	11.061	0	116.000
churndeleted	0.367	1.228	0	8.000

The number of source files was 158

Table 5.4 Descriptive Statistics of Header Dataset

Variable	Mean	Std Dev	Minimum	Maximum
FilComGlbNbr	0.000	0.000	0	0.000
FilComGlbVol	0.000	0.000	0	0.000
FilComTotNbr	0.000	0.000	0	0.000
FilComTotVol	0.000	0.000	0	0.000
FilDecClaNbr	0.000	0.000	0	0.000
FilDecGncTypNbr	0.000	0.000	0	0.000
FilDecGndTypTotNbr	0.000	0.000	0	0.000
FilDecObjExtNbr	5.000	14.109	0	52.000
FilDecStruNbr	0.643	1.865	0	7.000
FilDefObjGlbNbr	0.214	0.802	0	3.000
FilDefRtnNbr	0.000	0.000	0	0.000
FilIncDirNbr	0.857	1.657	0	6.000
FilIncNbr	2.214	5.132	0	18.000
FilLnsNbr	58.714	86.343	0	222.000
FilLnsSkpSum	0.000	0.000	0	0.000
FilStxErrNbr	0.000	0.000	0	0.000
Rchange	12.214	17.850	0	59.000
Rnew	20.071	35.642	0	123.000
Rdeleted	13.357	39.212	0	148.000
churnchange	3.643	4.667	0	18.000
churnnew	6.428	10.250	0	36.000
churndeleted	0.571	1.158	0	3.000

The number of source files was 14

can be considered for further analysis. We preferred *RtnStmCtlCaseNbr* for further analysis. Also given in that table are those metrics that were found to be highly correlated, but these were not eliminated from further anlysis as they were not perfectly correlated.

The Table 5.6 shows the metrics that where found to be highly correlated among the raw metrics collected from the header data file. The routine level metrics *FilDecObjExtNbr* and *FilDefObjGlbNbr* and the file metrics *FilIncDirNbr* and *FilIncNbr* were almost perfectly correlated, hence either one of each pair could be considered metrics for further analysis. We preferred *FilDefObjGlbNbr* and *FilIncNbr* for further analysis. Also given in that table are those metrics that were found to be highly correlated, but these were not eliminated from further anlysis as they were not perfectly correlated.

Table 5.7 shows the metrics that were found to be highly correlated among the raw metrics collected for the C source files. Similar to the total data file, we see that the routine level metrics *RtnStmCtlCaseNbr* and *RtnStmCtlBrkNbr* were almost perfectly correlated, hence either one of these metrics could be considered for further analysis. We preferred *RtnStmCtlCaseNbr* for further analysis. Also given in that table are those metrics that were found to be highly correlated, but these were not eliminated from further anlysis as they were not perfectly correlated.

5.4 Modeling

The following section explains in detail the principal components analysis procedure and the regression technique used to build the model. Metrics *RtnStmCtlLopNbr*,

Table 5.5 Correlation Coefficients of Raw Metrics of Total Dataset

Variable	*RtnStmCtlBrkNbr*	*RtnStmCtlCaseNbr*	*RtnStmCtlCtnNbr*	*RtnStmCtlDfltNbr*
RtnStmCtlBrkNbr	1.000	**0.998**	-0.008	**0.976**
RtnStmCtlCaseNbr	**0.998**	1.000	-0.020	**0.974**
RtnStmCtlCtnNbr	-0.008	-0.020	1.000	-0.017
RtnStmCtlDfltNbr	**0.976**	**0.975**	-0.017	1.000

Table 5.6 Correlation Coefficients of Raw Metrics of Header Dataset

Variable	*FilDecObjExtNbr*	*FilDefObjGlbNbr*	*FilIncDirNbr*	*FilIncNbr*
FilDecObjExtNbr	1.00000	**0.95876**	0.24340	0.08924
FilDefObjGlbNbr	**0.95876**	1.00000	0.19846	-0.01202
FilIncDirNbr	0.24340	0.19846	1.00000	**0.95345**
FilIncNbr	0.08924	-0.01202	**0.95345**	1.00000

Table 5.7 Correlation Coefficients of Raw Metrics of C Dataset

Variable	*RtnStmCtlBrkNbr*	*RtnStmCtlCaseNbr*	*RtnStmCtlCtnNbr*	*RtnStmCtlDfltNbr*
RtnStmCtlBrkNbr	1.000	**0.998**	-0.011	**0.976**
RtnStmCtlCaseNbr	**0.998**	1.000	-0.022	**0.974**
RtnStmCtlCtnNbr	-0.010	-0.023	1.000	-0.019
RtnStmCtlDfltNbr	**0.976**	**0.974**	-0.019	1.000

RtnStmCtlNbr, and *RtnStmDecNbr* were removed due to PCA error "correlation matrix is singular:some scoring coefficients will be zero". In the total data file and C data file from the fit dataset, we used 37 product metrics after eliminating undesirable metrics as discussed above.

5.4.1 Principal Components Analysis

Principal components analysis (PCA) [19] is a technique for transforming multivariate data into variables that are not correlated. These variables in this thesis are referred to as "Factors". In order that a model be robust, the independent variables that the model is built on should be uncorrelated. In order to obtain the principal components, we ran the `Proc Factor` with `Method=Principal Rotate=Varimax` and stopping rule `Min eigenvalue=1`. Figure 5.1 gives the pictorial depiction of principal components analysis. Given below are the step-by-step split of the PCA [19]:

1. Standardize the metric data for a given $n \times m$ matrix represented as **X** with elements x_{ij}, where m is the number of product metrics and the n is the number of modules (here files). For instance, total dataset had n = 175 and m = 37. Let the standardized data matrix be **Z**, with elements z_{ij}. The formula used is

$$z_{ij} = \frac{x_{ij} - \mu_j}{\sigma_j} \tag{5.2}$$

   where μ_j is the mean value for that attribute and σ_j is the standard deviation of that product attribute. This is shown by the step " Standardize" in Figure 5.1.

2. Calculate the covariance matrix, $\mathbf{\Sigma}$ of **Z**. The general formula for covariance of two variables x and y [26] is

$$COV(x, y) = E(xy) - E(x)E(y) \tag{5.3}$$

   where $E(x)$ is the expected value.

3. Reduce the dimensionality of the data, by choosing a stopping rule such as a minimum eigenvalue of one. With this stopping rule only those factors will be chosen whose eigenvalue is at least one. In the Figure 4.1 the eigenvectors form a 37×37 matrix where each column is an eigenvector denoted by e_j and the eigenvalues form a 37×1 vector where each eigenvalue is denoted by λ_j.

4. Calculate the standardized transformation matrix, **T** with columns t_j. The tranformation value for j^{th} value is calculated as follows.

$$t_j = \frac{e_j}{\sqrt{\lambda_j}} \tag{5.4}$$

5. Calculate the factors for each module.

$$\mathbf{D} = \mathbf{ZT} \tag{5.5}$$

   Where the columns of **D** consist of variables *FACTOR 1*, *FACTOR 2*, *FACTOR 3*, *FACTOR 4*, *FACTOR 5*, etc. and the rows represent each module. This is depicted by the `Proc score` in the Figure 4.1, which is the procedure in SAS to multiply matrices.

The variance maximizing rotation or the varimax rotation of the principal component analysis, rotates the original variable space in such a way that it maximizes the variability of the new variables i.e, factors, while minimizing the variance among the raw metrics. This rotation yields the rotated factor table, which makes the interpretation of the metrics easier in terms of the factors.

Table 5.8 shows the rotated factor pattern of the total data file from the fit dataset, which contains the data collected from the C source files and also the header files of the MPICH. The columns of the table are the factors which are extracted from the principal component analysis and the rows are raw metrics. Each entry is the correlation coefficient of a raw metric and a corresponding factor. The factors account for the correlation among the raw metrics. They identify the latent dimensions that may explain why variables are correlated with each other.

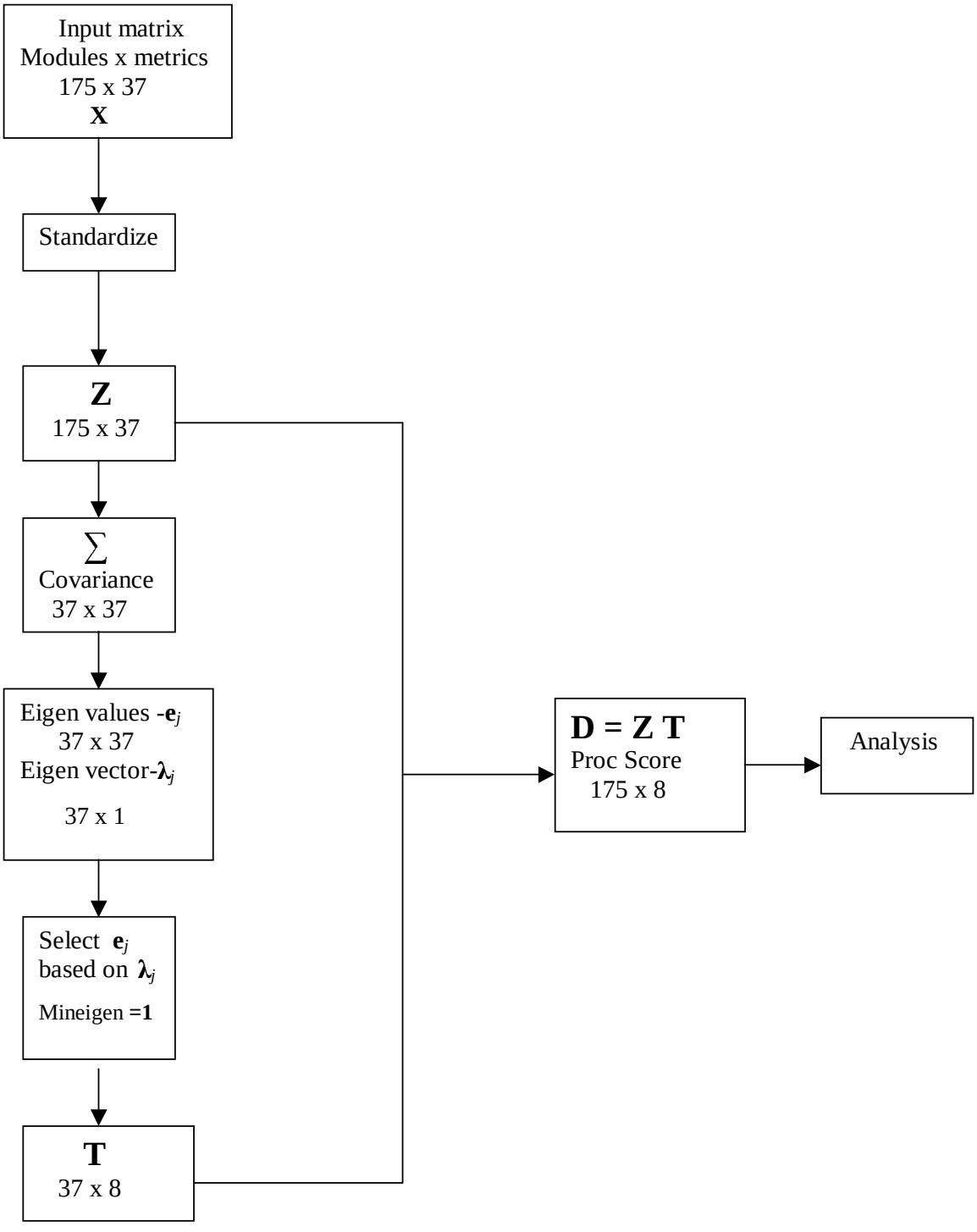


Figure 5.1 Principal Components Analysis

Table 5.8 Rotated Factor Pattern for Total Dataset

Variable	*FACTOR 1*	*FACTOR 2*	*FACTOR 3*	*FACTOR 4*	*FACTOR 5*	*FACTOR 6*	*FACTOR 7*	*FACTOR 8*
RtnStmNstLvlSum	**0.97085**	0.01733	0.06253	0.14906	0.09262	0.11621	0.00131	0.00059
RtnStmCtlDfltNbr	**0.96423**	0.02718	-0.04629	-0.11559	-0.00855	-0.17261	0.00280	0.03010
RtnStmCtlSwiNbr	**0.95885**	0.02682	-0.02092	-0.08554	-0.01091	-0.17885	-0.00195	0.03334
RtnCplCycNbr	**0.95351**	0.04743	0.05949	0.24358	0.08651	0.10211	-0.00397	0.00922
RtnCastXplNbr	**0.95233**	0.06511	0.09928	0.00727	0.01891	-0.00199	0.01681	0.03027
RtnStmDecObjNbr	**0.94593**	0.13361	0.03381	0.23987	0.02013	0.00700	-0.00128	-0.01070
RtnStmCtlCaseNbr	**0.94054**	0.01802	-0.00766	-0.10225	-0.00946	-0.18252	-0.00536	0.04465
RtnScpNbr	**0.92421**	0.03107	0.04934	0.30719	0.10878	0.14456	-0.00055	-0.00808
RtnStmNbr	**0.91885**	0.09092	0.22833	0.23749	0.08055	0.16663	-0.00176	-0.00218
RtnScpNstLvlSum	**0.90597**	-0.01926	0.02403	0.29706	0.12047	0.10253	0.00062	-0.02192
RtnStmXpdNbr	**0.89206**	0.06584	0.17166	0.21785	0.13723	0.30463	-0.00117	-0.01121
RtnCplExeSum	**0.88109**	0.09489	0.34593	0.22676	0.08900	0.12389	0.00625	-0.01395
RtnCplCtlSum	**0.86643**	0.21529	0.05111	0.29836	0.11426	0.23493	-0.01917	-0.05640
RtnStmExeNbr	**0.81840**	0.06281	0.42363	0.22968	0.09293	0.24339	0.00332	-0.00801
RtnLnsNbr	**0.81470**	0.10968	0.34911	0.27093	0.09131	0.31453	-0.00269	-0.00477
FilLnsNbr	**0.74891**	0.04284	0.40917	0.28875	0.06075	0.30452	0.15278	-0.02356
RtnStmCtlIfNbr	**0.57531**	0.08655	0.16420	0.54197	0.23157	0.45627	-0.01752	-0.03515
FilDecStruNbr	**0.51852**	-0.17959	0.26508	-0.14330	-0.07984	-0.03769	0.43567	-0.05739
RtnCplCtlMax	0.01575	**0.86604**	0.00925	-0.01119	-0.13326	0.05153	-0.05996	-0.18270
RtnCplCtlAvg	-0.03014	**0.82587**	-0.06657	-0.09851	-0.16164	-0.01654	-0.06257	-0.18306
RtnCplExeAvg	0.06558	**0.74211**	0.05030	0.17492	0.26627	-0.05056	-0.09500	-0.04348
FilIncNbr	0.04727	**0.73916**	0.10437	0.22343	0.27340	0.09633	-0.07498	0.33468
RtnCplExeMax	0.30029	**0.61894**	0.38063	0.26104	0.26594	-0.00903	-0.04977	-0.07604
FilIncDirNbr	0.08037	**0.61524**	0.17035	0.10231	0.18570	0.18289	0.28653	0.33640

Table 5.8 Rotated Factor Pattern for Total Dataset (continued)

Variable	*FACTOR 1*	*FACTOR 2*	*FACTOR 3*	*FACTOR 4*	*FACTOR 5*	*FACTOR 6*	*FACTOR 7*	*FACTOR 8*
FilDefRtnNbr	0.48989	0.06068	**0.78902**	0.15996	-0.03427	-0.00124	0.00990	0.01790
RtnStmDecPrmNbr	0.59619	0.21498	**0.68808**	0.09490	-0.03864	-0.07208	-0.01271	-0.07022
RtnScpNstLvlMax	0.29434	0.30161	**0.63332**	0.28963	0.43938	0.15923	-0.06595	-0.00476
FilDefObjGlbNbr	0.02980	-0.02036	-0.11629	**0.90251**	0.00523	-0.04168	0.03233	-0.01435
RtnCalXplNbr	0.18465	0.08982	0.28664	**0.86074**	0.07655	0.08066	-0.01936	0.01918
RtnArgXplSum	0.26324	0.22647	0.21596	**0.85460**	0.04686	0.09771	-0.01864	0.00508
RtnStmCtlCtnNbr	-0.00574	-0.01425	-0.09966	-0.01137	**0.85191**	-0.05607	0.02988	-0.09052
RtnStmNstLvlAvg	0.34057	0.39107	0.35074	0.10630	**0.66329**	0.18023	-0.10109	0.10769
RtnScpNstLvlAvg	0.26628	0.49372	0.37319	0.20460	**0.61644**	0.14363	-0.11569	0.04383
RtnStmDecTypeNbr	0.15194	-0.02235	-0.06357	0.01681	-0.00352	**0.91672**	-0.00152	-0.00857
RtnStmCtlRetNbr	0.05755	0.36501	0.47708	0.12997	0.05446	**0.61633**	-0.04687	0.03776
FilDecObjExtNbr	-0.03178	-0.09944	-0.00265	-0.04677	-0.02723	-0.01473	**0.91476**	-0.03808
RtnStmDecRtnNbr	-0.01111	-0.11568	-0.03465	-0.02561	-0.07022	-0.01677	-0.05280	**0.87321**
Eigenvalue	18.1864	5.1404	2.5454	1.9105	1.4883	1.2203	1.0915	1.0415
Difference	13.0459	2.5950	0.6348	0.4221	0.2680	0.1287	0.0499	0.2035
Proportion	0.4915	0.1389	0.0688	0.0516	0.0402	0.0330	0.0295	0.0282
Cumulative	0.4915	0.6305	0.6993	0.7509	0.7911	0.8241	0.8536	0.8817

Eight factors were extracted, which accounted for 37 variables, that we gave the PCA.

The metrics whose correlation coefficients are shown in bold are highly correlated with the corresponding factor. The following are the observations that can be drawn from the rotated factor pattern of the total data file Table 5.8:

- The metrics highly correlated with *FACTOR 1* are, *RtnStmNstLvlSum*, *RtnStmCtlDfltNbr*, *RtnStmCtlSwiNbr*, *RtnCplCycNbr*, *RtnCastXplNbr*, *RtnStmDecObjNbr*, *RtnStmCtlCaseNbr*, *RtnScpNbr*, *RtnStmNbr*, *RtnStmXpdNbr*, *RtnCplExeSum*, *RtnCplCtlSum*, *RtnScpNstLvlSum*, *RtnStmExeNbr*, *RtnLnsNbr*, *FilLnsNbr*, *FilDecStruNbr*, and *RtnStmCtlIfNbr*. We see that *FACTOR 1* accounted for cyclomatic complexity [25], the lines of code at the routine and the file level (size) and the number of control structures.
- The metrics that were correlated with *FACTOR 2* are *RtnCplCtlMax*, *RtnCplCtlAvg*, *RtnCplExeMax*, *RtnCplExeAvg*, *FilIncNbr*, and *FilIncDirNbr*. *FACTOR 2* was associated with the predicate complexity and the file includes.
- *FACTOR 3* was correlated with *FilDefRtnNbr*, *RtnStmDecPrmNbr*, and *RtnScpNstLvlMax*. *FACTOR 3* accounts for the number of parameter (variable) declarations, within each routine or function. *FACTOR 3* was associated with the function definitions.
- *FACTOR 4* was correlated with *FilDefObjGlbNbr*, *RtnCalXplNbr*, and *RtnArgXplSum*. *FACTOR 4* is associated with the number of variables declarations which represent the interfaces between modules.
- *FACTOR 5* was correlated with *RtnStmCtlCtnNbr*, *RtnScpNstLvlAvg*, and *RtnStmNstLvlAvg*. *FACTOR 5* represents the depth to which the control structures are nested.
- The metrics correlated with the *FACTOR 6* were *RtnStmDecTypeNbr*, and *RtnStmCtlRetNbr*. *FACTOR 6* accounts for the number of type declarations statements in the routine and the number of return statements.
- The metric associated with *FACTOR 7* was *FilDecObjExtNbr*, which is the number of `extern` objects declared in the file. This is the only metric that was associated with *FACTOR 7*.
- The metric associated with *FACTOR 8* was *RtnStmDecRtnNbr*, which was the number of routine declarations within the routine.

Table 5.9 presents the rotated factor pattern for the data collected from the C source code of the fit dataset. The rotated factor pattern for the data collected from the C source code is similar to the rotated factor pattern of the total datafile. The following are the observations that can be drawn from the table:

- The metrics highly correlated with *FACTOR 1* were, *RtnStmNstLvlSum*, *RtnStmCtlDfltNbr*, *RtnStmCtlSwiNbr*, *RtnCplCycNbr*, *RtnCastXplNbr*, *RtnStmDecObjNbr*, *RtnStmCtlCaseNbr*, *RtnScpNbr*, *RtnStmNbr*, *RtnScpNstLvlSum*, *RtnStmXpdNbr*, *RtnCplExeSum*, *RtnCplCtlSum*, *RtnStmExeNbr*, *RtnLnsNbr*, *FilLnsNbr*, and *RtnStmCtlIfNbr*. *FACTOR 1* accounted for cyclomatic complexity [25], the lines of code at the routine and the file level and the number of control structures.

- The metrics that were correlated with the *FACTOR 2* were *FilDefObjGlbNbr*, *RtnCalXplNbr*, and *RtnArgXplSum*. *FACTOR 2* is associated with the number of variables declarations which represent the interfaces between modules.

- *FACTOR 3* was correlated with *RtnCplCtlMax*, *RtnCplCtlAvg*, *RtnCplExeAvg*, *RtnCplExeMax*, *FilIncNbr*, and *FilIncDirNbr*. *FACTOR 3* accounted for the control predicate statement complexity of the control structures and the number of direct and indirect file includes.

- *FACTOR 4* was correlated with *FilDefRtnNbr*, *RtnStmDecPrmNbr*, and *RtnScpNstLvlMax*. *FACTOR 4* accounts for the number of parameter (variable) declarations, within each routine or function. *FACTOR 3* was associated with the function definitions.

- *FACTOR 5* was correlated with *RtnStmCtlCtnNbr*, *RtnScpNstLvlAvg*, and *RtnStmNstLvlAvg*. *FACTOR 5* represents the depth to which the control structures are nested.

- The metrics correlated with the *FACTOR 6* were *RtnStmDecTypeNbr*, and *RtnStmCtlRetNbr*. *FACTOR 6* accounts for the number of type declarations statements in the routine and the number of return statements.

- The metric associated with *FACTOR 7* was *FilDecObjExtNbr*, and *FilDecStruNbr*. *FilDecObjExtNbr* is the number of `extern` objects declared in the file. *FACTOR 7* accounts for the number of extern object declarations.

- The metric associated with *FACTOR 8* was *RtnStmDecRtnNbr*, which is the number of routine declarations within the routine.

Table 5.9 Rotated Factor Pattern for C Dataset

Variable	*FACTOR 1*	*FACTOR 2*	*FACTOR 3*	*FACTOR 4*	*FACTOR 5*	*FACTOR 6*	*FACTOR 7*	*FACTOR 8*
RtnStmNstLvlSum	**0.96991**	0.06186	-0.00509	0.13878	0.11635	0.11792	0.01865	-0.00310
RtnStmCtlDfltNbr	**0.96660**	-0.03838	0.02867	-0.10384	-0.01362	-0.17537	0.00127	0.02315
RtnStmCtlSwiNbr	**0.96083**	-0.01273	0.02618	-0.07394	-0.01290	-0.17986	-0.00451	0.02718
RtnCastXplNbr	**0.95478**	0.10786	0.04988	0.00154	0.02759	0.00148	0.00214	0.02529
RtnCplCycNbr	**0.95363**	0.06067	0.01496	0.23020	0.11871	0.11024	0.00175	0.00642
RtnStmDecObjNbr	**0.94821**	0.03757	0.10555	0.23928	0.05211	0.01638	-0.00336	-0.00657
RtnStmCtlCaseNbr	**0.94318**	0.00153	0.01375	-0.09397	-0.01543	-0.18313	-0.01321	0.03628
RtnScpNbr	**0.92285**	0.04700	-0.00207	0.29319	0.14674	0.15274	0.01242	-0.00869
RtnStmNbr	**0.92005**	0.22969	0.05429	0.22214	0.11557	0.17549	0.01570	-0.00259
RtnScpNstLvlSum	**0.90222**	0.01873	-0.04300	0.28891	0.15680	0.10593	0.02462	-0.02220
RtnStmXpdNbr	**0.89199**	0.16972	0.02838	0.19297	0.17125	0.30868	0.02142	-0.01409
RtnCplExeSum	**0.88051**	0.34732	0.06431	0.21570	0.12541	0.13055	0.02829	-0.01248
RtnCplCtlSum	**0.86712**	0.04490	0.17910	0.28967	0.16888	0.24481	0.01396	-0.05003
RtnStmExeNbr	**0.81730**	0.42302	0.02636	0.21021	0.12818	0.25087	0.03924	-0.00790
RtnLnsNbr	**0.81676**	0.35103	0.06308	0.24490	0.13035	0.32729	0.01561	-0.00583
FilLnsNbr	**0.74456**	0.39401	0.03691	0.30175	0.12514	0.31290	0.24455	-0.00585
RtnStmCtlIfNbr	**0.57040**	0.15606	0.02042	0.50063	0.30254	0.47829	0.00412	-0.03788
FilDefObjGlbNbr	0.03119	**0.89997**	-0.03958	-0.12019	-0.01038	-0.05035	0.02993	-0.02928
RtnArgXplSum	0.25803	**0.87227**	0.16897	0.19082	0.08464	0.11464	-0.00128	0.00382
RtnCalXplNbr	0.17360	**0.86432**	0.03670	0.26521	0.11439	0.09616	0.05205	0.02259
RtnCplCtlMax	-0.00381	-0.01016	**0.88457**	0.03000	-0.07529	0.05328	0.03414	-0.12765
RtnCplCtlAvg	-0.04983	-0.09475	**0.83869**	-0.05848	-0.13293	-0.02366	0.03825	-0.12737
RtnCplExeAvg	0.05780	0.05919	**0.66434**	0.10518	0.32837	-0.05227	-0.16853	-0.07854
FilIncNbr	0.04102	0.12930	**0.61476**	0.13434	0.35240	0.13777	-0.28785	0.36390
FilIncDirNbr	0.09663	0.23324	**0.59687**	0.05979	0.28916	0.22960	-0.09856	0.34905
RtnCplExeMax	0.30408	0.41074	**0.54868**	0.22769	0.33280	-0.00335	-0.07669	-0.08572

Table 5.9 Rotated Factor Pattern for C Dataset (continued)

Variable	*FACTOR 1*	*FACTOR 2*	*FACTOR 3*	*FACTOR 4*	*FACTOR 5*	*FACTOR 6*	*FACTOR 7*	*FACTOR 8*
FilDefRtnNbr	0.48482	0.16202	0.00946	**0.79296**	0.03051	0.03209	-0.01579	0.03902
RtnStmDecPrmNbr	0.59512	0.09592	0.17218	**0.70970**	0.02693	-0.04691	-0.02287	-0.04866
RtnScpNstLvlMax	0.28945	0.29305	0.16553	**0.59642**	0.53452	0.18700	-0.05836	-0.01938
RtnStmCtlCtnNbr	-0.01846	-0.03489	-0.06666	-0.15258	**0.80941**	-0.09521	0.06860	-0.09096
RtnStmNstLvlAvg	0.34954	0.09467	0.21985	0.26231	**0.76778**	0.19660	-0.09287	0.08776
RtnScpNstLvlAvg	0.27520	0.20770	0.32931	0.29901	**0.73964**	0.16278	-0.10550	0.02047
RtnStmDecTypeNbr	0.15580	0.01338	-0.03608	-0.10861	-0.00136	**0.90457**	0.01312	-0.01738
RtnStmCtlRetNbr	0.04815	0.13804	0.27143	0.42328	0.11781	**0.67774**	-0.09956	0.03414
FilDecObjExtNbr	-0.04032	-0.00988	-0.11386	-0.00545	-0.02425	-0.00787	**0.93619**	0.00613
FilDecStruNbr	0.57035	0.24908	-0.10750	-0.07131	-0.06508	-0.05887	**0.61867**	0.00086
RtnStmDecRtnNbr	-0.01735	-0.04574	-0.15161	-0.02016	-0.06916	-0.03018	0.02464	**0.87860**
Eigenvalue	18.1467	4.8445	2.6428	2.0224	1.5799	1.2645	1.2213	1.0004
Difference	18.1467	13.3021	2.2017	0.6202	0.4425	0.3153	0.0427	0.2213
Proportion	0.4905	0.1309	0.0714	0.0547	0.0427	0.0342	0.0330	0.0270
Cumulative	0.4905	0.6214	0.6928	0.7475	0.7902	0.8244	0.8574	0.8844

`extern` objects are storage class variables which are declared when the source program spawns more an one source file, it tells the compiler that the given variable types and names are already declared somewhere and there is no need to create storage space for them. When source code consists of multiple files, then each file can share a variable provided it is declared as an external variable. They are one way that files may be coupled together.

5.4.2 Regression

The following equation is a classical example of a multiple linear regression model. The y is the dependent variable, which can be predicted from the set of independent variables, $x_1, x_2, x_3 \ldots x_{ip}$. y in the current thesis is the total change, the set of x_j are the product attributes and reuse variables. The a_j are the parameter estimates which are calculated from the known values of the x's and y. For one observation i,

$$y_i = a_0 + a_1 x_{i1} + a_2 x_{i2} + a_3 x_{i3} + ... + a_p x_{ip} \tag{5.6}$$

In order to obtain the parameter estimates for the model, we use the least squares method. This paragraph gives an example of how the parameter estimates are calculated using the least squares method in a simple case. Consider the following example that has two independent variables and one dependent variable

$$z = a + bx + cy \tag{5.7}$$

For a given dataset $(x_1, y_1, z_1), (x_2, y_2, z_2), \cdots, (x_n, y_n, z_n)$ the best fitting plane $f(x, y)$ has a least squared error, that is,

$$\begin{aligned} E = \sum_{i=1}^{n} {e_i}^2 &= \sum_{i=1}^{n} [z_i - f(x_i, y_i)]^2 \\ &= \sum_{i=1}^{n} [z_i - (a + bx_i + cy_i)]^2 = \text{ minimum squared error} \end{aligned} \tag{5.8}$$

in the equation a, b, c are unknown and x_i, y_i, z_i are known for all i. We can obtain the values of a, b, c by equating the first partial derivatives of the equation to zero.

$$\begin{aligned} \frac{\partial E}{\partial a} &= 2 \sum_{i=1}^{n} [z_i - (a + bx_i + cy_i)] \\ &= 0 \end{aligned} \tag{5.9}$$

$$\begin{aligned} \frac{\partial E}{\partial b} &= 2 \sum_{i=1}^{n} x_i [z_i - (a + bx_i + cy_i)] \\ &= 0 \end{aligned} \tag{5.10}$$

$$\begin{aligned} \frac{\partial E}{\partial c} &= 2 \sum_{i=1}^{n} y_i [z_i - (a + bx_i + cy_i)] \\ &= 0 \end{aligned} \tag{5.11}$$

Solving the above linear equations we can obtain the values of the unknown coefficients a, b, c. `Proc Reg` extends this approach to multiple independent variables. This step is calculated by `Proc Reg` depicted as the first circle in the Figure 5.2.

In a stepwise procedure in multiple regression, the forward selection procedure alternates with the backward elimination procedure. In the forward selection procedure, analysis begins with no variables in the regression model. For each variable, a statistic

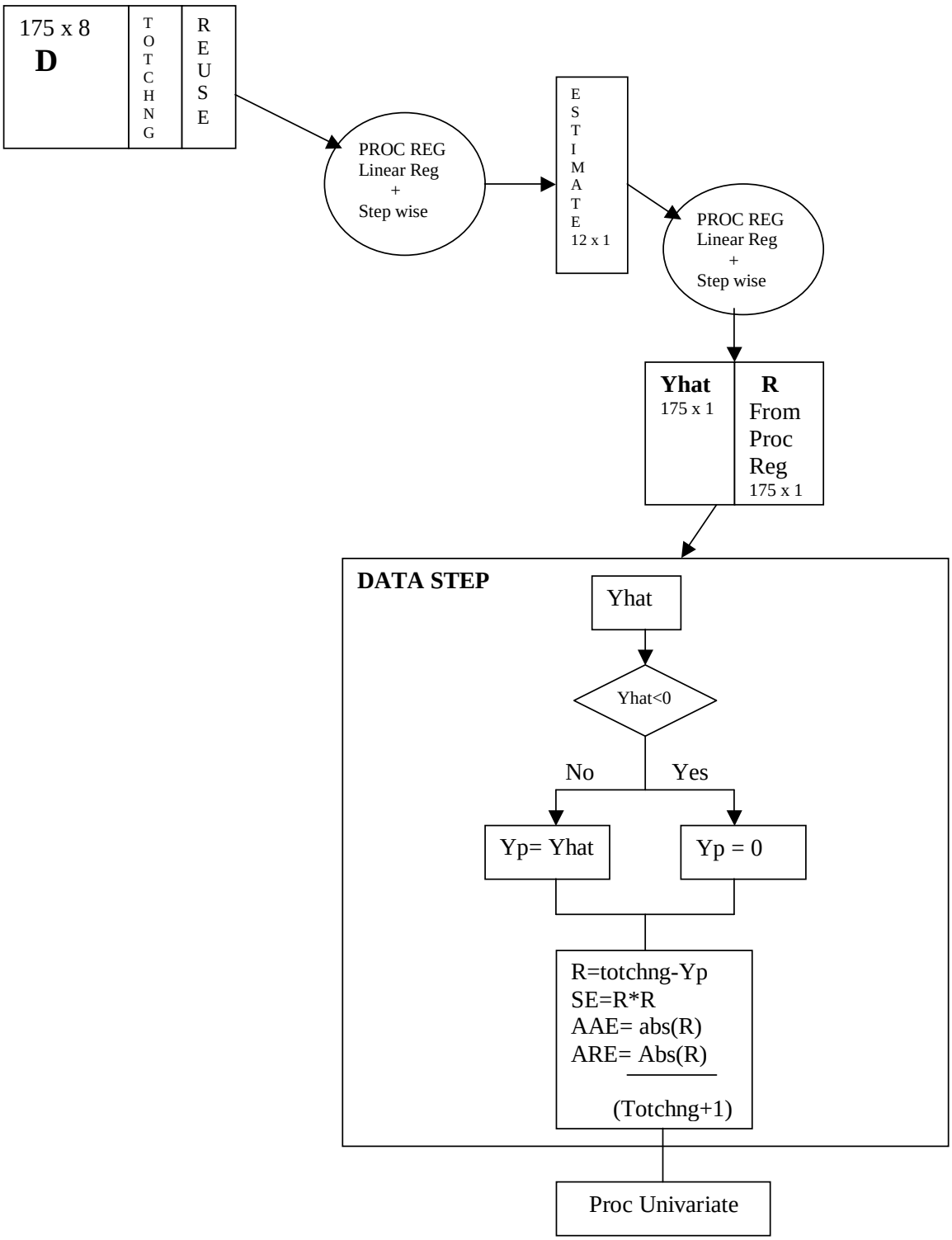


Figure 5.2 Linear Regression

called the F-value is calculated. The F-value signifies the amount of a selected variable's contribution to explaining the behavior of the dependent variable. The variable with the highest value of the F-statistic is considered for entry into the model. If the F-value is significant, that is below the significance level of entry, here 15%, then the variable is added to the model. In the backward elimination procedure, for each variable currently in the model the F-value is calculated for removal from the model. The variable with the lowest value of the F-statistic is considered for removal from the model. This process is depicted by the second `proc reg` circle in Figure 5.2.

After the regression variables are selected and the parameters are estimated, predicted code-churn was calculated for each file $\hat{y}_i$. The SAS data step specifies that all the values of predicted total change $\hat{y}_i$ that are negative, be made zero. Since from the definition code-churn cannot be negative.

$$y_{pi} = \begin{cases} \hat{y}_i & \text{if } \hat{y}_i > 0 \\ 0 & \text{otherwise} \end{cases}$$

The results of the regression procedure on the three datasets are discussed in the following paragraphs. For the total dataset containing the C file data and the header file data of the MPICH, the independent variables that were given to the regression procedure were *FACTOR 1*, *FACTOR 2*, *FACTOR 3*, *FACTOR 4*, *FACTOR 5*, *FACTOR 6*, *FACTOR 7*, *FACTOR 8, Rchange, Rnew, Rdeleted*. Using the stepwise procedure the regression procedure chose *FACTOR 3* and *FACTOR 7* as the independent variables for the model. The linear regression model is as below:

$$\hat{y} = 10.41 + 9.9\textit{FACTOR 3} + 6.68\textit{FACTOR 7} \tag{5.12}$$

Table 5.10 gives the stepwise selection results at a significance level of 0.15. The Table 5.11 gives the quantile values of the variable y_p for the total data file.

Table 5.10 Summary of Stepwise Selection for the Total Dataset

Variable	Parameter Estimate	Standard Error	F value	$\Pr > F$
Intercept	10.41	2.18	22.80	$< .0001$
FACTOR 3	10.10	2.18	20.61	$< .0001$
FACTOR 7	6.67	2.18	9.35	0.0026

From the regression equation, we can say that *FACTOR 3*, which represented the number of function definitions in a file had the most influence on the total change for the total dataset. *FACTOR 7*, which represented the number of `extern` declarations, also had significant influence on the total change.

For the C data file the following were the independent variables that were given to the regression procedure: *FACTOR 1*, *FACTOR 2*, *FACTOR 3*, *FACTOR 4*, *FACTOR 5*, *FACTOR 6*, *FACTOR 7*, *FACTOR 8*, *Rchange*, *Rnew*, and *Rdeleted*. The stepwise procedure chose *FACTOR 2* and *FACTOR 4*. The regression fit equation for the model is as follows:

$$\hat{y} = 8.28 + 9.9\textit{FACTOR 2} + 4.5\textit{FACTOR 4} \tag{5.13}$$

Table 5.11 Spread of the Predicted Code-Churn in Quantiles for Total Dataset

Quantile	y_p
100%Max	128.9
99%	75.3
95%	23.9
90%	15.8
75%Q3	10.9
50%Median	7.4
25%Q1	5.7
10%	4.3
5%	4.3

From the above regression equation we see that *FACTOR 2*, which represented the number of function definitions, had the most influence on the total change for C source code observations. *FACTOR 4*, which represents the number of variables declarations, also had significant influence on the total change.

Table 5.12 gives the stepwise selection results at a significance level of 15% for the data collected from the C source files. Table 5.13 gives the quantile values of the variable y_p.

For the header dataset, the following are the variables given to the regression equation *FilDecStruNbr*, *FilDefObjGlbNbr*, *FilIncNbr*, *FilLnsNbr*, *Rchange*, *Rnew*, and *Rdeleted*. The stepwise regression chose *FilDecStruNbr*, *FilIncNbr*, *FilLnsNbr*, and *Rnew* as the independent variables for the model. Table 5.14 gives the stepwise selection results at a significance level of 15% for the data collected from the header files. Table 5.15 gives the quantile values of the variable y_p. The regression fit equation for the model is as follows:

Table 5.12 Summary of Stepwise Selection for the C Dataset

Variable	Parameter Estimate	Standard Error	F Value	$\Pr > F$
Intercept	8.30	1.33	38.72	$< .0001$
FACTOR 2	9.90	1.33	54.94	$< .0001$
FACTOR 4	4.59	1.33	11.81	0.0008

Table 5.13 Spread of the Predicted Code-Churn in Quantiles for C Dataset

Quantile	y_p
100% Max	116.0
99%	41.6
95%	22.3
90%	14.4
75% Q3	9.8
50% Median	5.2
25% Q1	4.2
10%	3.3
5%	2.2

Table 5.14 Summary of Stepwise Selection for the Header Dataset

Variable	Parameter Estimate	Standard Error	F Value	$\Pr > F$
Intercept	14.71822	7.8990	3.47	0.0643
FilDecStruNbr	9.63191	3.2781	8.63	0.0038
FilIncNbr	-0.63568	0.4292	2.19	0.1406
FilLnsNbr	0.08824	0.0237	13.76	0.0003
Rnew	-0.21280	0.0519	16.75	0.0001

Table 5.15 Spread of the Predicted Code-Churn in Quantiles for Header Dataset

Quantile	y_p
100% Max	104.7
99%	91.1
95%	29.9
90%	14.7
75% Q3	10.3
50% Median	7.2
25% Q1	5.3
10%	4.1
5%	0.4

$$\hat{y} = 14.7 + 9.6\textit{FilDecStruNbr} - 0.636\textit{FilIncNbr} + 0.088\textit{FilLnsNbr} - 0.213\textit{Rnew} \quad (5.14)$$

Table 5.14 we see that the number of new lines had the most influence on the total change followed by the number of lines of code, the number of structure declarations and the number of file includes. The total change in header files can be attributed to functional enhancements, related to corresponding changes in C source files.

In the SAS procedure for linear regression given in the Appendix, `data` specifies the SAS dataset to be used by the regression procedure. The `outest` statement requests that the parameter estimates be calculated to the specified SAS dataset. The statement `simple` requests that simple descriptive statistics be specified for each variable used in the procedure. The `model` statement specifies the dependent and the candidate independent variables in the regression model. The `selection` option specifies how the variables should be introduced in the model, if the option is set to none, then all the variables in

the model are selected. If the option is set to `stepwise` then only those variables that are significant at the `slentry` level are selected and deletes the variable if they are not significant at the `slstay` level. The `output` is an important option and must follow the `model` statement. The `output` statement creates a SAS dataset which contains the original dataset along with the new variables as specified in the output statement `predicted=yhat`, `residual=R`, `press=PRESS`, `rstudent=RST`.

The `plot` procedure gives a plot of the variables specified, here the residual vs the predicted value of code-churn. The proc `univariate` gives descriptive statistics such as the mean, standard deviation, skewness, variance of each variable specified. Figure 5.2 gives the pictorial depiction of the regression procedure for the fit dataset.

5.5 Inferential Statistics

The data for the purpose of validation of the model were collected from the versions 1.0.5 and 1.0.6. Reuse variables were collected from the changes in source code between versions SEP/94 and 1.0.5. The product metrics were collected from version 1.0.5 and the actual code-churn was measured from the changes in source code of versions 1.0.5 and 1.0.6. The summary statistics of the total validation data is given in Table 5.16. The summary statistics for the C validation dataset is in Table 5.17 and the summary statistics for the header file data is in Table 5.18.

The Figure 5.3 gives the pictorial depiction of the validation procedure. The procedure for the purpose of validation of the model follows the SAS procedure for regression. In

Table 5.16 Summary Statistics of Validation Total Dataset

Variable	Mean	Std Dev	Minimum	Maximum
FilComGlbNbr	0.000	0	0	0.000
FilComGlbVol	0.000	0	0	0.000
FilComTotNbr	0.000	0	0	0.000
FilComTotVol	0.000	0	0	0.000
FilDecClaNbr	0.000	0	0	0.000
FilDecGncTypNbr	0.000	0	0	0.000
FilDecGndTypTotNbr	0.000	0	0	0.000
FilDecObjExtNbr	0.460	4.241	0	52.000
FilDecStruNbr	0.2393	1.110	0	8.000
FilDefObjGlbNbr	1.123	6.572	0	84.000
FilDefRtnNbr	1.687	2.518	0	17.000
FilIncDirNbr	1.656	0.788	0	6.000
FilIncNbr	19.392	5.634	0	29.000
FilLnsNbr	109.269	147.216	0	1166.000
FilLnsSkpSum	0.055	0.254	0	2.000
FilStxErrNbr	0.055	0.254	0	2.000
RtnArgXplSum	22.901	33.1253	0	308.000
RtnCalXplNbr	8.172	14.277	0	124.000
RtnCastXplNbr	4.803	17.3047	0	208.000
RtnComNbr	0.000	0	0	0.000
RtnComVol	0.000	0	0	0.000
RtnCplCtlAvg	12.348	10.951	0	50.000
RtnCplCtlMax	28.085	25.536	0	102.000
RtnCplCtlSum	78.858	125.894	0	1226.000
RtnCplCycNbr	10.276	24.988	0	278.000
RtnCplExeAvg	8.805	4.5471	0	24.500
RtnCplExeMax	17.926	12.580	0	63.000
RtnCplExeSum	204.049	446.913	0	4417.000
RtnLblNbr	0.000	0	0	0.000
RtnLnsNbr	69.239	120.229	0	1049.000
RtnLnsSkpSum	0.061	0.240	0	1.000
RtnScpNbr	11.650	28.383	0	302.000
RtnScpNstLvlAvg	1.931	0.972	0	5.686

The number of observations was 163.

Table 5.16 Summary Statistics of Validation Total Dataset (continued)

Variable	Mean	Std Dev	Minimum	Maximum
RtnScpNstLvlMax	2.938	2.218	0	13.000
RtnScpNstLvlSum	36.429	121.763	0	1264.000
RtnStmCtlBrkNbr	1.257	9.536	0	116.000
RtnStmCtlCaseNbr	1.049	8.587	0	104.000
RtnStmCtlCtnNbr	0.049	0.243	0	2.000
RtnStmCtlDfltNbr	0.122	1.109	0	14.000
RtnStmCtlGotoNbr	0.000	0.000	0	0.000
RtnStmCtlIfNbr	5.595	8.609	0	56.000
RtnStmCtlLopNbr	1.822	8.650	0	104.000
RtnStmCtlNbr	12.006	25.220	0	278.000
RtnStmCtlRetNbr	3.141	3.048	0	19.000
RtnStmCtlSwiNbr	0.141	1.132	0	14.000
RtnStmCtlThwNbr	0.000	0.000	0	0.000
RtnStmDecNbr	11.858	24.368	0	284.000
RtnStmDecObjNbr	11.846	24.341	0	284.000
RtnStmDecPrmNbr	5.607	7.207	0	51.000
RtnStmDecRtnNbr	0.012	0.156	0	2.000
RtnStmDecTypeNbr	0.012	0.156	0	2.000
RtnStmExeNbr	20.079	43.4171	0	378.000
RtnStmNbr	43.957	89.486	0	940.000
RtnStmNstLvlAvg	1.195	0.732	0	3.545
RtnStmNstLvlSum	87.558	290.424	0	3332.000
RtnStxErrNbr	0.061	0.240	0	1.000
RtnStmXpdNbr	56.134	125.421	0	1238.000
Rchange	6.098	15.277	0	142.000
Rnew	5.834	28.692	0	329.000
Rdeleted	1.337	11.937	0	152.000
churnchange	2.926	9.347	0	99.000
churnnew	6.024	54.004	0	684.000
churndeleted	1.061	11.929	0	152.000

The number of observations was 163.

Table 5.17 Summary Statistics of Validation C Dataset

Variable	Mean	Std Dev	Minimum	Maximum
FilComGlbNbr	0.000	0.000	0	0.000
FilComGlbVol	0.000	0.000	0	0.000
FilComTotNbr	0.000	0.000	0	0.000
FilComTotVol	0.000	0.000	0	0.000
FilDecClaNbr	0.000	0.000	0	0.000
FilDecGncTypNbr	0.000	0.000	0	0.000
FilDecGndTypTotNbr	0.000	0.000	0	0.000
FilDecObjExtNbr	0.006	0.081	0	1.000
FilDecStruNbr	0.120	0.713	0	7.000
FilDefObjGlbNbr	1.200	6.843	0	84.000
FilDefRtnNbr	1.833	2.573	0	17.000
FilIncDirNbr	1.713	0.638	0	3.000
FilIncNbr	20.860	2.258	0	29.000
FilLnsNbr	109.360	149.039	0	1166.000
FilLnsSkpSum	0.060	0.264	0	2.000
FilStxErrNbr	0.060	0.264	0	2.000
RtnArgXplSum	24.886	33.812	0	308.000
RtnCalXplNbr	8.880	14.673	0	124.000
RtnCastXplNbr	5.220	17.983	0	208.000
RtnComNbr	0.000	0.000	0	0.000
RtnComVol	0.000	0.000	0	0.000
RtnCplCtlAvg	13.418	10.767	0	50.000
RtnCplCtlMax	30.520	25.184	0	102.000
RtnCplCtlSum	85.693	129.006	0	1226.000
RtnCplCycNbr	11.166	25.862	0	278.000
RtnCplExeAvg	9.568	3.890	0	24.500
RtnCplExeMax	19.480	11.900	0	63.000
RtnCplExeSum	221.733	461.747	0	4417.000
RtnLblNbr	0.000	0.000	0	0.000
RtnLnsNbr	75.240	123.538	0	1049.000
RtnLnsSkpSum	0.066	0.250	0	1.000
RtnScpNbr	12.660	29.377	0	302.000
RtnScpNstLvlAvg	2.099	0.821	0	5.685

The number of observations was 150.

Table 5.17 Summary Statistics of Validation C Dataset (continued)

Variable	Mean	Std Dev	Minimum	Maximum
RtnScpNstLvlMax	3.193	2.129	0	13.000
RtnScpNstLvlSum	39.586	126.467	0	1264.000
RtnStmCtlBrkNbr	1.366	9.936	0	116.000
RtnStmCtlCaseNbr	1.140	8.948	0	104.000
RtnStmCtlCtnNbr	0.053	0.253	0	2.000
RtnStmCtlDfltNbr	0.133	1.156	0	14.000
RtnStmCtlGotoNbr	0.000	0.000	0	0.000
RtnStmCtlIfNbr	6.080	8.810	0	56.000
RtnStmCtlLopNbr	1.980	9.002	0	104.000
RtnStmCtlNbr	13.046	26.036	0	278.000
RtnStmCtlRetNbr	3.413	3.028	0	19.000
RtnStmCtlSwiNbr	0.153	1.180	0	14.000
RtnStmCtlThwNbr	0.000	0.000	0	0.000
RtnStmDecNbr	12.886	25.145	0	284.000
RtnStmDecObjNbr	12.873	25.117	0	284.000
RtnStmDecPrmNbr	6.093	7.313	0	51.000
RtnStmDecRtnNbr	0.013	0.163	0	2.000
RtnStmDecTypeNbr	0.013	0.163	0	2.000
RtnStmExeNbr	21.820	44.847	0	378.000
RtnStmNbr	47.766	92.322	0	940.000
RtnStmNstLvlAvg	1.299	0.668	0	3.544
RtnStmNstLvlSum	95.146	301.626	0	3332.000
RtnStxErrNbr	0.066	0.250	0	1.000
RtnStmXpdNbr	61.000	129.631	0	1238.000
Rchange	5.593	13.914	0	142.000
Rnew	5.046	29.1256	0	329.000
Rdeleted	0.366	1.222	0	8.000
churnchange	2.193	5.496	0	64.000
churnnew	5.360	55.895	0	684.000
churndeleted	0.100	0.880	0	10.000

The number of observations was 150.

Table 5.18 Summary Statistics of Validation Header Dataset

Variable	Mean	Std Dev	Minimum	Maximum
FilComGlbNbr	0.000	0.000	0	0.000
FilComGlbVol	0.000	0.000	0	0.000
FilComTotNbr	0.000	0.000	0	0.000
FilComTotVol	0.000	0.000	0	0.000
FilDecClaNbr	0.000	0.000	0	0.000
FilDecGncTypNbr	0.000	0.000	0	0.000
FilDecGndTypTotNbr	0.000	0.000	0	0.000
FilDecObjExtNbr	5.692	14.510	0	52.000
FilDecStruNbr	1.615	2.844	0	8.000
FilDefObjGlbNbr	0.2302	0.832	0	3.000
FilDefRtnNbr	0.000	0.00	0	0.000
FilIncDirNbr	1.000	1.683	0	6.000
FilIncNbr	2.461	5.269	0	18.000
FilLnsNbr	108.230	129.5103	0	429.000
FilLnsSkpSum	0.000	0.000	0	0.000
FilStxErrNbr	0.000	0.000	0	0.000
Rchange	11.923	26.590	0	99.000
Rnew	14.923	22.001	0	78.000
Rdeleted	12.538	41.923	0	152.000
churnchange	11.384	26.837	0	99.000
churnnew	13.692	22.584	0	78.000
churndeleted	12.153	42.033	0	152.000

The number of observations was 13

the validation procedure, we multiply the raw metrics with the transformed variables got from the PCA as the first step. Then the product is multiplied with the parameter estimates obtained from the regression procedure resulting in a set of $\hat{y}_i$ values. This output is retained for further analysis.

Metric data and the reuse history on modules from the test dataset are input to the fitted model, which gives predictions of the total change for each module of the test dataset. The test dataset consists of the product attributes and the reuse measures of the versions 1.0.5 and actual code-churn of the version 1.0.6. The actual change in source code for each of the i^{th} module, y_i is known, we can validate the predictions, y_{pi} .

We evaluate the predictions using average absolute error AAE and average relative error ARE [19].

$$abserr = |y_{pi} - y_i| \tag{5.15}$$

$$AAE = \frac{1}{n}\sum_{i=1}^{n}|y_{pi} - y_i| \tag{5.16}$$

$$ARE = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{y_{pi} - y_i}{y_i + 1}\right| \tag{5.17}$$

The denominator of ARE has one added to the actual value of code-churn to avoid division by zero. Table 5.19 summarizes the absolute and the relative error of the various datasets.

On evaluating the model, as a conventional quantitative model for all the three datasets, the model was significant, but the quality of predictions was low. AAE and ARE indicate that the model was off by factors of 2 to 5. However, predicting the exact value of a quality

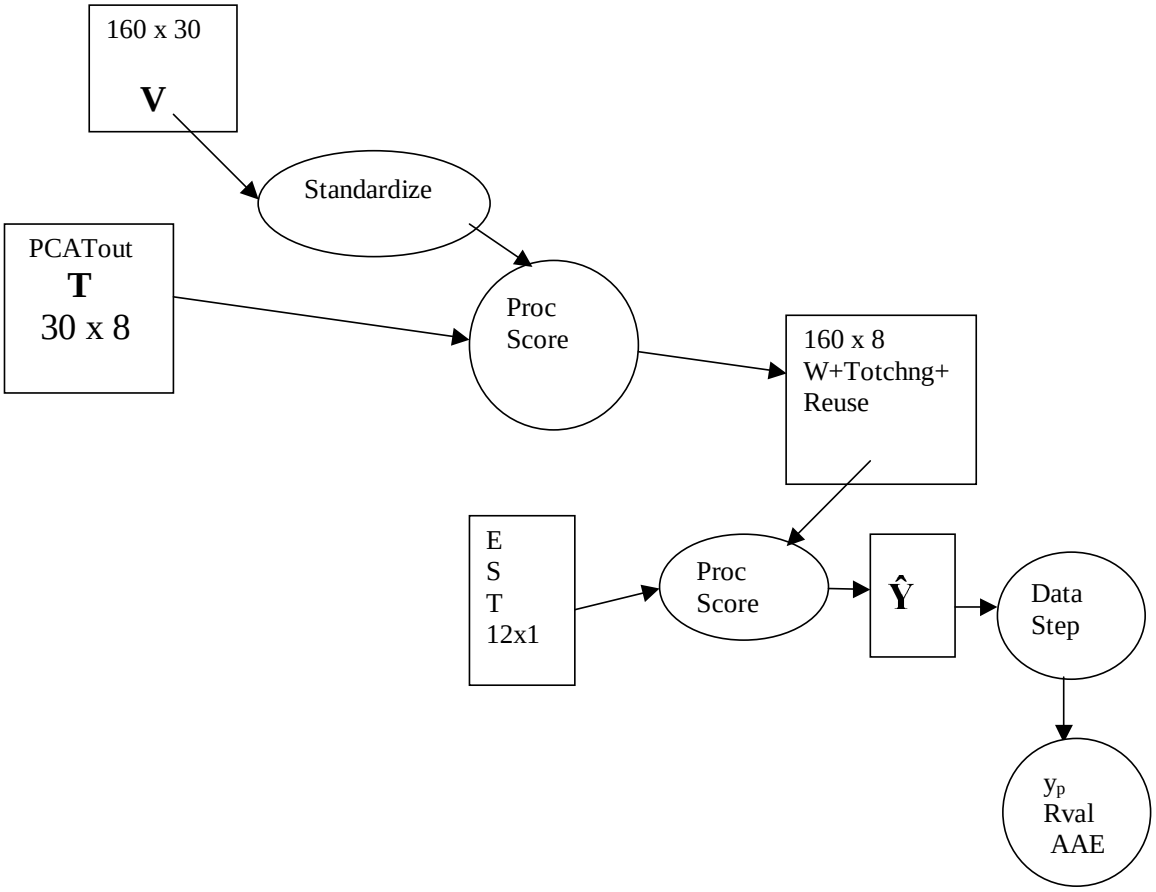


Figure 5.3 Validation

Table 5.19 Absolute and Relative error

DataSet	AAE(lines)	ARE
Total Dataset	10.2	1.99
C Dataset	7.73	1.93
Header Dataset	12.64	4.82

measure for each module is often not necessary, we can come up with a classification model to identify churn-prone and not churn-prone modules. From the predicted list we can classify the modules as churn-prone and not churn-prone [18]. Churn-prone modules are those which are likely to have a large amount of code churn.

A module-order-model [18] is a software-quality model, based on software product metrics and reuse variables, that is used to predict the rank-order of modules according to a quality factor. A module-order model has an underlying quantitative software quality model that predicts the amount of a quantitative quality factor.

We came up with a module-order model for each dataset. For this purpose, we sorted the actual total change (code-churn) y_i and the predicted code-churn y_{pi} in descending order. We took the most churn-prone 20% as the cut off and compared both the sorted lists for common modules (files).

Table 5.20 gives the rank order of the top 20% of the modules in the descending order of actual code-churn y_i for the total dataset. Table 5.21 gives the rank-order of the top 20% of the modules in the descending order of the predicted code-churn y_{pi}. The modules which are in bold are those modules which were predicted to be churn-prone (common in both the lists). We can see that the total dataset model predicted 14 out of the 35 modules to be churn-prone, which is almost 50% of the 35 modules.

The Table 5.24 gives the rank order of the top 20% of the modules in the descending order of actual code-churn for the only C source code and Table 5.25 gives the rank-order of the top 20% of the modules in the descending order of the predicted code-churn. From

Table 5.20 Rank Order of top 20% of Total Dataset with Descending $totchng$

Rank	Filename	$totchng$	y_{pi}	$abserr$
1	**dmp4.h**	329	24.0	305.0
2	**comm_util.c**	158	19.4	9.0
3	**dmpi.c**	124	19.0	5.0
4	**init.c**	109	129.0	20.0
5	isend.c	75	6.0	70.0
6	**mpir.h**	57	49.	8.0
7	**mpirutil.c**	52	33.0	19.0
8	**alltoall.c**	46	13.0	33.0
9	**alltoallv.c**	46	14.0	32.0
10	**dmpiatom.h**	35	32.0	3.0
11	buffree.c	29	7.0	22.0
12	initsend.c	29	9.0	20.0
13	testall.c	26	6.0	20.0
14	bufattach.c	25	6.0	19.0
15	**mpi.h**	24	75.0	51.0
16	opfree.c	23	11.0	12.0
17	bsend_init.c	20	7.5	12.5
18	dmpipk.c	20	11.0	9.0
19	start.c	20	10.0	10.0
20	mpiimpl.h	16	20.0	4.0
21	sbcnst.h	16	4.0	11.0
22	type_struct.c	15	11.0	3.0
23	address.c	14	9.0	5.0
24	testsome.c	14	9.0	5.0

Table 5.20 Rank Order of top 20% of Total Dataset with Descending $totchng$ (continued)

Rank	Filename	$totchng$	y_{pi}	$abserr$
25	recv.c	13	6.0	7.0
26	**reduce.c**	13	20.0	7.0
27	waitsome.c	13	10.0	3.0
28	mpi_bc.h	12	8.0	4.0
29	waitany.c	12	10.0	2.0
30	testany.c	11	7.0	3.0
31	**waitall.c**	11	12.0	1.0
32	**barrier.c**	10	15.0	5.0
33	bsend.c	10	5.0	4.0
34	commreq_free.c	10	9.0	2.0
35	**comm_split.c**	9	16.0	7.0

Table 5.21 Rank Order of top 20% of Total Dataset with Descending y_{pi}

Rank	Filename	$totchng$	y_{pi}	$abserr$
1	**init.c**	109	129.0	20.0
2	**mpi.h**	24	75.0	51.0
3	**mpir.h**	57	49.0	8.0
4	**mpirutil.c**	52	33.0	19.0
5	**dmpiatom.h**	35	33.0	2.0
6	finalize.c	5	32.0	27.0
7	sbcnst.c	2	26.0	24.0
8	ic_create.c	6	25.0	19.0
9	**dmp4.h**	329	24.0	305.0
10	mpiimpl.h	16	20.0	4.0
11	**reduce.c**	13	20.0	7.0
12	**comm_util.c**	158	19.0	138.0
13	error_string.c	9	19.0	10.0
14	**dmpi.c**	124	19.0	105.0
15	comm_dup.c	7	18.0	11.0
16	red_scat.c	2	17.0	15.0
17	ic_merge.c	2	17.0	15.0
18	**comm_split.c**	9	16.0	7.0
19	scan.c	5	15.0	10.0
20	bcast.c	4	15.0	11.0
21	scatterv.c	8	15.0	7.0
22	scatter.c	8	15.0	7.0
23	sendrecv_rep.c	2	15.0	13.0
24	**barrier.c**	10	15.0	5.0

Table 5.21 Rank Order of top 20% of Total Dataset with Descending y_{pi} (continued)

Rank	Filename	$totchng$	y_{pi}	$abserr$
25	gather.c	6	15.0	9.0
26	group_util.c	6	15.0	9.0
27	gatherv.c	6	14.0	8.0
28	**alltoallv.c**	46	13.0	32.0
29	cart_create.c	2	13.0	11.0
30	attr_util.c	2	13.0	11.0
31	graph_create.c	2	13.0	11.0
32	**alltoall.c**	46	13.0	33.0
33	bswap.c	2	13.0	11.0
34	dims_create.c	1	12.0	11.0
35	**waitall.c**	11	12.0	1.0

the table we see that the model for only C source files predicted 13 out of 32 files to be churn-prone, which are shown in bold.

Table 5.22 gives the rank order of the top 20% of the modules in descending order of actual code-churn from the header data file. The Table 5.23 gives the rank order of the top 20% of the modules in the descending order of the predicted code-churn $\hat{y}_i$. Since the header dataset had only 14 observations in the fit dataset and 13 observations in the test dataset, the 20% constituted of 3 modules. The module-order model identified two modules out of the three to be churn-prone.

Table 5.22 Rank Order of top 20% of Header Dataset with Descending on $totchng$

Rank	Filename	$totchng$	y_{pi}	$abserr$
1	**dmp4.h**	329	243.0	86.0
2	mpir.h	57	776.0	19.0
3	**dmpiatom.h**	35	77.0	42.0

Table 5.23 Rank Order of top 20% of Header Dataset with Descending y_{pi}

Rank	Filename	$totchng$	y_{pi}	$abserr$
1	**dmp4.h**	329	243.0	86.0
2	binding.h	2	117.0	115.0
3	**dmpiatom.h**	35	77.0	42.0

Table 5.24 Rank Order of top 20% of C Dataset with Descending $totchng$

Rank	Filename	$totchng$	y_{pi}	$abserr$
1	**comm_util.c**	158	21.0	137.0
2	**dmpi.c**	124	39.0	85.0
3	**init.c**	109	116.0	7.0
4	isend.c	75	6.0	69.0
5	**mpirutil.c**	52	42.0	10.0
6	alltoall.c	46	10.0	35.0
7	**alltoallv.c**	46	12.0	34.0
8	buffree.c	29	4.0	25.0
9	initsend.c	29	6.0	22.0
10	testall.c	26	5.0	21.0
11	bufattach.c	25	5.0	20.0
12	opfree.c	23	4.0	19.0
13	bsend_init.c	20	3.5	16.0
14	**dmpipk.c**	20	26.0	6.0
15	start.c	20	6.0	14.0
16	**type_struct.c**	15	13.0	3.0
17	address.c	14	4.0	10.0
18	testsome.c	14	2.0	12.0
19	recv.c	13	5.0	7.0
20	**reduce.c**	13	17.0	4.0
21	waitsome.c	13	5.0	8.0
22	waitany.c	12	4.0	7.0

Table 5.24 Rank Order of top 20% of C Dataset with Descending $totchng$ (continued)

Rank	Filename	$totchng$	y_{pi}	$abserr$
23	testany.c	11	1.0	10.0
24	waitall.c	11	0.0	10.0
25	**barrier.c**	10	12.0	2.0
26	bsend.c	10	4.0	6.0
27	commreq_free.c	2	6.0	4.0
28	**comm_split.c**	9	11.0	2.0
29	**error_string.c**	9	13.0	3.0
30	send.c	9	7.0	2.0
31	**scatter.c**	8	14.0	6.0
32	**scatterv.c**	8	14.0	6.0

Table 5.25 Rank Order of top 20% of C Dataset with Descending y_{pi}

Rank	Filename	$totchng$	y_{pi}	$abserr$
1	**init.c**	109	116.0	7.0
2	**mpirutil.c**	52	41.0	10.0
3	bswap.c	2	41.0	39.0
4	**dmpi.c**	124	38.0	85.0
5	**dmpipk.c**	20	26.0	6.0
6	sbcnst.c	2	26.0	24.0
7	attr_util.c	2	23.0	21.0
8	group_util.c	6	22.0	17.0
9	**comm_util.c**	158	21.0	137.0
10	ic_create.c	6	18.0	12.0
11	finalize.c	5	18.0	13.0
12	**reduce.c**	13	17.0	4.0
13	bcast.c	4	15.0	11.0
14	sendrecv_rep.c	2	15.0	13.0
15	mperror.c	5	14.0	9.0
16	**scatterv.c**	8	14.0	6.0
17	**scatter.c**	8	14.0	6.0
18	util_hbt.c	2	14.0	12.0
19	gather.c	6	13.0	7.0
20	topo_util.c	1	13.0	12.0
21	gatherv.c	6	13.0	7.0

Table 5.25 Rank Order of top 20% of the C Dataset with Descending y_{pi} (continued)

Rank	Filename	$totchng$	y_{pi}	$abserr$
22	**error_string.c**	9	12.0	3.0
23	**type_struct.c**	15	12.0	2.0
24	**barrier.c**	10	12.0	2.0
25	**alltoallv.c**	46	12.0	33.0
26	comm_dup.c	7	12.0	5.0
27	pack.c	2	12.0	10.0
28	group_rexcl.c	2	12.0	10.0
29	**comm_split.c**	9	11.0	2.0
30	red_scat.c	2	11.0	9.0
31	graph_create.c	2	11.0	9.0
32	cart_create.c	2	11.0	9.0

Table 5.26 Module-Order Model Performance

DataSet	Case	G	$\widehat{G}$	ϕ
Total Dataset	Model	1483	653.0	0.75
Total Dataset	No Model	1483	364.0	0.24
Only C Dataset	Model	987	641.0	0.64
Only C Dataset	No Model	987	332.7	0.22
Header Dataset	Model	421	366	0.86
Header Dataset	No Model	421	90.3	0.21

Table 5.26 summarizes the model performance of the module-order model for all the three datasets, when compared to random selection (no model). Using the notation in [18], in Table 5.26, G is the sum of the *totchng* variable for all the 35 files, 20% of 175 when the modules are sorted in descending order of the actual change y_i, $\widehat{G}$ is the sum of the total change when the files are sorted in descending order of y_{pi} the predicted value of total change given by the model.

The value of G for the total data file was 1483 lines and the value of $\widehat{G}$ was 653.60 lines. The ratio $\phi = \widehat{G}/G = 0.75$. The value of ϕ is a measure of the performance of the module-order model. The value of ϕ for a perfect model would be one [18].

Consider the case when there is no quality model. The actual change was G_o. $\widehat{G}_o$ was the average of the total change multiplied by the total number of file considered. Then for the threshold that we have chosen, i.e, 20%, the value of $\widehat{G}_o$ was 364.0. The value ϕ_o was 0.24, which is much lower than value than the value of ϕ of the module-order model. The probability of finding a churn-prone file is higher when the model is used than when the model is not used.

For the data collected from the C source files, the value is G is 987 and $\widehat{G}$ is 641.0. The performance of the module order model ϕ is 0.64. The ratio of actual churn to total churn in the case when there is no model would have been 0.22. For the header file the value of G is 421 and the predicted $\widehat{G}$ is 366. The model performance ϕ is 0.86. Without the model, the ratio would be ϕ = 0.21.

From the above approach of analysis, we can say that the module-order model would have been useful in predicting churn-prone modules

CHAPTER VI

ANALYSIS

This chapter addresses the research questions. Also the various internal and external constraints and limitations of the research are discussed. Each section heading is a paraphrase of a research question followed by conclusions drawn from the results of the research, with relevance to that question.

6.1 Is quality assessment feasible using code-churn between versions?

> 1. How can the quality of open-source software MPICH be assessed using changes in source code between consecutive versions?

We were able to measure added, changed, and deleted lines from the configuration management data. We did not need to use intrusive measurement methods to assess quality.

6.2 What are the software attributes that are indicators of quality for MPICH?

> 2. What are software attributes derived from software metrics which are indicators of the quality of MPICH?

From the statistical analysis that we performed on the data, the following are the factors that were chosen by regression for the model for the total data file:

1. *FACTOR 3* was associated with function definitions, parameter declarations in the routine, scope of nesting level and the number of control if structures.
2. *FACTOR 7* was associated with the `extern` declarations.

With respect to software engineering in practice, as the number of function definitions increased the code-churn tended to increase. The number of `if` control structures and nesting level of control structures contributed to the code-churn. Function parameters and `extern` declarations are often interfaces between files. Changes to these declarations also contributed to code-churn.

The software attributes that were chosen for the model for the C source file data were:

1. *FACTOR 2* was associated with function definitions, parameter declarations in the routine, scope of nesting level and the number of control `if` structures.
2. *FACTOR 4* was associated with variable declarations, the number of explicit calls to functions and the sum of the explicit arguments passed to other functions..

Similar to the above observations, *FACTOR 2* was associated with the same software concepts as the *FACTOR 3* in total dataset. Because *FACTOR 4* was chosen for the model, changes to variable declarations and explicit arguments passed contributed to code-churn. These indicate the amount of different data the software was using.

The software attributes that were chosen for the model for the header file data were:

1. The number of file includes.
2. Size of the file.
3. Number of structure declarations in the file.
4. The number of new lines.

As the number of new lines increased, code-churn tended to increase, indicating that new code tended to need some modifications in the next release. Structure declarations, which are associated with major complex data structures, were associated with code-churn. File includes, which indicate the extent of interfaces were also associated with code-churn. The number of lines in the file is a measure of size. Larger files naturally have more code that might be changed.

In the case study, we were able to measure the added, deleted and new lines from the changes in source code between versions. From the statistical results obtained for the total data file and C source file data, we have seen that none of the reuse variables were selected for the model either for the total dataset or the C dataset. For the header file data, the number of new lines was selected for the model. However, the number of header files was small, limiting the accuracy of the model. From the above observations we conclude that the role of reuse variables was not very significant in predicting code-churn.

6.3 Is a quality model feasible for MPICH?

> 3. Is a quality model feasible which gives an understanding of quality and is suitable for the open-source software MPICH ?

Statistically significant quality models were feasible for MPICH using multiple regression. However, the accuracy of predictions by the model was poor.

Using the module-order model we could predict about 50% of the top 20% of churn-prone modules. This is a significant improvement over random selection.

6.4 Threats to Validity

The following two sections summarize the internal and external threats to the validity of the models.

6.4.1 Threats to Internal Validity

Threats to internal validity are those issues which are limited to the subject of case study-MPICH.

- It was observed from the pattern of the residuals of fit data that the underlying quantitative model is not perfectly linear. A non-linear model could be more appropriate.
- The model developed for the header files was built from a dataset of only 14 observations and hence the model may not be robust.
- The changes in source code was measured using the `diff` utility of CVS and the changed and new lines of the second version and the deleted lines from the first version were counted.
- Blank lines and comment lines were included in the count for reuse variables and churn variables.

The above mentioned issues have caused the error percentage in the models to be high. If these issues are properly dealt, then the models can be more accurate.

6.4.2 Threats to External Validity

Threats to external validity are those issues concerning generalization of modeling results.

- A case study is inherently restricted to the system under study. Results may not generalize to other systems.
- The transformation matrix from the principal components analysis presented in the thesis is applicable only to the current case study of MPICH.

- All of the raw metrics from the data files and the factors from the principal components analysis are pertinent to the current case study and may not be applicable to other software.

CHAPTER VII

CONCLUSIONS

7.1 Evaluation of Hypothesis

The research hypothesis is the following:

> By studying the data available in the form of past versions of the MPICH, a methodology can be developed that will enable quality assessment of the software.

Supporting the above hypothesis, a methodology which uses data from past releases of MPICH was developed. Quality is broad topic and it has many factors contributing to it. All these factors need to be balanced in a high quality software product. Correctness is a quality factor that is often measured by the number of defects per lines of code. Changes are made to the source code in order to fix bugs and to fine tune functions. This activity of making changes to the code is quantified as code-churn. For this thesis, a model which can predict the amount of code-churn for MPICH was developed. This case study resulted in predictions that could have been useful for quality assessment. Thus, the hypothesis was supported.

The following is a summary of the case study results. Data was collected from the four releases of MPICH. The data collected consisted of reuse measurements (independent variables), product attributes (independent variables) and code-churn measurements

(dependent variable). Fit and test datasets consisting of the mentioned measurements were collected from the four versions of MPICH. Each fit and test dataset in turn consisted of a total dataset, a C dataset, and a header dataset. The total dataset was the combination of the C dataset and the header dataset. Statistical analysis was performed on each fit dataset to build a model. The following product attributes were found an to have effect on code-churn.

- Total Dataset
  - Number of function definitions
  - Number of `extern` object declarations
- C dataset
  - Number of function declarations
  - Number of variable declarations
- Header dataset
  - Number of structure declarations
  - Number of file includes.
  - Size of the file
  - Number of new lines

A test dataset was used to validate each model. The results of the validation are discussed in section5.5. Using each test dataset, modules were sorted in descending order of predicted code-churn. A module-order model was built that predicted about 50% of the most churn-prone modules. The model was significant when compared to random selection.

7.2 Contributions

The contribution of this thesis is the methodology to obtain a quality assessment of software from prior releases. For a recent version of a software product, quality can be predicted. For example, suppose that the year is 1995 and the release 1.0.7 of MPICH has been released. If a user is considering using version 1.0.7, then the question is the risk of bugs in C modules? Given the quality model built in our case study, here are the steps one should follow. The transformation matrix **T** from principal components analysis is available and estimated regression coefficients a_j are available:

1. Measure the Product attributes of the C source modules using DATRIX. Obtain a product metrics dataset, **C**.
2. The transformation matrix **T** and the standardized dataset **C** should be matrix-multiplied, resulting in a factors matrix.
3. The factors matrix should be multiplied with the regression coefficients vector to obtain the predicted code-churn.
4. The user then has an idea of the amount of code-churn predicted for the module.
5. From the above information, the user can decide if the modules of interest are likely to have bugs in the near future. This decision will help the user whether to stay with the existing system (version 1.0.6) with known bugs or to get the update (version 1.0.7) and unknown bugs.

Similarly, other open-source software can use the methodology and make better judgements regarding usage of the software.

7.3 Further Research

To use the methodology on other open-source software would be interesting future work. A refinement of the quantitative model presented in this thesis, in terms of reduction

in error would be useful. It was recommended by an MPICH insider, that a technique that can omit comment lines from the count into changes lines, especially for software with license information would be beneficial. Also a classification model that can predict the code-churn because of fixing bugs, code-churn because of functional enhancement and fine tuning of source code separately for a software product will be very useful. Classification models for other open-source software using the same methodology is also suggested.

REFERENCES

[1] *Teach Yourself Perl in 21 Days*, Techmedia, Delhi, 1999.

[2] E. B. Allen, "CAREER: Assessment of Open Source Software for High-Performance Computing," *Proposal to National Science Foundation*, Mar. 2001.

[3] V. R. Basili, F. Lanubile, and F. Shull, "Building Knowledge Through Families," *IEEE Transactions on Software Engineering*, vol. 25, no. 4, Aug. 1999, pp. 456–473.

[4] *Datrix Metric Reference Manual*, version 4.1 edition, Bell Canada, Montreal, Quebec, Canada, May 2001.

[5] P. Cederquist, *Concurrent Version System*, http://www.cvshome.org.

[6] M. K. Daskalantonakis, "A Practical View of Software Maintenance and Implementation Experiences," *IEEE Transactions on Software Engineering*, vol. 18, no. 11, Nov. 1992, pp. 998–1010.

[7] D. Dunston, "Open Source Enterprise," *eWeek*, vol. 19, no. 27, July 2002, pp. 35–43.

[8] J. Feller and B. Fitzgerald, *Understanding Open Source Software Development*, Pearson Education Limited, Edinburgh Gate, Harlow, London, 2002.

[9] J. Feller, B. Fitzgerald, F. Hecker, S. Hissam, K. Lakhani, and A. van der Hoek, "Meeting Challanges and Surviving Success: The Second Workshop on Open Source Software Engineering.," *Software Engineering Notes*, vol. 27, no. 5, Sept. 2002, pp. 69–71.

[10] R. A. Fink, "Reliability Modeling of Freely-Available Internet-Distributed Software," *Proceedings: Fifth International Symposium on Software Metrics*, Bethesda, Maryland, Nov. 1998, pp. 101–104, IEEE Computer Society.

[11] R. B. Grady, "Successfully Applying Software Metrics," *Computer*, vol. 27, no. 9, Sept. 1994, pp. 18–25.

[12] W. Gropp, E. Lusk, A. Skjellum, and N. Doss, *A High–Performance,Portable Implementation of the Message Passing Interface Standard*, http://www-unix.mcs.anl.gov/mpi/mpich/.

[13] W. Groppp, E. Lusk, N. Doss, and A. Skjellum, "A High Performance, Portable Implementation of the MPI Message Passing Interface," *Parallel Computing*, vol. 22, no. 6, Sept. 1996, pp. 789–828.

[14] *Statistical Analysis System*, 4th edition, http://www.sas.com, Cary, North Carolina, Sept. 1991.

[15] W. S. Humphrey, T. R. Synder, and R. R. Willis, "Software Process Improvement at Hughes Aircraft," *IEEE Software*, vol. 8, no. 4, July 1991, pp. 11–23.

[16] D. R. Jeffrey and M. Berry, "A Framework for Evaluation and Prediction of Metrics Program Success," *Proceedings: First International Software Metrics Symposium*, Baltimore, Maryland, May 1993, pp. 28–39.

[17] D. R. Jeffrey and L. Votta, "Guest Editor's Special Section Introduction," *IEEE Transactions on Software Engineering*, vol. 25, no. 4, July 1999, pp. 435–437.

[18] T. M. Khoshgoftaar and E. B. Allen, "Ordering Fault-Prone Software Modules," *Software Quality Journal*, vol. 11, no. 1, Mar. 2003, pp. 19–37.

[19] T. M. Khoshgoftaar, E. B. Allen, W. D. Jones, and J. P. Hudepohl, "Data Mining of Software Development Databases," *Software Quality Journal*, vol. 9, no. 3, Nov. 2001, pp. 161–176.

[20] B. Kitchenham, S. L. Pfleeger, L. Pickard, P. Jones, D. Hoaglin, K. El Emam, and J. Rosenberg, "Preliminary Guidelines for Empirical Research in Software Engineering," *IEEE Transactions on Software Engineering*, vol. 28, no. 8, Aug. 2002, pp. 721–733.

[21] T. Lawrie and C. Gacek, "Issues of Dependability in Open Source Software Development," *Software Engineering Notes*, vol. 27, no. 3, May 2002, pp. 34–37.

[22] I. Myrtveit and E. Stensurd, "A Controlled Experiment to Assess the Benefits of Estimating with Analogy and Regression Models," *IEEE Transactions on Software Engineering*, vol. 25, no. 6, July 1999, pp. 510–524.

[23] D. Paulish and A. Carleton, "Case Studies of Software Process Improvement Measurement," *Computer*, vol. 27, no. 9, Sept. 1994, pp. 50–57.

[24] S. L. Pfleeger, "Assessing Measurement," *IEEE Software*, vol. 14, no. 2, Mar. 1997, pp. 25–26, Editor's introduction to special issue.

[25] R. Pressman, *Software Engineering A Practitioners Approach*, McGraw-Hill, New York, 2001.

[26] D. G. Rees, *Foundations of Statistics*, Chapman and Hall, New York, 1993.

[27] C. B. Seaman, "Qualitative Methods in Empirical Studies of Software Engineering," *IEEE Transactions on Software Engineering*, vol. 25, no. 4, July 1999, pp. 571–572.

[28] S. S. Shapiro and A. J. Gross, *Statistical Modeling Techniques*, 2nd edition, Marcel Dekker,Inc, New York, 1981.

[29] J. Singer and N. G. Vinson, "Ethical Issues in Empirical Studies of Software Engineering," *IEEE Transactions on Software Engineering*, vol. 28, no. 12, Dec. 2002, pp. 1171–1179.

[30] M.-W. Wu and Y.-D. Lin, "Open Source Software Developement: An Overview," *Computer*, vol. 34, no. 6, June 2001, pp. 33–38.

APPENDIX

SOURCE CODE OF ALL SCRIPTS

For the purpose of building a linear regression model, the `Proc Reg` procedure from SAS was used. The following is the sas code for performing linear regression.

```
title ’Multiple Linear Regression’;
title2 ’Fit data set =thesis.onlycfile’;

 proc reg
 data=thesis.onlycfilepca
 outest=thesis.EST
 simple
 ;
yhat: model totchng= FACTOR1 FACTOR2 FACTOR3 FACTOR4 FACTOR5 FAC-
TOR6 FACTOR7
FACTOR8 Rchange Rnew Rdel
 /selection= STEPWISE
 slentry=0.15
 slstay=0.15
 ;
 output out = REGFIT
 predicted=yhat
 residual=R
 press=PRESS
 rstudent=RST
 ;
 plot residual.*predicted.
 ;
 run;

data;
set REGFIT;
if (yhat<0) then Yp=0;
else
Yp=yhat;
R=totchng-Yp;
SE=R*R;
abserr=abs(R);
relerr=abs(R/(totchng+1));

proc univariate plot normal;
var R;

proc univariate;
var
 SE
```

```
 Yp
 PRESS
 RST
 abserr
 relerr
 ;
 run;
/*************** VALIDATION PROCEDURE BEGINS HERE ***************/
title 'Multiple Linear Regression';
title2 'Testdataset = thesis.validationdatafile';

data thesis.Cvalidatafilepca;
set thesis.validatafile1;
 if FLAG = 0;



data thesis.Cvalidatafilepca;
set  scor;
totchng=churnchange+churnnew+churndel;
run;

proc score
  data =thesis.Cvalidatafilepca
  score=thesis.onlycfileTout
  out = scor
  ;
  var
FilDecObjExtNbr FilDecStruNbr FilDefObjGlbNbr FilDefRtnNbr
FilIncDirNbr FilIncNbr FilLnsNbr RtnArgXplSum RtnCalXplNbr
RtnCastXplNbr RtnCplCtlAvg RtnCplCtlMax RtnCplCtlSum
RtnCplCycNbr RtnCplExeAvg RtnCplExeMax RtnCplExeSum
RtnLnsNbr RtnScpNbr RtnScpNstLvlAvg  RtnStmCtlRetNbr
RtnStmCtlSwiNbr RtnStmDecObjNbr RtnStmDecPrmNbr RtnStmDecRtnNbr
RtnStmDecTypeNbr RtnStmExeNbr RtnStmNbr RtnStmNstLvlAvg
RtnStmNstLvlSum RtnStmXpdNbr RtnStmCtlCtnNbr RtnStmCtlDfltNbr
RtnStmCtlIfNbr RtnScpNstLvlMax RtnScpNstLvlSum  RtnStmCtlCaseNbr
;
run;

data thesis.estimates;
set thesis.EST;
run;
```

```
title3 ’prediction analysis’;

proc score
 data=scor
 score=thesis.est
 out=testpred
 type=parms
 predict
 ;
 var
 FACTOR2 FACTOR4 FACTOR7
 ;

 data thesis.mypred;
  set testpred;

data PRED;
 set testpred;
 if (yhat<0) then Yp=0;
else
Yp=yhat;
R=totchng-Yp;
SE=R*R;
abserr=abs(R);
relerr=abs(R/(totchng+1));

proc sort data= PRED
 out= thesis.sortedCPRED;
 by descending totchng;
 run;
 proc print data=thesis.sortedCPred;
 var Filename totchng yhat R Abserr;
 run;
 proc sort data= PRED
 out= thesis.sortedCPRED;
 by descending yhat;
 run;
 proc print data=thesis.sortedCPred;
 var Filename totchng yhat R Abserr;
 run;

proc plot;
 plot R*Yp;
```

```
 proc univariate plot normal;
 var R;

 proc univariate;
 var
  Yp
  SE
  abserr
  relerr
  ;
  run;



title ’ calculate principal component analysis for collfilemets’;
 title2 ’dataset = thesis.test’;

  proc factor
   data=thesis.totdatafile
   outstat=thesis.totdatafileTout
   all
   method=principal
   /*stopping rules*/
   /*nfactors =4*/
   /*proportion=0.95*/
   mineigen=1
   score
   rotate=VARIMAX
   reorder
   ;
   var
FilDecObjExtNbr$ FilDecStruNbr$$ FilDefObjGlbNbr$ FilDefRtnNbr$
FilIncDirNbr$ FilIncNbr$ FilLnsNbr$ RtnArgXplSum RtnCalXplNbr$
RtnCastXplNbr$ RtnCplCtlAvg RtnCplCtlMax RtnCplCtlSum
RtnCplCycNbr RtnCplExeAvg RtnCplExeMax RtnCplExeSum RtnLnsNbr
RtnScpNbr RtnScpNstLvlAvg RtnStmCtlNbr RtnStmCtlRetNbr
RtnStmCtlSwiNbr RtnStmDecObjNbr RtnStmDecPrmNbr
RtnStmDecRtnNbr RtnStmDecTypeNbr RtnStmExeNbr RtnStmNbr
RtnStmNstLvlAvg RtnStmNstLvlSum RtnStmXpdNbr RtnStmCtlBrkNbr
RtnStmCtlCtnNbr RtnStmCtlDfltNbr RtnStmCtlIfNbr
RtnStmCtlLopNbr RtnScpNstLvlMax RtnScpNstLvlSum RtnStmCtlCaseNbr
   ;
   title3 ’PCA of & variables’;
   run;
proc score
```

```
    data=thesis.onlycfile
    score=thesis.onlycfileTout
    out=scor
    ;
    var
FilDecObjExtNbr FilDecStruNbr FilDefObjGlbNbr FilDefRtnNbr
FilIncDirNbr FilIncNbr FilLnsNbr RtnArgXplSum RtnCalXplNbr
RtnCastXplNbr RtnCplCtlAvg RtnCplCtlMax RtnCplCtlSum
RtnCplCycNbr RtnCplExeAvg RtnCplExeMax RtnCplExeSum
RtnLnsNbr RtnScpNbr RtnScpNstLvlAvg  RtnStmCtlRetNbr
RtnStmCtlSwiNbr RtnStmDecObjNbr RtnStmDecPrmNbr RtnStmDecRtnNbr
RtnStmDecTypeNbr RtnStmExeNbr RtnStmNbr RtnStmNstLvlAvg
RtnStmNstLvlSum RtnStmXpdNbr RtnStmCtlCtnNbr
RtnStmCtlDfltNbr RtnStmCtlIfNbr
RtnScpNstLvlMax RtnScpNstLvlSum  RtnStmCtlCaseNbr
    ;
    run;

    data thesis.onlycfilepca;
    set scor;
    totchng=churnchange+churnnew+churndel;
    run;
```

Given below is the SAS procedure for finding the correlation among the raw metrics. The step `data` gives the path of the input data file to be worked on, here totdatafile is the name of the input data file.

```
proc corr
 data=thesis.totdatafile;
 run;
```

The following is the procedure for finding descriptive statistics of the measured product attributes, reuse variables and code-churn variables using SAS.

```
proc means
 data=thesis.Totdatafile;
 run;
```

The following is the SAS data step for splitting the total dataset into two datasets:

```
data thesis.onlycfile;
 set thesis.totdatafile;
 if FLAG = 0;

data thesis.onlyhfile;
 set thesis.totdatafile;
 if FLAG = 1;
```

In the above procedure the statement `data` specifies the output dataset to which the output should be written. The statement `set` specifies the input dataset. The `if` specifies the condition based on which the dataset should be split. Here in the above procedure the all the observations which have the `Flag` variable set to zero are C modules and all the observations whose `Flag` variable is set to one are header file observations. When the data step is run SAS creates a new dataset with all the observations containing the `Flag` variable turned to zero and keeps the the original dataset intact, it does not delete the observations copied to the new dataset. Similarly in the second datastep specified above SAS creates a new dataset containing all the observations which have the `Flag` variable turned to one. At the end of the two data steps we have three datasets, one is the dataset with only C observations, second with only header file observations and third with both the observations which is the total data file.

The step in which we sum the *churnchange* , *churnnew* and *churndeleted* variables collected from the changes in source of the test dataset are summed to get code-churn is also done in SAS. The following is the data step which does the calculation:

```
data thesis.Cvalidatafilepca;
```

```
set  scor;
totchng=churnchange+churnnew+churndel;
run;
```

The following is a Unix script, the step Collect Metrics in the flow diagram. The script preprocesses every `*.C` file in the directory and runs DATRIX on the preprocessed file and collects the routine and files metrics in separate files.

```
#! /bin/sh
 for cfile in *.c
 do
 gcc -E -o output.i -I/home/oshpc/oshpc/mpichapr/include $cfile;
 echo enter ouput filename;
 read ofile;
 cp output.i $ofile;
 done
 for ofile in *.i
 do
 dxmetc $ofile -rmet  > metrics.txt;
 done
```

```
Index: address.c
===================================================
RCS file: /home/oshpc/cvsroot/cvsroot/mpichapr
/context/pt2pt/address.c,v
retrieving revision 1.3
retrieving revision 1.4
diff -c -b -B -C0 -r1.3 -r1.4
*** address.c 2003/04/11 14:52:20 1.3
--- address.c 2003/04/11 15:05:22 1.4
***************
*** 2 ****
!* $Id: address.c,v 1.3 2003/04/11 14:52:20 oshpc Exp $
--- 2 ----
!*$Id: address.c,v 1.4 2003/04/11 15:05:22 oshpc Exp $
***************
*** 5 ****
!*All rights reserved.  See COPYRIGHT in top-level directory.
--- 5 ----
! * See COPYRIGHT in top-level directory.
***************
```

```
*** 8 ****
! #include "mpi.h"
--- 8,12 ----
! #ifndef lint
! static char vcid[] ="$Id: address.c,v 1.4 2003/04/11 15:05:22
! #endif /* lint */
!
! #include "mpiimpl.h"
***************
*** 12 ****
!     MPI_Address - Get the address of a location in memory
--- 16 ----
!     MPI_Address - Gets the address of a location in memory
***************
*** 20 ****
!     This routine is provided primarily for the For-
tran programmer.
--- 25,29 ----
!  This routine is provided for both the For-
tran and C programmers.
!  On many systems, the address returned by this rou-
tine will be the same
!  as produced by the C '&' operator, but this is not re-
quired in C and
!  may not be true of systems with word- rather than byte-
oriented
!  instructions or systems with segmented address spaces.
***************
*** 26 ****
!     *address = (int) location - MPI_BOTTOM;
--- 35 ----
!     *address = (int) ((char *)location - (char *)MPI_BOTTOM);
```

Script for formatting the metrics.

```
#!/usr/bin/perl -w
format RTNMET_TOP =
"FILE NAME", "FUNCTION NAME","RtnArgXplSum","RtnCalXplNbr",
"RtnCastXplNbr","RtnComNbr","RtnComVol", "RtnCplCtlAvg",
"RtnCplCtlMax","RtnCplCtlSum","RtnCplCycNbr","RtnCplExeAvg",
"RtnCplExeMax","RtnCplExeSum", "RtnLblNbr","RtnLnsNbr",
"RtnLnsSkpSum","RtnScpNbr","RtnScpNstLvlAvg",
"RtnScpNstLvlMax","RtnScpNstLvlSum", "RtnStmCtlBrkNbr",
"RtnStmCtlCaseNbr","RtnStmCtlCtnNbr","RtnStmCtlDfltNbr",
"RtnStmCtlGotoNbr","RtnStmCtlIfNbr","RtnStmCtlLopNbr",
```

```
 "RtnStmCtlNbr","RtnStmCtlRetNbr","RtnStmCtlSwiNbr",
 "RtnStmCtlThwNbr","RtnStmDecNbr", "RtnStmDecObjNbr",
 "RtnStmDecPrmNbr","RtnStmDecRtnNbr","RtnStmDecTypeNbr",
 "RtnStmExeNbr","RtnStmNbr","RtnStmNstLvlAvg",
 "RtnStmNstLvlSum","RtnStxErrNbr","RtnStmXpdNbr"
.


 format RTNMET =
 @<<<<<<<<<,@<<<<<<<<<<,@###.#,@###.##,@###.##,@###.##,@###.##,@###.#,
 $FILE_NAME $SCOPED_NAME$RtnArgXplSum $RtnCalXplNbr $RtnCastXplNbr
 $RtnComNbr $RtnComVol $RtnCplCtlAvg  $RtnCplCtlMax $RtnCplCtlSum
 $RtnCplCycNbr $RtnCplExeAvg $RtnCplExeMax $RtnCplExeSum $RtnLblNbr
 $RtnLnsNbr $RtnLnsSkpSum $RtnScpNbr $RtnScpNstLvlAvg $RtnScpNstLvlMax
 $RtnScpNstLvlSum $RtnStmCtlBrkNbr $RtnStmCtlCaseNbr $RtnStmCtlCtnNbr
 $RtnStmCtlDfltNbr $RtnStmCtlGotoNbr  $RtnStmCtlIfNbr$RtnStmCtlLopNbr
 $RtnStmCtlNbr $RtnStmCtlRetNbr $RtnStmCtlSwiNbr $RtnStmCtlThwNbr
 $RtnStmDecNbr $RtnStmDecObjNbr $RtnStmDecPrmNbr $RtnStmDecRtnNbr
 $RtnStmDecTypeNbr $RtnStmExeNbr $RtnStmNbr $RtnStmNstLvlAvg
 $RtnStmNstLvlSum $RtnStmXpdNbr $RtnStxErrNbr
.

  @array=('RtnArgXplSum','RtnCalXplNbr','RtnCastXplNbr','RtnComNbr',
  'RtnComVol','RtnCplCtlAvg','RtnCplCtlMax','RtnCplCtlSum',
  'RtnCplCycNbr','RtnCplExeAvg','RtnCplExeMax','RtnCplExeSum',
'RtnLblNbr','RtnLnsNbr','RtnLnsSkpSum','RtnScpNbr',
'RtnScpNstLvlAvg','RtnScpNstLvlMax','RtnScpNstLvlSum','RtnStmCtlBrkNbr',
'RtnStmCtlCaseNbr','RtnStmCtlCtnNbr','RtnStmCtlDfltNbr','RtnStmCtlGotoNbr',
'RtnStmCtlIfNbr','RtnStmCtlLopNbr','RtnStmCtlNbr','RtnStmCtlRetNbr',
'RtnStmCtlSwiNbr','RtnStmCtlThwNbr','RtnStmDecNbr','RtnStmDecObjNbr',
'RtnStmDecPrmNbr','RtnStmDecRtnNbr','RtnStmDecTypeNbr','RtnStmExeNbr',
'RtnStmNbr','RtnStmNstLvlAvg','RtnStmNstLvlSum',
    'RtnStmXpdNbr','RtnStxErrNbr');


  open (RTN,"$ARGV[0]");
  open(RTNMET,">>$ARGV[1]");
  open(MET,">$ARGV[2]");


  @rtnmetrics= <RTN>;
  #%hash_pattern=grep { $_=~/FILE_NAME/} @rtnmetrics;
  @array_pattern=grep { $_=~/FILE_NAME/} @rtnmetrics;
  @function=grep {$_=~/MANGLED_NAME/} @rtnmetrics;
  print @array_pattern;
  $size=@array_pattern;
  print "$size \n ";
  $size_of_function=@function;
  ####### ****** THIS IS  THE BEGINNING OF THE MAIN FOR LOOP **********
  for ($l=0,$m=0;$l<$size, $m<=$size_of_function; $l++, $m++)
```

```
{
($pattern1,$pattern2,$FILE_NAME)= split(/\s+/,$array_pattern[$l]);
($pattern1,$pattern2,$SCOPED_NAME)=split(/\s+/,$function[$m]);

$size_of_routine=@routine;
@routine1= grep{$_=~/RtnArgXplSum/} @rtnmetrics;
$size_of_routine1=@routine1;

for($i=0;$i<=$size_of_routine1;$i++)
{
($pattern1,$pattern2,$metric1[$i]) = split(/\s+/, $routine1[$i]);
}
$size_of_metric=@metric1;

$RtnArgXplSum = $metric1[$m];

@routine2=grep{$_=~/RtnCalXplNbr/} @rtnmetrics;
$size_of_routine2=@routine2;
#print MET @routine2;
for($i=0;$i<=$size_of_routine2;$i++)
{
($pattern1,$pattern2,$metric2[$i]) = split(/\s+/, $routine2[$i]);
}
$size_of_metric=@metric2;
#print MET @metric2;
$RtnCalXplNbr=$metric2[$m];
print MET $RtnCalXplNbr;
@routine3= grep{$_=~/RtnCastXplNbr/} @rtnmetrics;
$size_of_routine3=@routine3;
for($i=0;$i<=$size_of_routine3;$i++)
{
($pattern1,$pattern2,$metric3[$i]) = split(/\s+/, $routine3[$i]);
}
$size_of_metric3=@metric3;

$RtnCastXplNbr=$metric3[$m];

@routine4= grep{$_=~/RtnComNbr/} @rtnmetrics;
$size_of_routine4=@routine4;
for($i=0;$i<=$size_of_routine4;$i++)
{
($pattern1,$pattern2,$metric4[$i]) = split(/\s+/, $routine4[$i]);
}
$size_of_metric=@metric4;

$RtnComNbr=$metric4[$m];

@routine5= grep{$_=~/RtnComVol/} @rtnmetrics;
$size_of_routine5=@routine5;
for($i=0;$i<=$size_of_routine5;$i++)
```

```
{
($pattern1,$pattern2,$metric5[$i]) = split(/\s+/, $routine5[$i]);
}
$size_of_metric=@metric2;
$RtnComVol=$metric5[$m];

@routine6= grep{$_=~/RtnCplCtlAvg/} @rtnmetrics;
$size_of_routine6=@routine6;
for($i=0;$i<=$size_of_routine6;$i++)
{
($pattern1,$pattern2,$metric6[$i]) = split(/\s+/, $routine6[$i]);
}
$size_of_metric=@metric6;

$RtnCplCtlAvg=$metric6[$m];

@routine7= grep{$_=~/RtnCplCtlMax/} @rtnmetrics;
$size_of_routine7=@routine7;
for($i=0;$i<=$size_of_routine7;$i++)
{
($pattern1,$pattern2,$metric7[$i]) = split(/\s+/, $routine7[$i]);
}
$size_of_metric=@metric2;
$RtnCplCtlMax=@metric7[$m];
@routine40= grep{$_=~/RtnCplCtlSum/} @rtnmetrics;
$size_of_routine40=@routine40;
for($i=0;$i<=$size_of_routine7;$i++)
{
($pattern1,$pattern2,$metric40[$i]) = split(/\s+/, $routine40[$i]);
}
$size_of_metric=@metric40;
$RtnCplCtlSum=$metric40[$m];
@routine8= grep{$_=~/RtnCplCycNbr/} @rtnmetrics;
$size_of_routine8=@routine8;
for($i=0;$i<=$size_of_routine8;$i++)
{
($pattern1,$pattern2,$metric8[$i]) = split(/\s+/, $routine8[$i]);
}
$size_of_metric=@metric2;

$RtnCplCycNbr=$metric8[$m];
@routine9= grep{$_=~/RtnCplExeAvg/} @rtnmetrics;
$size_of_routine9=@routine9;
for($i=0;$i<=$size_of_routine9;$i++)
{
($pattern1,$pattern2,$metric9[$i]) = split(/\s+/, $routine9[$i]);
}
$size_of_metric=@metric2;

$RtnCplExeAvg=$metric9[$m];
```

```
 @routine10= grep{$_=~/RtnCplExeMax/} @rtnmetrics;
 $size_of_routine10=@routine10;
 for($i=0;$i<=$size_of_routine10;$i++)
 {
 ($pattern1,$pattern2,$metric10[$i]) = split(/\s+/, $routine10[$i]);
 }
 $size_of_metric=@metric2;

 $RtnCplExeMax=$metric10[$m];
 @routine11= grep{$_=~/RtnCplExeSum/} @rtnmetrics;
 $size_of_routine11=@routine11;
 for($i=0;$i<=$size_of_routine11;$i++)
 {
 ($pattern1,$pattern2,$metric11[$i]) = split(/\s+/, $routine11[$i]);
 }
 $size_of_metric=@metric2;

 $RtnCplExeSum=$metric11[$m];
 @routine12= grep{$_=~/RtnLblNbr/} @rtnmetrics;
 $size_of_routine12=@routine12;
 for($i=0;$i<=$size_of_routine12;$i++)
 {
 ($pattern1,$pattern2,$metric12[$i]) = split(/\s+/, $routine12[$i]);
 }
 $size_of_metric=@metric2;

 $RtnLblNbr=$metric12[$m];
 @routine13= grep{$_=~/RtnLnsNbr/} @rtnmetrics;
 $size_of_routine13=@routine13;
for($i=0;$i<=$size_of_routine13;$i++)
 {
 ($pattern1,$pattern2,$metric13[$i]) = split(/\s+/, $routine13[$i]);
 }
 $size_of_metric=@metric2;

 $RtnLnsNbr=$metric13[$m];

 $size_of_routine14=@routine14;
 @routine14= grep{$_=~/RtnLnsSkpSum/} @rtnmetrics;
 for($i=0;$i<=$size_of_routine14;$i++)
 {
 ($pattern1,$pattern2,$metric14[$i]) = split(/\s+/, $routine14[$i]);
 }
 $size_of_metric=@metric2;

 $RtnLnsSkpSum=$metric14[$m];
 @routine15= grep{$_=~/RtnScpNbr/} @rtnmetrics;
 $size_of_routine15=@routine15;
 for($i=0;$i<=$size_of_routine15;$i++)
 {
```

```
($pattern1,$pattern2,$metric15[$i]) = split(/\s+/, $routine15[$i]);
}
$size_of_metric=@metric2;

$RtnScpNbr=$metric15[$m];
@routine16= grep{$_=~/RtnScpNstLvlAvg/} @rtnmetrics;
$size_of_routine16=@routine16;
for($i=0;$i<=$size_of_routine16;$i++)
{
($pattern1,$pattern2,$metric16[$i]) = split(/\s+/, $routine16[$i]);
}
$size_of_metric=@metric2;

$RtnScpNstLvlAvg=$metric16[$m];
@routine17= grep{$_=~/RtnScpNstLvlMax/} @rtnmetrics;
$size_of_routine17=@routine17;
for($i=0;$i<=$size_of_routine17;$i++)
{
($pattern1,$pattern2,$metric17[$i]) = split(/\s+/, $routine17[$i]);
}
$size_of_metric=@metric2;

$RtnScpNstLvlMax=$metric17[$m];
@routine18= grep{$_=~/RtnScpNstLvlSum/} @rtnmetrics;
$size_of_routine18=@routine18;
for($i=0;$i<=$size_of_routine18;$i++)
{
($pattern1,$pattern2,$metric18[$i]) = split(/\s+/, $routine18[$i]);
}
$size_of_metric=@metric2;

$RtnScpNstLvlSum=$metric18[$m];

@routine19= grep{$_=~/RtnStmCtlBrkNbr/} @rtnmetrics;
$size_of_routine19=@routine19;
for($i=0;$i<=$size_of_routine19;$i++)
{
($pattern1,$pattern2,$metric19[$i]) = split(/\s+/, $routine19[$i]);
}
$size_of_metric=@metric2;

$RtnStmCtlBrkNbr=$metric19[$m];
@routine20= grep{$_=~/RtnStmCtlCaseNbr/} @rtnmetrics;
$size_of_routine20=@routine20;
for($i=0;$i<=$size_of_routine20;$i++)
{
($pattern1,$pattern2,$metric20[$i]) = split(/\s+/, $routine20[$i]);
}
$size_of_metric=@metric2;
```

```
$RtnStmCtlCaseNbr=$metric20[$m];
@routine21= grep{$_=~/RtnStmCtlCtnNbr/} @rtnmetrics;
$size_of_routine21=@routine21;
for($i=0;$i<=$size_of_routine21;$i++)
{
($pattern1,$pattern2,$metric21[$i]) = split(/\s+/, $routine21[$i]);
}
$size_of_metric=@metric2;

$RtnStmCtlCtnNbr=$metric21[$m];
@routine22= grep{$_=~/RtnStmCtlDfltNbr/} @rtnmetrics;
$size_of_routine22=@routine22;
for($i=0;$i<=$size_of_routine22;$i++)
{
($pattern1,$pattern2,$metric22[$i]) = split(/\s+/, $routine22[$i]);
}
$size_of_metric=@metric2;

$RtnStmCtlDfltNbr=$metric22[$m];
@routine23= grep{$_=~/RtnStmCtlGotoNbr/} @rtnmetrics;
$size_of_routine23=@routine23;
for($i=0;$i<=$size_of_routine23;$i++)
{
($pattern1,$pattern2,$metric23[$i]) = split(/\s+/, $routine23[$i]);
}
$size_of_metric=@metric2;

$RtnStmCtlGotoNbr=$metric23[$m];
@routine24= grep{$_=~/RtnStmCtlIfNbr/} @rtnmetrics;
$size_of_routine24=@routine24;
for($i=0;$i<=$size_of_routine24;$i++)
{
($pattern1,$pattern2,$metric24[$i]) = split(/\s+/, $routine24[$i]);
}
$size_of_metric=@metric2;

$RtnStmCtlIfNbr=$metric24[$m];
@routine25= grep{$_=~/RtnStmCtlLopNbr/} @rtnmetrics;
$size_of_routine25=@routine25;
for($i=0;$i<=$size_of_routine25;$i++)
{
($pattern1,$pattern2,$metric25[$i]) = split(/\s+/, $routine25[$i]);
}
$size_of_metric=@metric2;

$RtnStmCtlLopNbr=$metric25[$m];
@routine26= grep{$_=~/RtnStmCtlNbr/} @rtnmetrics;
$size_of_routine26=@routine26;
for($i=0;$i<=$size_of_routine26;$i++)
{
```

```
($pattern1,$pattern2,$metric26[$i]) = split(/\s+/, $routine26[$i]);
}
$size_of_metric=@metric2;

$RtnStmCtlNbr=$metric26[$m];
@routine27= grep{$_=~/RtnStmCtlRetNbr/} @rtnmetrics;
$size_of_routine27=@routine27;
for($i=0;$i<=$size_of_routine27;$i++)
{
($pattern1,$pattern2,$metric27[$i]) = split(/\s+/, $routine27[$i]);
}
$size_of_metric=@metric2;

$RtnStmCtlRetNbr=$metric27[$m];
@routine28= grep{$_=~/RtnStmCtlSwiNbr/} @rtnmetrics;
$size_of_routine28=@routine28;
for($i=0;$i<=$size_of_routine28;$i++)
{
($pattern1,$pattern2,$metric28[$i]) = split(/\s+/, $routine28[$i]);
}
$size_of_metric=@metric2;

$RtnStmCtlSwiNbr=$metric28[$m];
@routine29= grep{$_=~/RtnStmCtlThwNbr/} @rtnmetrics;
$size_of_routine29=@routine29;
for($i=0;$i<=$size_of_routine29;$i++)
{
($pattern1,$pattern2,$metric29[$i]) = split(/\s+/, $routine29[$i]);
}
$size_of_metric=@metric2;

$RtnStmCtlThwNbr=$metric29[$m];
@routine41= grep{$_=~/RtnStmDecNbr/} @rtnmetrics;
$size_of_routine41=@routine41;
for($i=0;$i<=$size_of_routine41;$i++)
{
($pattern1,$pattern2,$metric41[$i]) = split(/\s+/, $routine41[$i]);
}
$size_of_metric=@metric2;

$RtnStmDecNbr=$metric41[$m];

@routine30= grep{$_=~/RtnStmDecObjNbr/} @rtnmetrics;
$size_of_routine30=@routine30;
for($i=0;$i<=$size_of_routine30;$i++)
{
($pattern1,$pattern2,$metric30[$i]) = split(/\s+/, $routine30[$i]);
}
$size_of_metric=@metric2;
```

```
$RtnStmDecObjNbr=$metric30[$m];
@routine31= grep{$_=~/RtnStmDecPrmNbr/} @rtnmetrics;
$size_of_routine31=@routine31;
for($i=0;$i<=$size_of_routine31;$i++)
{
($pattern1,$pattern2,$metric31[$i]) = split(/\s+/, $routine31[$i]);
}
$size_of_metric=@metric2;

$RtnStmDecPrmNbr=$metric31[$m];
@routine32= grep{$_=~/RtnStmDecRtnNbr/} @rtnmetrics;
$size_of_routine32=@routine32;
for($i=0;$i<=$size_of_routine32;$i++)
{
($pattern1,$pattern2,$metric32[$i]) = split(/\s+/, $routine32[$i]);
}
$size_of_metric=@metric2;

$RtnStmDecRtnNbr=$metric32[$m];
@routine33= grep{$_=~/RtnStmDecTypeNbr/} @rtnmetrics;
$size_of_routine33=@routine33;
for($i=0;$i<=$size_of_routine33;$i++)
{
($pattern1,$pattern2,$metric33[$i]) = split(/\s+/, $routine33[$i]);
}
$size_of_metric=@metric2;

$RtnStmDecTypeNbr=$metric33[$m];
@routine34= grep{$_=~/RtnStmExeNbr/} @rtnmetrics;
$size_of_routine34=@routine34;
for($i=0;$i<=$size_of_routine34;$i++)
{
($pattern1,$pattern2,$metric34[$i]) = split(/\s+/, $routine34[$i]);
}
$size_of_metric=@metric2;

$RtnStmExeNbr=$metric34[$m];
@routine35= grep{$_=~/RtnStmNbr/} @rtnmetrics;
$size_of_routine35=@routine35;
for($i=0;$i<=$size_of_routine35;$i++)
{
($pattern1,$pattern2,$metric35[$i]) = split(/\s+/, $routine35[$i]);
}
$size_of_metric=@metric35;

$RtnStmNbr=$metric35[$m];
@routine36= grep{$_=~/RtnStmNstLvlAvg/} @rtnmetrics;
$size_of_routine36=@routine36;
for($i=0;$i<=$size_of_routine36;$i++)
{
```

```
  ($pattern1,$pattern2,$metric36[$i]) = split(/\s+/, $routine36[$i]);
  }
  $size_of_metric=@metric2;

  $RtnStmNstLvlAvg=$metric36[$m];
  @routine37= grep{$_=~/RtnStmNstLvlSum/} @rtnmetrics;
  $size_of_routine37=@routine37;
  for($i=0;$i<=$size_of_routine37;$i++)
  {
  ($pattern1,$pattern2,$metric37[$i]) = split(/\s+/, $routine37[$i]);
  }
  $size_of_metric=@metric2;

  $RtnStmNstLvlSum=$metric37[$m];
  @routine38= grep{$_=~/RtnStmXpdNbr/} @rtnmetrics;
  $size_of_routine38=@routine38;
  for($i=0;$i<=$size_of_routine38;$i++)
  {
  ($pattern1,$pattern2,$metric38[$i]) = split(/\s+/, $routine38[$i]);
  }
  $size_of_metric=@metric2;

  $RtnStmXpdNbr=$metric38[$m];
  @routine39= grep{$_=~/RtnStxErrNbr/} @rtnmetrics;
  $size_of_routine39=@routine39;
  for($i=0;$i<=$size_of_routine39;$i++)
  {
  ($pattern1,$pattern2,$metric39[$i]) = split(/\s+/, $routine39[$i]);
  }
  $size_of_metric=@metric2;

  $RtnStxErrNbr=$metric39[$m];

 write RTNMET;
  }

 #******* THIS IS THE  END OF THE MAIN LOOP ********
 close RTNMET;
 close RTN;
 close MET;


#!/usr/bin/perl

format OUT_TOP =
 "FILE NAME", "FUNCTION NAME","RtnArgXplSum","RtnCalXplNbr","RtnCastXplNbr",
 "RtnComNbr","RtnComVol","RtnCplCtlAvg","RtnCplCtlMax","RtnCplCtlSum",
 "RtnCplCycNbr","RtnCplExeAvg","RtnCplExeMax","RtnCplExeSum","RtnLblNbr",
 "RtnLnsNbr","RtnLnsSkpSum","RtnScpNbr","RtnScpNstLvlAvg","RtnScpNstLvlMax",
 "RtnScpNstLvlSum","RtnStmCtlBrkNbr","RtnStmCtlCaseNbr","RtnStmCtlCtnNbr",
 "RtnStmCtlDfltNbr","RtnStmCtlGotoNbr","RtnStmCtlIfNbr","RtnStmCtlLopNbr",
```

```
 "RtnStmCtlNbr","RtnStmCtlRetNbr","RtnStmCtlSwiNbr","RtnStmCtlThwNbr",
 "RtnStmDecNbr","RtnStmDecObjNbr","RtnStmDecPrmNbr","RtnStmDecRtnNbr",
 "RtnStmDecTypeNbr","RtnStmExeNbr","RtnStmNbr","RtnStmNstLvlAvg",
 "RtnStmNstLvlSum","RtnStxErrNbr","RtnStmXpdNbr"
.
 format OUT =
 @<<<<<<<<,@<<<<<<<<<<,@###.######,@###.####,@###.####,@###.######,
 $file $SCOPED_NAME $RtnArgXplSum $RtnCalXplNbr $RtnCastX-
plNbr $RtnComNbr
 $RtnComVol $RtnCplCtlAvg  $RtnCplCtlMax $RtnCplCtlSum $RtnCplCycNbr
 $RtnCplExeAvg $RtnCplExeMax $RtnCplExeSum $RtnLblNbr $RtnLnsNbr
 $RtnLnsSkpSum $RtnScpNbr  $RtnScpNstLvlAvg $RtnScpNstLvlMax
 $RtnScpNstLvlSum $RtnStmCtlBrkNbr $RtnStmCtlCaseNbr $RtnStmCtlCtnNbr
 $RtnStmCtlDfltNbr $RtnStmCtlGotoNbr  $RtnStmCtlIfNbr$RtnStmCtlLopNbr
 $RtnStmCtlNbr $RtnStmCtlRetNbr $RtnStmCtlSwiNbr $RtnStmCtlThwNbr
 $RtnStmDecNbr $RtnStmDecObjNbr $RtnStmDecPrmNbr $RtnStmDecRtnNbr
 $RtnStmDecTypeNbr $RtnStmExeNbr $RtnStmNbr $RtnStmNstLvlAvg
 $RtnStmNstLvlSum $RtnStmXpdNbr $RtnStxErrNbr
.

open(INPT,"$ARGV[0]") or die "can’t open files";
open(OUT,">>$ARGV[1]") or die "can’t open files:";

@input=<INPT>;

$size=@input;
print $size;
print OUT "$ARGV[0]\n";
for($i=0;$i<$size;$i++)
{
$filename[$i]=substr($input[$i],0,15);
#print "$i $filename[$i] \n";
}

$i=1; $n=0;$totRtnArgXplSum=0;$totRtnCalXplNbr=0;
$maxRtnCplCtl=0; $maxRtnCplExe=0; $maxRtnScpNstLvl=0;
while($i<$size)
{
$r=($filename[$i] cmp $filename[$i+1]);
print $r;
if ($r==0)
{

($file[$n],$functionname,$RtnArgXplSum[$n],$RtnCalXplNbr[$n],
$RtnCastXplNbr[$n], $RtnComNbr[$n], $RtnComVol[$n], $RtnCplCtlAvg[$n],
$RtnCplCtlMax[$n] ,$RtnCplCtlSum[$n], $RtnCplCycNbr[$n],
$RtnCplExeAvg[$n], $RtnCplExeMax[$n], $RtnCplExeSum[$n],$RtnLblNbr[$n],
$RtnLnsNbr[$n], $RtnLnsSkpSum[$n], $RtnScpNbr[$n],  $RtnScpN-
stLvlAvg[$n],
$RtnScpNstLvlMax[$n], $RtnScpNstLvlSum[$n],$RtnStmCtlBrkNbr[$n],
```

```
$RtnStmCtlCaseNbr[$n], $RtnStmCtlCtnNbr[$n], $RtnStmCtlDfltNbr[$n],
$RtnStmCtlGotoNbr[$n],  $RtnStmCtlIfNbr[$n],$RtnStmCtlLopNbr[$n],
$RtnStmCtlNbr[$n], $RtnStmCtlRetNbr[$n], $RtnStmCtlSwiNbr[$n],
$RtnStmCtlThwNbr[$n], $RtnStmDecNbr[$n], $RtnStmDecObjNbr[$n],
$RtnStmDecPrmNbr[$n], $RtnStmDecRtnNbr[$n], $RtnStmDecTypeNbr[$n],
$RtnStmExeNbr[$n], $RtnStmNbr[$n], $RtnStmNstLvlAvg[$n], $RtnStmNstLvl-
Sum[$n],
$RtnStmXpdNbr[$n], $RtnStxErrNbr[$n])=split(/\s*,\s*/,$input[$i]);

($file[$n+1],$functionname1,$RtnArgXplSum[$n+1],$RtnCalXplNbr[$n+1],
$RtnCastXplNbr[$n+1], $RtnComNbr[$n+1], $RtnComVol[$n+1], $RtnCplCt-
lAvg[$n+1],
$RtnCplCtlMax[$n+1],$RtnCplCtlSum[$n+1], $RtnCplCycNbr[$n+1], $RtnC-
plExeAvg[$n+1],
$RtnCplExeMax[$n+1], $RtnCplExeSum[$n+1],$RtnLblNbr[$n+1], $RtnLnsNbr[$n+1],
$RtnLnsSkpSum[$n+1], $RtnScpNbr[$n+1],  $RtnScpNstLvlAvg[$n+1],
$RtnScpNstLvlMax[$n+1], $RtnScpNstLvlSum[$n+1],$RtnStmCtlBrkNbr[$n+1],
$RtnStmCtlCaseNbr[$n+1], $RtnStmCtlCtnNbr[$n+1], $RtnStmCtlD-
fltNbr[$n+1],
$RtnStmCtlGotoNbr[$n+1],  $RtnStmCtlIfNbr[$n+1],$RtnStmCtlLopNbr[$n+1],
$RtnStmCtlNbr[$n+1], $RtnStmCtlRetNbr[$n+1], $RtnStmCtlSwiNbr[$n+1],
$RtnStmCtlThwNbr[$n+1], $RtnStmDecNbr[$n+1], $RtnStmDecObjNbr[$n+1],
$RtnStmDecPrmNbr[$n+1], $RtnStmDecRtnNbr[$n+1], $RtnStmDec-
TypeNbr[$n+1],
$RtnStmExeNbr[$n+1], $RtnStmNbr[$n+1], $RtnStmNstLvlAvg[$n+1],
$RtnStmNstLvlSum[$n+1], $RtnStmXpdNbr[$n+1],
$RtnStxErrNbr[$n+1])=split(/\s*,\s*/,$input[$i+1]);
#Find the value of metrics defined to find the maxi-
mum value of a given product attribute.
if($maxRtnCplCtl<$RtnCplCtlMax[$n])
{
$maxRtnCplCtl=$RtnCplCtlMax[$n];
}
else
{
$maxRtnCplCtl=$maxRtnCplCtl;
}
if($maxRtnCplExe<$RtnCplExeMax[$n])
{
$maxRtnCplExe=$RtnCplExeMax[$n];
}
else
{
 $maxRtnCplExe=$maxRtnCplExe;
}
if($maxRtnScpNstLvl<$RtnScpNstLvlMax[$n])
{
$maxRtnScpNstLvl=$RtnScpNstLvlMax[$n];
}
else
```

```
{
$maxRtnScpNstLvl=$maxRtnScpNstLvl;
}
if($n==0)
{
#Find the sum of the metrics for the number or the sum of a given prod-
uct attribute.
$totRtnArgXplSum=$RtnArgXplSum[$n]+$RtnArgXplSum[$n+1];
$totRtnCalXplNbr=$RtnCalXplNbr[$n]+$RtnCalXplNbr[$n+1];
$totRtnCastXplNbr=$RtnCastXplNbr[$n]+$RtnCastXplNbr[$n+1];
$totRtnComNbr=$RtnComNbr[$n]+$RtnComNbr[$n+1];
$totRtnComVol=$RtnComVol[$n]+$RtnComVol[$n+1];
$totRtnCplCtlSum=$RtnCplCtlSum[$n]+$RtnCplCtlSum[$n+1];
$totRtnCplCycNbr=$RtnCplCycNbr[$n]+$RtnCplCycNbr[$n+1];
$totRtnCplExeSum=$RtnCplExeSum[$n]+$RtnCplExeSum[$n+1];
$totRtnLblNbr=$RtnLblNbr[$n]+$RtnLblNbr[$n+1];
$totRtnLnsNbr=$RtnLnsNbr[$n]+$RtnLnsNbr[$n+1];
$totRtnLnsSkpSum=$RtnLnsSkpSum[$n]+$RtnLnsSkpSum[$n+1];
$totRtnScpNbr=$RtnScpNbr[$n]+$RtnScpNbr[$n+1];
$totRtnScpNstLvlSum=$RtnScpNstLvlSum[$n]+$RtnScpNstLvlSum[$n+1];
$totRtnStmCtlBrkNbr=$RtnStmCtlBrkNbr[$n]+$RtnStmCtlBrkNbr[$n+1];
$totRtnStmCtlCaseNbr=$RtnStmCtlCaseNbr[$n]+$RtnStmCtlCaseNbr[$n+1];
$totRtnStmCtlCtnNbr=$RtnStmCtlCtnNbr[$n]+$RtnStmCtlCtnNbr[$n+1];
$totRtnStmCtlDfltNbr=$RtnStmCtlDfltNbr[$n]+$RtnStmCtlDfltNbr[$n+1];
$totRtnStmCtlGotoNbr=$RtnStmCtlGotoNbr[$n]+$RtnStmCtlGotoNbr[$n+1];
$totRtnStmCtlIfNbr=$RtnStmCtlIfNbr[$n]+$RtnStmCtlIfNbr[$n+1];
$totRtnStmCtlLopNbr=$RtnStmCtlLopNbr[$n]+$RtnStmCtlLopNbr[$n+1];
$totRtnStmCtlNbr=$RtnStmCtlNbr[$n]+$RtnStmCtlNbr[$n+1];
$totRtnStmCtlRetNbr=$RtnStmCtlRetNbr[$n]+$RtnStmCtlRetNbr[$n+1];
$totRtnStmCtlSwiNbr=$RtnStmCtlSwiNbr[$n]+$RtnStmCtlSwiNbr[$n+1];
$totRtnStmCtlThwNbr=$RtnStmCtlThwNbr[$n]+$RtnStmCtlThwNbr[$n+1];
$totRtnStmDecNbr=$RtnStmDecNbr[$n]+$RtnStmDecNbr[$n+1];
$totRtnStmDecObjNbr=$RtnStmDecObjNbr[$n]+$RtnStmDecObjNbr[$n+1];
$totRtnStmDecPrmNbr=$RtnStmDecPrmNbr[$n]+$RtnStmDecPrmNbr[$n+1];
$totRtnStmDecRtnNbr=$RtnStmDecRtnNbr[$n]+$RtnStmDecRtnNbr[$n+1];
$totRtnStmDecTypeNbr=$RtnStmDecTypeNbr[$n]+$RtnStmDecTypeNbr[$n+1];
$totRtnStmExeNbr=$RtnStmExeNbr[$n]+$RtnStmExeNbr[$n+1];
$totRtnStmNbr=$RtnStmNbr[$n]+$RtnStmNbr[$n+1];
#$totRtnStmNstLvlAvg=$RtnStmNstLvlAvg[$n]+$RtnStmNstLvlAvg[$n+1];
$totRtnStmNstLvlSum=$RtnStmNstLvlSum[$n]+$RtnStmNstLvlSum[$n+1];
$totRtnStmXpdNbr=$RtnStmXpdNbr[$n]+$RtnStmXpdNbr[$n+1];
$totRtnStxErrNbr=$RtnStxErrNbr[$n]+$RtnStxErrNbr[$n+1];
print "$totRtnArgXplSum \n";
print "$totRtnCalXplNbr \n";
}
else
{
$totRtnArgXplSum+=$RtnArgXplSum[$n+1];
$totRtnCalXplNbr+=$RtnCalXplNbr[$n+1];
$totRtnCastXplNbr+=$RtnCastXplNbr[$n+1];
```

```
$totRtnComNbr+=$RtnComNbr[$n+1];
$totRtnComVol+=$RtnComVol[$n+1];
$totRtnCplCtlSum+=$RtnCplCtlSum[$n+1];
$totRtnCplCycNbr+=$RtnCplCycNbr[$n+1];
$totRtnCplExeSum+=$RtnCplExeSum[$n+1];
$totRtnLblNbr+=$RtnLblNbr[$n+1];
$totRtnLnsNbr+=$RtnLnsNbr[$n+1];
$totRtnLnsSkpSum+=$RtnLnsSkpSum[$n+1];
$totRtnScpNbr+=$RtnScpNbr[$n+1];
$totRtnScpNstLvlSum+=$RtnScpNstLvlSum[$n+1];
$totRtnStmCtlBrkNbr+=$RtnStmCtlBrkNbr[$n+1];
$totRtnStmCtlCaseNbr+=$RtnStmCtlCaseNbr[$n+1];
$totRtnStmCtlCtnNbr+=$RtnStmCtlCtnNbr[$n+1];
$totRtnStmCtlDfltNbr+=$RtnStmCtlDfltNbr[$n+1];
$totRtnStmCtlGotoNbr+=$RtnStmCtlGotoNbr[$n+1];
$totRtnStmCtlIfNbr+=$RtnStmCtlIfNbr[$n+1];
$totRtnStmCtlLopNbr+=$RtnStmCtlLopNbr[$n+1];
$totRtnStmCtlNbr+=$RtnStmCtlNbr[$n+1];
$totRtnStmCtlRetNbr+=$RtnStmCtlRetNbr[$n+1];
$totRtnStmCtlSwiNbr+=$RtnStmCtlSwiNbr[$n+1];
$totRtnStmCtlThwNbr+=$RtnStmCtlThwNbr[$n+1];
$totRtnStmDecNbr+=$RtnStmDecNbr[$n+1];
$totRtnStmDecObjNbr+=$RtnStmDecObjNbr[$n+1];
$totRtnStmDecPrmNbr+=$RtnStmDecPrmNbr[$n+1];
$totRtnStmDecRtnNbr+=$RtnStmDecRtnNbr[$n+1];
$totRtnStmDecTypeNbr+=$RtnStmDecTypeNbr[$n+1];
$totRtnStmExeNbr+=$RtnStmExeNbr[$n+1];
$totRtnStmNbr+=$RtnStmNbr[$n+1];
$totRtnStmNstLvlSum+=$RtnStmNstLvlSum[$n+1];
$totRtnStmXpdNbr+=$RtnStmXpdNbr[$n+1];
$totRtnStxErrNbr+=$RtnStxErrNbr[$n+1];
}
$n=$n+1;
$RtnArgXplSum=$totRtnArgXplSum;
$RtnCalXplNbr=$totRtnCalXplNbr;
$RtnCastXplNbr=$totRtnCastXplNbr;
$RtnComNbr=$totRtnComNbr;
$RtnComVol=$totRtnComVol;

#Find the average of the metrics which define the average of a prod-
uct attribute.

$RtnCplCtlAvg=($totRtnCplCtlSum/
($totRtnStmCtlIfNbr+$totRtnStmCtlSwiNbr+$totRtnStmCtlLopNbr));
print "the value of totRtnCplCtlSum is $totRtnCplCtl-
Sum,$totRtnStmCtlIfNbr";
print " the value of RtnCplCtlAvg is $RtnCplCtlAvg\n";
$RtnCplCtlMax=$maxRtnCplCtl;
print " the value of RtnCplCtlMax is $maxRtnCplCtl\n";
$RtnCplCtlSum=$totRtnCplCtlSum;
```

```
$RtnCplCycNbr=$totRtnCplCycNbr;
$RtnCplExeAvg=($totRtnCplExeSum/$totRtnStmExeNbr);
print " the value of RtnCplExeAvg is $RtnCplExeAvg\n";
$RtnCplExeMax=$maxRtnCplExe;
print "the max value of RtnCplExeMax is $maxRtnCplExe\n";
$RtnCplExeSum=$totRtnCplExeSum;
$RtnLblNbr=$totRtnLblNbr;
$RtnLnsNbr=$totRtnLnsNbr;
$RtnLnsSkpSum=$totRtnLnsSkpSum;
$RtnScpNbr=$totRtnScpNbr;
$RtnScpNstLvlAvg=($totRtnScpNstLvlSum/$totRtnScpNbr);
print " the value of RtnScpNstLvlAvg is $RtnScpNstLvlAvg\n";
$RtnScpNstLvlMax=$maxRtnScpNstLvl;
print "the max value of RtnScpNstLvlMax is $maxRtnScpNstLvl\n";
$RtnScpNstLvlSum=$totRtnScpNstLvlSum;
$RtnStmCtlBrkNbr=$totRtnStmCtlBrkNbr;
$RtnStmCtlCaseNbr=$totRtnStmCtlCaseNbr;
$RtnStmCtlCtnNbr=$totRtnStmCtlCtnNbr;
$RtnStmCtlDfltNbr=$totRtnStmCtlDfltNbr;
$RtnStmCtlGotoNbr= $totRtnStmCtlGotoNbr;
$RtnStmCtlIfNbr=$totRtnStmCtlIfNbr;
$RtnStmCtlLopNbr=$totRtnStmCtlLopNbr;
$RtnStmCtlNbr=$totRtnStmCtlNbr;
$RtnStmCtlRetNbr=$totRtnStmCtlRetNbr;
$RtnStmCtlSwiNbr= $totRtnStmCtlSwiNbr;
$RtnStmCtlThwNbr=$totRtnStmCtlThwNbr;
$RtnStmDecNbr=$totRtnStmDecNbr;
$RtnStmDecObjNbr=$totRtnStmDecObjNbr;
$RtnStmDecPrmNbr=$totRtnStmDecPrmNbr;
$RtnStmDecRtnNbr=$totRtnStmDecRtnNbr;
$RtnStmDecTypeNbr=$totRtnStmDecTypeNbr;
$RtnStmExeNbr=$totRtnStmExeNbr;
$RtnStmNbr=$totRtnStmNbr;
$RtnStmNstLvlAvg=($totRtnStmNstLvlSum/$totRtnStmNbr);
print " the value of RtnStmNstLvlAvg is $RtnStmNstLvlAvg";
$RtnStmNstLvlSum=$totRtnStmNstLvlSum;
$RtnStmXpdNbr= $totRtnStmXpdNbr;
$RtnStxErrNbr=$totRtnStxErrNbr;
$file=$file[$n];
$i=$i+1;
write OUT;
}
else
{
($file,$RtnArgXplSum,$RtnCalXplNbr,$RtnCastXplNbr, $RtnComNbr,
$RtnComVol, $RtnCplCtlAvg , $RtnCplCtlMax ,$RtnCplCtlSum,
$RtnCplCycNbr, $RtnCplExeAvg, $RtnCplExeMax, $RtnCplExeSum,
$RtnLblNbr, $RtnLnsNbr, $RtnLnsSkpSum, $RtnScpNbr,  $RtnScpNstLvlAvg,
$RtnScpNstLvlMax, $RtnScpNstLvlSum,$RtnStmCtlBrkNbr, $RtnStmCtlCaseNbr,
$RtnStmCtlCtnNbr, $RtnStmCtlDfltNbr, $RtnStmCtlGotoNbr,
```

```
$RtnStmCtlIfNbr,$RtnStmCtlLopNbr, $RtnStmCtlNbr, $RtnStmCtlRetNbr,
$RtnStmCtlSwiNbr, $RtnStmCtlThwNbr, $RtnStmDecNbr, $RtnStmDecObjNbr,
$RtnStmDecPrmNbr, $RtnStmDecRtnNbr, $RtnStmDecTypeNbr, $RtnStmExeNbr,
$RtnStmNbr, $RtnStmNstLvlAvg, $RtnStmNstLvlSum,
$RtnStmXpdNbr, $RtnStxErrNbr)=split(/\s*,\s*/,$input[$i]);
write OUT;
$totRtnArgXplSum=0;
$totRtnCalXplNbr=0;
$totRtnArgXplSum=0;
$totRtnCalXplNbr=0;
$totRtnCastXplNbr=0;
$totRtnComNbr=0;
$totRtnComVol=0;
$totRtnCplCtlAvg=0;
$totRtnCplCtlMax=0;
$totRtnCplCtlSum=0;
$totRtnCplCycNbr=0;
$totRtnCplExeAvg=0;
$totRtnCplExeMax=0;
$totRtnCplExeSum=0;
$totRtnCplCycNbr=0;
$totRtnCplExeAvg=0;
$totRtnCplExeMax=0;
$totRtnCplExeSum=0;
$totRtnLblNbr=0;
$totRtnLnsNbr=0;
$totRtnLnsSkpSum=0;
$totRtnScpNbr=0;
$totRtnScpNstLvlAvg=0;
$totRtnScpNstLvlMax=0;
$totRtnScpNstLvlSum=0;
$totRtnStmCtlBrkNbr=0;
$totRtnStmCtlCaseNbr=0;
$totRtnStmCtlCtnNbr=0;
$totRtnStmCtlDfltNbr=0;
$totRtnStmCtlGotoNbr=0;
$totRtnStmCtlIfNbr=0;
$totRtnStmCtlLopNbr=0;
$totRtnStmCtlNbr=0;
$totRtnStmCtlRetNbr=0;
$totRtnStmCtlSwiNbr=0;
$totRtnStmCtlThwNbr=0;
$totRtnStmDecNbr=0;
$totRtnStmDecObjNbr=0;
$totRtnStmDecPrmNbr=0;
$totRtnStmDecRtnNbr=0;
$totRtnStmDecTypeNbr=0;
$totRtnStmExeNbr=0;
$totRtnStmNbr=0;
$totRtnStmNstLvlAvg=0;
```

```
$totRtnStmNstLvlSum=0;
$totRtnStmXpdNbr=0;
$totRtnStxErrNbr=0;
$n=0;
$i=$i+1;
}
}

close(INPT);
close(OUT);
```

```
#!/usr/bin/perl

format CHNG_TOP=
"FILE NAME", "CHANGED LINES", "ADDED LINES", "DELETED LINES"
.
format CHNG=
@<<<<<<<<<<<< @###.####,    @###.###,   @###.##
$ARGV[0]      $changedlines,$addedlines,$deletedlines
.

open(INPT,"$ARGV[0]") or die "can't open files"; #****** En-
ter the file containing the Diff output*****
open(OUT,">$ARGV[1]") or die "can't open files";
open(CHNG,">>changesrcode") or die "can't open changesrcode";
@input =<INPT>;
#print @input;
$size=@input;
#print $size;

for($i=0;$i<=$size;$i++)
{

  if($input[$i]=~/^\- /)
  {
      print OUT $input[$i];
     }
}

for($i=0;$i<=$size;$i++)
{
    if($input[$i]=~/^\-{3}/)
     {
       print "$input[$i] \n";
       #print " this is inside the if loop\n";
       #print "$i \n";
       $pos=$i+1;
       print "outof while \n";
      while($input[$pos]!~/\*{12,15}/)
      {
        print "this is inside the while loop \n";
```

```
        print "$input[$pos] \n";
        print OUT $input[$pos];
       $pos=$pos+1;
      if ($pos==$size)
       {
      last;
      }
       print "$pos \n";
       }

 print "this is outside the while loop \n";
       }
}
close(OUT);
close(INPT);
#open(CHNG,">>changesrcode") or die "can't open changesrcode";
open(OUT,"$ARGV[1]") or die "can't open file $ARGV[1]";
 @totalines=<OUT>;
 print @totalines;
 $arraysize=@totalines;
 print "the size of the array is $arraysize \n";
 $changedlines=0;
 $addedlines=0;
 $deletedlines=0;

 for($i=0;$i<=$arraysize;$i++)
 {
  if($totalines[$i]=~/^\!/)
  {
  $changedlines+=1;
  }
  else{
  if($totalines[$i]=~/^\+/)
  {
  $addedlines+=1;
  }else{
  if($totalines[$i]=~/^\-/)
  {
  $deletedlines+=1;
  }
  }
  }
  }
  print "the total number of changedlines is $changedlines\n";
  print "the total number of addedlines is $addedlines\n";
  print "the total number of deletedlines is $deletedlines\n";
  write CHNG;
  #write CHNG $changedlines;
  #write CHNG $addedlines;
  #write CHNG $deletedlines;
```

```
#print "after open files\n";
close(OUT);
close(CHNG);
```