Estimating software readiness using predictive models

Mie Mie Thet Thwin

2006

Mie Mie Thet Thwin. (2006). Estimating software readiness using predictive models.
Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/4893>

<https://doi.org/10.32657/10356/4893>

Nanyang Technological University

Downloaded on 26 Apr 2023 19:34:10 SGT

Q210.41

Estimating Software Readiness Using Predictive Models

Mie Mie Thet Thwin

School of Electrical & Electronic Engineering

A thesis submitted to the Nanyang Technological University
in fulfilment of the requirement for the degree of
Doctor of Philosophy

2006

QA
-16.76
-AGS
MG21
2006

Acknowledgments

I would like to take this opportunity to thank individually, all those who have assisted me in one way or the other with my PhD research. There are many people to thank for their support and encouragement, without whom this thesis would not have been possible.

First and foremost, I am greatly indebted to my supervisor, Dr. Quah Tong Seng, for kindly leading me on and supporting my research with his technical expertise. He showed me how to conduct research and aided the development of my written and presentation skills.

I am particularly grateful to the Ministry of Science & Technology, Myanmar, and the government of Singapore for allowing me to do my PhD research at Nanyang Technological University. I am greatly indebted to Nanyang Technological University for providing me the scholarship.

I would like greatly thank Dr. Pyke Tin, University of Computer Studies, Yangon, Myanmar for his assistance and supervision the years during my Masters and Dr. Ni Lar Thein, University of Computer Studies, Yangon, Myanmar for her kind support and encouragement throughout this study.

Thanks to technicians in the InfoComm Research Lab, and friends and colleagues who have directly or indirectly been helpful throughout my study.

I am also grateful to a number of my colleagues and friends from Myanmar and Singapore who helped me in the tedious task of data collection. I gratefully acknowledged Dr. Mei-Hwa Chen, Computer Science Department, University at Albany, State University of New York, CAIB GmbH, Murrhardt, Germany and Hwafuh, Myanmar Company for providing us with the software development data used in this research.

Finally, I would like to acknowledge with deepest gratitude of my grand parents, parents and sisters throughout my education. Therefore, this doctoral thesis is gratefully dedicated to all of them.

Table of Contents

Acknowledgments.....	i
Table of Contents	ii
Summary	vi
List of Figures.....	viii
List of Tables	ix
 Chapter 1	 1
1. Introduction.....	1
1.1 Motivation.....	1
1.2 Objective.....	3
1.3 Research Questions.....	3
1.4 Related Research Area.....	4
1.4.1 Software Reliability	4
1.4.2 Software Metrics.....	5
1.4.3 Static Analysis Tools	5
1.4.4 Neural Network Techniques for Building Predictive Models	6
1.5 Major Contribution of the Thesis.....	6
1.6 Organization of the Thesis.....	7
 Chapter 2.....	 9
2. Literature Review	9
2.1 Software Reliability Models	9
2.1.1 Software Reliability Growth Models	10
2.1.2 Modelsthat are Based on the Structure of the Program	11
2.1.2.1 Traditional Software Metrics.....	11
2.1.2.2 Object-Oriented Metrics	12

2.1.2.3 Empirical Validation of Object-Oriented Metrics Using Logistic Regression.....	14
2.1.2.4 Empirical Validation of Object-Oriented Metrics Using Multiple LinearRegression	6
2.1.2.5 Empirical Validation of Object-OrientedMetrics Using Machine Learning Techniques.....	17
2.1.2.6 Summary Remarks for Object-Oriented Metrics Studies.....	17
2.1.3 Error-seeding Models.....	18
2.2 Static Analysis Tools	18
2.3 NeuralNetwork Techniques for Building Predictive Models	20
2.3.1 Comparison with Other Models.....	20
2.3.2 Neural Network Models for Classification.....	21
2.3.3 Summary Remarksfor Metric-based Neural Network Models..	23
2.4 Summary Remarks.....	23
Chapter 3.....	25
3. Software Quality Prediction Models for Object-Oriented Systems.....	25
3.1 Chapter Preface.....	25
3.2 Design of the Study.....	26
3.2.1 The choice of neuralnetwork architecture.....	26
3.2.2 Neural network modeling	28
3.2.3 Principal component analysis	30
3.3 Prediction of number of faults	30
3.3.1 Description of data.....	30
3.3.2 Selection of metrics.....	31
3.3.3 Experiments	33
3.3.4 Experiment results	35
3.3.5 Results from a 10-cross-validation study.....	36
3.4 Prediction of maintenance effort.....	38
3.4.1 Description of data.....	38
3.4.2 Selection of metrics.....	38
3.4.3 Experiments.....	39
3.4.4 Experiment results	41
3.4.5 Resultsfrom a 10-cross-validation study.....	43

3.5	Conclusion	44
Chapter 447		
4.	Prediction of Software Development Faults in PL/SQL Files	47
4.1	Chapter Preface.....	47
4.2	SQL Metrics	48
4.3	Data Collection	48
4.4	Experiment.....	49
4.5	Experimental Results	51
4.6	Results from a 10-cross-validation study	52
4.7	Conclusion	53
Chapter 5 55		
5.	Estimating Software Readiness using Predictive Models	55
5.1	Chapter Preface.....	55
5.2	Software metrics for the Data Access Tier	59
5.3	Software metricsfor the Business Tier	62
5.4	Software metrics for the Presentation Logic Tier	64
5.5	Empirical study.....	65
5.5.1	Data collection	66
5.5.2	Experiment for the Data Access Tier	68
5.5.2.1	Experiment 1	68
5.5.2.2	Experiment 2.....	70
5.5.3	Experiment for Business Tie	72
5.5.4	Experiment for Presentation Logic Tier	74
5.6	Conclusion and Discussion.....	77
Chapter 6.....79		
6.	An Implementation of Metrics Extraction and Analyzer Tool	79
6.1	Chapter Preface.....	79
6.2	Generated Software Metrics	80
6.3	Architecture Description of Metric Extraction and Analyzer Tool	80
6.3.1	Metric extraction module	81
6.3.2	Principal component analysis module	84
6.4	An Analysis of Mctrics Extracted Using the Tool Developed.....	85

6.5	Conclusion	88
Chapter 7..... 89		
7.	Conclusion and Recommendations.....	89
7.1	Conclusion	89
7.2	Findings in Relation to the Initial Research Questions.....	90
7.3	Other Findings	92
7.4	The Cross Section Study.....	93
7.5	Threads to Validity and Limitation.....	94
7.5.1	Construct validity and limitation	94
7.5.2	Internal validity and limitation.....	95
7.5.3	External validity and limitation	95
7.6	Recommendations for Further Research.....	96
Author's Publications		97
Bibliography		99
Appendix A Descriptive statistics and Regression Analysis for HMI system..		110
Appendix B Descriptive statistics and Regression Analysis for QUES system		111
Appendix C SQL metrics and principal components data from PLISQL files of warehouse management applications.....		112
Appendix D Descriptive statistics and regression Analysis for SQL metrics from PLISQL files of warehouse management applications		116
Appendix E Regression analysis for N-tier applications		117
Appendix F Correlations between faults,required time (in minutes) and lines		122
Appendix G Sstatistical measures.....		124

Summary

In times of global competition, companies require global information systems to compete in the marketplace. As internet applications are much more complex than conventional information system, new architectures have been developed. The N-tier application architecture supports a flexible and reusable application for distribution to any number of client interfaces. Although they are widely used nowadays, none of the prediction of number of software development faults (defects) studies is based on architecture layers yet. This research introduces predictive models that are based on N-tier software development architecture exploiting the new technologies. The desire to predict software readiness is essential to any software house. This task is treacherous, treading the line between releasing poor quality software early and high quality software late. Our system predicts the number of faults (defects) that occurs during system testing and the maintenance cost for correcting these faults (defects) by using software design metrics. By comparing the predicted number of faults (defects) and number of faults (defects) discovered in testing, software managers can decide whether the software are ready to be released or not. Our models can be used as one of the tools to assist software project managers, software quality assurance professions, or others in judging the readiness of software for release.

Artificial neural networks and genetic training strategy are adept at modeling nonlinear functional relationships that are difficult to model with other techniques, and thus are attractive for software quality modeling. The use of neural network model with a genetic training strategy is introduced to improve prediction results for estimating software readiness in this research. Our prediction model is divided into three parts: (1) prediction model for Presentation Logic Tier (2) prediction model for Business Tier and (3) prediction model for Data Access Tier. In N-tier application architecture, each layer serves distinct and separate tasks and different software metrics are needed for each of these tiers. Therefore the new set of software metrics is required for N-tier applications. The classification scheme was based on the nature of these faults (defects) and their relevance to the application architecture. These models are validated using real world application data which are collected from warehouse management system (WMS) and Hwafuh information system (HIS).

Although database applications are essential to every organization, studies on product measures for static database operation statements are rarely been studied and validated. Structured Query Language (SQL) is the standard language for relational database management systems. The relationship between the fault (defect) occurrence for database applications and SQL metrics is also studied in this research. We proposed new SQL metrics that have strong relationship with faults (defects) and then performed empirical validation for these metrics.

We have also done software quality estimation for object-oriented system using object-oriented design metrics and evaluation of that prediction model using real word applications; the Human Machine Interface System and Quality Evaluation System.

List of Figures

Figure 3.1. Ward Neural Network	29
Figure 3.2. GRNN Network	29
Figure 3.3. Comparison chart for number of faults prediction	45
Figure 3.4. Comparison chart for number of line changes prediction	46
Figure 5.1. A Typical N-Tier Model.....	57
Figure 6.1. Architecture description	81
Figure 6.2. Class diagram for class information generation sub system	82
Figure 6.3. Data model for class information generation sub system.....	83
Figure 6.4. Software metrics for Business Tier	84
Figure 6.5. Software metrics for Data Access Tier	84

List of Tables

Table 3.1 Comparison of prediction results using seven neural network structures	27
Table 3.2. Rotated Principal Components for HMI system.....	34
Table 3.3. Ward neural network architecture used	35
Table 3.4. GRNN neural network architecture used.....	35
Table 3.5. Experimental result for HMI system	36
Table 3.6. Results from 10-cross validation for GRNN network	37
Table 3.7. Results from 10-cross validation for Ward Network	37
Table 3.8. Rotated principal components for QUES system	40
Table 3.9. Ward neural network architecture used for QUES system	41
Table 3.10. GRNN neural network architecture used.....	41
Table 3.11. Experimental result for QUES system.....	43
Table 3.12. Results from 10-cross validation for GRNN network	43
Table 3.13. Results from 10-cross validation for Ward network.....	44
Table 4.1. Proposed SQL metrics	48
Table 4.2. Principal Components	49
Table 4.3. Ward neural network architecture used	50
Table 4.4. GRNN architecture used.....	51
Table 4.5. Experimental result	51
Table 4.6. Results from 10-cross validation for GRNN network	52
Table 4.7 Results from 10-cross validation for Ward Network	53
Table 5.1. Descriptive statistics for the Data Access Tier of the WMS system data set.....	67
Table 5.2. Descriptive statistics for Presentation Logic tier of HIS system data set	67
Table 5.3. Descriptive statistics for the Data Access Tier of HIS system data set	68
Table 5.4. Descriptive statistics for the Business tier of HIS system data set	68
Table 5.5. Experimental result for the WMA system	69
Table 5.6. Relative importance of Data Access Tier metrics for the WMA system	70
Table 5.7. Relative importance of Data Access Tier metrics	72
Table 5.8 Experiments 2 results for the Data Access Tier	72
Table 5.9. Relative importance of Business Tier metrics	74
Table 5.10. Experiments results for the Business Tier	74

Table 5.11. Relative importance of Presentation Logic Tier metrics	76
Table 5.12. Experiments results for the Presentation Logic Tier	76
Table 5.13. Comparative performance data for the models	76
Table 6.1. Data Access Tier metrics	80
Table 6.2. Business Tier metrics	80
Table 6.3. Correlation metrics for Data Access Tier metrics	85
Table 6.4. Generated principal components for Data Access Tier metrics	86
Table 6.5. Generated principal components for Business Tier metrics	87
Table 7.1. The relative importance of metrics for N-tier applications	92
Table 7.2 Comparison of prediction results for three applications	93

Chapter 1

1. Introduction

1.1 Motivation

In the competitive commercial software market, software companies feel compelled to release software the moment it is ready. Their task is treacherous, treading the line between releasing poor quality software early and high quality software late [75]. Software cannot be tested into correctness [11]. It is important to release software when it is ready, and not before or after this time. If software is released too early, then customers will be sent poor quality code; if we wait too long before releasing software we avoid the problems of low quality, but run the risk of exceeding deadlines and being penalised for the late delivery [85].

In this research, we explore the behavior of neural network in predicting software readiness. Neural networks have seen an explosion of interest over the years, and are being successfully applied across a range of problem domains, in areas as diverse as finance, medicine, engineering, geology and physics. Indeed, anywhere that there are problems of prediction, classification or control, neural networks are being introduced. Neural network can be used as a predictive model because it is sophisticated modeling techniques capable of modeling complex functions. It can learn by example. In order to make a neural network useful, the user needs to gather representative data, and then invokes training algorithms to train the neural network. We use software metrics data including object-oriented design metrics to train the neural network.

If we want to improve the software we develop, we must measure our designs by well-defined standards. Meaningful software metrics are used for measuring project progress and quality. They provide help for developers and managers to promote better designs, more reusable code, and better estimates. The software metrics can be divided into project metrics and design metrics. Project metrics are used to predict project needs, such as staffing levels and total effort. They also measure the dynamic changes that have taken place in the state of the project, such as

CHAPTER 1

"how much has been done" and "how much is left to do". These metrics are more global and less specific than the design metrics. Design metrics are measurements of the static state of the project design at a particular point in time. These metrics are more localized and prescriptive in nature. They look at the quality of the way the system is being built. The quality characteristics are reliability, maintainability, extendibility, usability and reusability [66].

There is great interest in use of the object-oriented approach to software development. With the increasing use of object-oriented methods in new software development there is a growing need to improve current practice in object-oriented design and development [27]. Object-oriented approach has encouraged modern software engineering concepts such as information hiding, data abstraction, encapsulation and concurrency. It is able to facilitate reusability, interoperability. When a software is well designed using object-oriented approach, this software can be easily modified, extended and maintained.

However since the object-oriented paradigm exhibits different characteristics from the procedural paradigm, different software metrics have to be defined and used. An Object Oriented application can be viewed as a collection of objects. Its functionality is achieved by interaction among these objects in terms of messages. OO approach supports N-tier application architecture which is a flexible and reusable application for distribution to any number of client interfaces. Using the N-tier system architecture to develop software will have the following advantages: (i) Clearly defined layers, (ii) loose coupling between layers and (iii) persistent storage [58]. It contains any number of levels arranged above another, each serving distinct and separate tasks. In this approach, application software is split into multiple tiers to ensure "separation of concern". Objects should rely on the interfaces and contracts offered by another object without relying on any underlying implementation details. For example, the Business Logic Tier depends on Data Access Tier to access data. The Business Tier however should never need to know how the data is physically accessed and what the underlying data store is. This is called abstraction. Thus, correct level of abstraction helps build a flexible and scalable application.

CHAPTER 1

It is not an easy job to reach the correct level of abstraction and the correct relationship between classes. Object Oriented Design metrics can be a very helpful measuring technique to evaluate the design stability. They can help to identify if we need to pay further attention to any of the code to make it more maintainable. They can be used in quality estimation. In practice, quality estimation means either estimating reliability or maintainability.

Software metrics were used in conjunction with prediction models to predict different aspects of software such as effort and faults. We aim to build an efficient predictive model by combining power of neural network and software metrics on the N-tier application architecture basis.

1.2 Objective

Most software today is developed so as to be integrated into existing systems. New software components are replaced or added into existing system. They use N-tier application architecture. Our predictive model for software readiness is aimed to be useable for this architecture. For example, when a new business tierclass or module is being developed by inheriting the existing components or classes, software developers need to judge when it will finish testing (including integration testing). Our system predicts estimated faults and maintenance cost for that class by using the product metrics data (e.g. the number of coupled objects, the depthof inheritance tree, and the number of newly added methods) and fault data from historical testing records. By comparing the predicted number of faults and number of faults discovered in testing, software managers can decide whether the softwareare ready to be released or not. Our models can bc uesd as one of the tools to assist software project managers, software quality assurance professions, or others in judging the readiness of software for release.

1.3 Research Questions

The primary research questions for this research are as follows:

- Can we build a predictive model that is able to answer three questions?

CHAPTER 1

- (i) How many faults (defects) are remaining in the programs?
- (ii) How many lines are required to be repaired to correct these errors?
- (iii) How much time is required for the above purpose?
- Can these models be applied to N-tier architecture based applications'?
- How can we predict the number of faults (defects) in PL/SQL files embedded in database applications?
- Can artificial neural networks and genetic training strategy be applied to predicting software quality for object-oriented systems?

1.4 Related Research Area

The prior studies that are related to this research can be classified into four areas:

- software reliability,
- software metrics,
- static analysis tools and
- neural network techniques for predictive models building,

1.4.1 Software Reliability

The proposed predictive models are based on the software reliability models that estimate software readiness by gauging on the residual defects. Residual defects are one of the most important factors that allow one to decide if a piece of software is ready to be released. There are essentially two types of software reliability models – those that attempt to predict: (i) from statistic code characteristics and (ii) from test data. Both types of models aim to predict residual errors of software under development. The first type of models are usually called "defect density" models and use code characteristics in traditional software codes such as lines of code, nesting of

CHAPTER 1

loops, external references, input/outputs, cyclomatic complexity and so forth to estimate the number of defects in the software. Our model belongs to the former. But our models use objected-oriented code characteristics and N-tier application architecture approach. The second type of models is usually called software reliability growth models. A number of analytical models have been proposed for modeling software reliability growth trends e.g. [22, 63, 70]. These models attempt to statistically correlate defect detection data with known functions such as an exponential function using parametric methods. if the correlation is good, the known function can be used to predict future behavior [100]. However, to the best of our knowledge, no model has been reported in the literature for the reliability models that are applicable for N-tier applications. Related literature review is presented in Section 2.1.

1.4.2 Software Metrics

Our prediction model is divided into three parts: (1) prediction model for Presentation Logic Tier (2) prediction model for Business Tier and (3) prediction model for Data Access Tier. The partitioning of the application into N-tier system architecture can also lead to more modular design, which increases maintainability [30]. In N-tier application architecture, each layer serves distinct and separate tasks and different software metrics are needed for each tier. Therefore, a new set of software metrics is required for N-tier applications. The classification scheme was based on the nature of these faults and their relevance to the application architecture. Existing object-oriented metrics and complexity software metrics are used in the Business Tier prediction model. New sets metrics have been proposed for the Presentation Logic Tier and Data Access Tier. These metrics are validated using real world application data which are collected from warehouse management system (WMS) and Hwafuh information system (HIS). Related literature review is presented in Section 2.1.2.

1.4.3 Static Analysis Tools

The metric generation and analyzer tools fall in the category of static analysis tools that analyze programs without running them. Static analysis covers the area of

CHAPTER 1

performing all kinds of tests on the source code of software programs. It is performed to improve and ensure the intrinsic quality of software programs. The static code analysis tools that compute various metrics have been implemented since the last decade. The motivation of the development of our Metrics Extraction and Analyzer tool is that the generated metrics should be based on N-tier application architecture. This Metrics Extraction and Analyzer tool can generate the metrics for Business Tier and Data Access Tier source codes. Metrics extraction tools for the Data Access Tier have not been found in the literature although the Data Access Tier is an essential tier in all applications. Moreover, to be able to automatically generate the principal components, correlation analysis report and descriptive statistics report, the principal component analysis tool has been embedded in this Metrics Extraction and Analyzer Tool. These analysis reports are essential reports for the study of software measurement and software quality management. Related literature review is presented in Section 2.2.

1.4.4 Neural Network Techniques for Building Predictive Models

A variety of traditional statistical techniques are used in software quality modeling. Models are often based on statistical relationships between measures of quality and measures of software metrics. However, relationships between static software metrics and quality factors are often complex and nonlinear, limiting the accuracy of conventional approaches [60]. Artificial neural networks and genetic training strategy are adept at modeling nonlinear functional relationships that are difficult to model with other techniques, and thus are attractive for software quality modeling. The use of neural network model with a genetic training strategy is introduced to improve prediction results for estimating software readiness in this study. Related literature review is presented in Section 2.3.

1.5 Major Contribution of the Thesis

These are the main contributions of this thesis.

CHAPTER 1

- Introducing the use of neural networks in software quality estimation using object-oriented design metrics and evaluation of that prediction model using real word application data. These works have been published in [91, 111].
- Proposing new SQL metrics for prediction of the software development Faults in database application and validation of the metrics using the Warehouse Application Data. These works have been published in [89, 90].
- Building a novel software readiness model for N-tier applications. We have carried out validation using Warehouse Application data and Hwafuh Information System data. This work has been published in [92]. The revised and enhanced versions are submitted to [112, 113].

A metric extraction and analyzer tool has also been developed. It is essential for building and validation of above models.

1.6 Organization of the Thesis

There are seven chapters in this thesis. This chapter has introduced the motivations, objectives, research questions, related research area, and major contributions of this research work. The rest of the thesis is organized as follows.

Chapter 2 discusses related work in the area of software metrics and predictive models for predicting software readiness. It also discusses research work carried out in the field of neural networks and genetic algorithms for software quality estimation. The context of use of features of an automated tool for gathering of source code metrics is also discussed.

Chapter 3 presents the application of neural networks in software quality estimation using object-oriented metrics that are extracted from real world applications; the Human Machine Interface System and Quality Evaluation System.

Chapter 4 presents prediction of the software development faults in PL/SQL files using proposed new SQL metrics. General Regression Neural Network and Ward Neural Network are used to evaluate the capability of this set of SQL metrics in predicting the number of faults in database applications.

CHAPTER 1

Chapter 5 presents a novel approach to estimate software readiness for N-tier applications using a neural network model with genetic training strategy by considering three factors:

- (1) how many faults remain in the programs
- (2) how many changes (effort level) are needed to correct the errors and
- (3) how much time is required to change the programs.

Our prediction model is divided into three parts:

- (1) prediction model for Presentation Logic Tier
- (2) prediction model for Business Tier and
- (3) prediction model for Data Access Tier

Chapter 6 discusses the aims and design of the implementation of the metric extraction and analyzer tool which is capable of generating software metrics which are used as independent variables in software quality prediction models. An analysis of metrics extracted using tool developed is also described in Chapter 6.

Chapter 7 concludes the thesis and gives some suggestions for further research.

Chapter 2

2. Literature Review

2.1 Software Reliability Models

The desire to predict software readiness is essential to any software house. Is the software ready to release now? How many faults are remaining in the programs? How many changes are required to correct the errors? How much time is needed to change the programs? How much is the maintenance cost after the program is placed into production?

In the industry, while there is no universal standard techniques on deciding when to release a software system to their customer, there are several common technique [11, 75, 119] that are used. The decision is usually based on data which has varying importance to different companies. Some of the points considered are:

- Defect tracking,
 - (i) When defect find rates fall below a set level
 - (iii) By comparing the predicted number of faults and number of faults discovered in testing,
- Predicting Software Reliability,
- Acceptance testing,
- When all tests are completed and passed, or failed for known reasons which are not going to be fixed in this release,
- When all requirements have been validated,
- Risk Annalysis,
- When the release date is reached and is iron clad,
- Past Experience and
- When the QA & Development Managers decide it is ready to release.

There are many other factors which can be considered also. But all in all, it seems to be a very subjective decision in most cases. Exactly when should a product

CHAPTER 2

be released depends on what type of product it is, what are the target users, what is the product quality history, etc. If the product is targeted for big money center like banks, etc. where a mistake could cause millions, the standard would be greatly differ than if the product is a freeware. Another factor is whether there is one specific customer or it is a general release piece of software. If there is one specific customer, the release criteria are the result of negotiation with that customer. Occasionally we still need to release, even if all criteria have not been met, but then we try to mitigate the risk by doing an assessment on what we have and what we wanted. Rothman Consulting Group has used defect tracking system to deciding software readiness by using the number of high priority defect as release criteria.

Modern science provides a modeling-tool that can accurately predict these answers. A model describes relationships among "input variables", each representing a fact that may influence a situation. The model may produce scores that guide the analyst's judgment. The prerequisite to building and using models has been to learn a good deal about statistics and modeling. Many types of advanced science are used to make accurate predictions: such as neural network models, intelligent decision engines, profiles, traditional statistical models, business modes, expert rules and context vectors.

Reliability is a significant factor in quantitatively characterizing quality and determining when to stop testing and release software on the basis of predetermined reliability objectives [119]. Software reliability models have been proposed to estimate the number of residual faults, failure rate and reliability of software.

Existing reliability models fall into three general categories:

- Software Reliability Growth Models.
- Models that are based on the structure of the program.
- Error-seeding models.

2.1.1 Software Reliability Growth Models

Software reliability growth models are mathematical functions that describe defect-detection rates. They have asymptotic behavior: The defect-detection rate

CHAPTER 2

decreases as the number of defect detected increased and the total number of defects detected asymptotically approaches a finite value. Infinite failure models assume the code has an infinite number of failures [116]. A number of analytical models have been proposed for modeling software reliability growth trends [50, 71, 73, 79]. The most well-known reliability models are Non-Homogeneous Poisson Process (NHPP) models. They assume the number of faults present in a software product to be a random variable most often assumed to display the behavior of a NHPP [43, 50, 117]. The SRGM are used to predict future failure rates, or the number of residual defects in the code. Zhang et al. [119] have studied fault removal efficiency in software reliability assessment using SRGMs.

The usefulness of connectionist models for software reliability growth prediction was illustrated in [2, 56, 57]. In [56], the applicability of the connectionist approach was explored using various network models, training regimes, and data representation methods. An empirical comparison was made between that approach and live well-known software reliability growth models using actual data sets from several different software projects. Their result showed that neural networks could be used for predicting software reliability growth.

2.1.2 Models that are Based on the Structure of the Program

Because of the close relationship between complexity of the program and reliability, some models attempt to predict software reliability from design parameters of the particular program. These models are also called "defect density" models and use code characteristics such as lines of code, nesting of loops, external references, input/outputs, cyclomatic complexity and so forth to estimate the number of defects in the software.

2.1.2.1 Traditional Software Metrics

Software metrics are being used to help improve the reliability by identifying areas of the software requirements specification and code that can potentially cause errors. Software metrics have been studied in the procedural paradigm as a quantitative means of assessing the software development process as well as the quality of software products. Basili and Perricone [9] have conducted an empirical

CHAPTER 2

investigation on software errors and complexity. Munson et al. [80] explored the use of the statistical technique of discriminant analysis as a tool for the detection of fault-prone programs. Norman E. Fenton and Niclas Ohlsson [37] studied the quantitative analysis of faults and failures using size metrics and complexity metrics. In [47], fault incidence was predicted using software change history and product metrics. Thomas et al. [110] studied the effects of reuse across seven projects in narrow domain from a single development organization. They used basic complexity measures such as McCabe's Cyclomatic Complexity and Halstead's Software Science, as well as counts for various types of declarations and statement usage. They found significant differences between errors associated with new and various types of reused components in terms of the types of errors committed, when errors are introduced, and the effect that the errors have on the development process.

Fenton et al. [38] have studied success, failures and new directions of traditional software metrics based around LOC (or very similar) size counts, defect counts, and crude effort figures (in person months). They used Bayesian Belief nets which are increasingly seen as the best means of handling decision-making under uncertainty.

Nikora et al. have developed methods of predicting the fault content of software systems based on measured characteristics of their structural evolution [94]. They analyzed the measurements of structural evolution and fault counts obtained from the JPL flight software technology development effort. Their results indicate that the measures of structural attributes of the evolving software system are suitable for forming predictors of the number of faults inserted into software modules during their development. The new fault standard also insures that the model so developed has greater predictive validity.

2.1.2.2 Object-Oriented Metrics

There is great interest in the use of the object-oriented approach to software engineering. With the increasing use of object-oriented methods in new software development there is a growing need to both document and improve current practice in object-oriented design and development [19]. Object-oriented metrics have been

CHAPTER 2

proposed to capture the structural quality of object-oriented (OO) code and design [12, 17, 21, 28, 29, 48]. OO metrics are used mainly to predict software engineering activities/efforts such as maintenance effort, error proneness, and error rate. There have been discussions about the effectiveness of metrics in different contexts [3]. Research on metrics for OO software development is limited and empirical evidence linking the OO methodology and project outcomes is scarce [106].

One such set of metrics is the set proposed by Chidamber and Kemerer (CK). A preliminary, conference paper version of these six metrics was first proposed in 1991 [29]. A later paper refined the metrics and collected empirical metrics data at two sites, and presented the empirical distributions [28]. Chidamber and Kemerer also report empirical data from two commercial organizations and suggest ways in which the metrics can be used to manage OO design efforts [28]. In that paper it was suggested that the primary use of the metrics by managers would be to identify outlying values that might reflect sub-optimal design practice.

The validity of OO metrics after controlling the class size has first started by by Emam et al. in 2001 [35]. They investigated whether there is a confounding effect of class size in validation studies of object-oriented metrics. Another study has performed by Subramanyam et al. [106]. They found that OO metrics significantly explain variance in defects after controlling for size. They presented new evidence in support of the association between a subset of CK metrics and defects in two current language environments, namely, C++ and Java.

Recently, empirical models for assessing high-level design quality attributes of object-oriented systems, such as reusability, flexibility, and complexity have been proposed [6]. Chatzigeorgiou [24] proposed a method for assessing the design quality of an object-oriented model in terms of responsibility distribution among the classes of the system by extending a method of link analysis employed for retrieving information from the Web.

2.1.2.3 Empirical Validation of Object-Oriented Metrics Using Logistic Regression

Many researchers used Logistic Regression to validate the various kinds of object-oriented metrics.

In [21], Lionel Briand, Prem Devanbu and Walcelio Melo proposed a comprehensive suite of measures to quantify the level of class coupling during the design of object-oriented systems. That suite has taken into account the different OO design mechanisms provided by the C++ language. They validated their proposed eighteen OO design metrics using Logistic Regression.

Saida Benlarhi and Walcelio L. Melo [10] defined and empirically investigated the quality impact of polymorphism on OO design. They defined measures of two main aspects of polymorphic behaviors provided by the c++ language: polymorphism based on compile time linking decisions and polymorphism based on run-time binding decisions. They validated their measures by evaluating their impact on class fault-proneness, a software quality attribute using Logistic Regression.

Basili et al. have reported on results of using the CK metrics suite to predict the quality (fault proneness) of student C++ programs [8]. In their study, they collected data about faults found in object-oriented classes. Based on these data, they verified how much Fault-proneness is influenced by internal (e.g., size, cohesion) and external (e.g., coupling) design characteristics of OO classes. Using Logistic Regression, five of the six metrics were shown to be useful predictors, and the suite of design metrics was shown to be a more effective predictor of fault proneness than extant code metrics.

Lionel C. Briand et al. [20] have empirically explored the relationship between existing object-oriented coupling, cohesion, and inheritance measures and the probability of fault detection in system classes during testing using Logistic Regression. The systems used for this study were developed by students participating in an upper division undergraduate/graduate level course offered by the Department of Computer Science at the University of Maryland. Testing phase was accomplished

CHAPTER 2

by and independent group composed of experienced software professionals. This group tested all system according to similar test plans and used functional testing techniques, spending eight hours testing each system. It was shown that by using a subset of existing measures, accurate models can be built to predict in which classes most of the faults are likely to lie in.

Mei-Huei Tang, Ming-Hung Kao and Mei-Hwa Chen [107] investigated the correlation between object-oriented design metrics and likelihood of the occurrence of object-oriented fault by using a number of natural faults of industrial real-time systems for the past three years using Logistic Regression. They classified the faults into three types: object-oriented faults, object management faults and traditional faults. They proposed a set of new metrics: Inheritance Coupling (IC), Coupling Between Method (CBM), Number of Object/Memory Allocation (NOMA) and Average Method Complexity (AMC) to identify the dynamic behavior of the program and the scenarios that the instances of the classes are referenced in the program. They validated CK metrics using three industrial real-time systems and the results suggested that Weighted Methods per Class (WMC) can be a good indicator for faulty classes and Response For a Class (RFC) is a good indicator for OO faults. Furthermore, they have presented a set of new metrics were useful indicators of OO fault-prone classes.

Toshihiro Kamiya et al. [55] presented a method to estimate the fault-proneness of the class in the early phase, using several complexity metrics for object-oriented software. They introduced four checkpoints into the analysis/design/implementation phase, and estimate the fault-prone classes using the applicable metrics at each checkpoint. Logistic Regression was used to validate their system.

Emanm and Melo [34] have performed to construct a model to predict which classes in a future release of a commercial Java application will be faulty. The model was then validated on a subsequent release of the same application using Logistic Regression. Their results indicated that the prediction model had a high accuracy. They used ten design metrics, two defined by Chidamber and Kemerer [28] and eight by Briand et al [21].

In [15], the relationships between existing object-oriented coupling, cohesion, and inheritance measures and the probability of fault detection in system classes during testing explored empirically using Logistic Regression. Their univariate analysis has shown that many coupling and inheritance measures are strongly related to the probability of fault detection in a class. Their multivariate analysis results showed that by using some of the coupling and inheritance measures, very accurate models could be derived to predict in which classes most of the faults actually lie. In order to draw more general conclusions and confirm the results, they conducted the same study again using an industrial system developed by professional in [18].

L. Briand et al. [14] built a fault-proneness prediction Logistic Regression model based on a set of OO measures using data collected from a mid-sized Java system, and then applied the model to a different Java system developed by the same team. They then evaluated the accuracy the model's prediction in that system and the model's economic viability using a cost-benefit model.

In [42], empirical study was performed with the data from a commercial Java application using Logistic Regression. They found that Depth of Inheritance Tree (DIT) is a good measure of familiarity and, has a quadratic relationship with fault-proneness. Their hypotheses were confirmed for Import Coupling to other classes, Export Coupling and Number of Children metrics. The Ancestor based Import Coupling metrics were not associated with fault-proneness after controlling for the confounding effect of DIT.

2.1.2.4 Empirical Validation of Object-Oriented Metrics Using Multiple Linear Regression

Wei Li and Sallie Henry [64] have done the Linear Regression analyses of a prediction model incorporating ten object-oriented metrics. Their result has shown that there is a strong relationship between metrics and maintenance effort in object-oriented systems.

Another empirical exploration of the validation of three existing OO design complexity metrics to assess their usefulness in predicting fault proneness using Linear Regression analysis was done by PingYu et al. [118]. They reported the

results of validating ten software metrics that relate to the following software attributes: the size of the software, coupling, cohesion, inheritance, and reuse.

Alshayeb et al. [3] presented an empirical study on the effectiveness of OO metrics in two iterative processes: the short-cycled agile process and the long-cycled framework evolution process using Linear Regressions. They found out that OO metrics are reasonably effective in predicting evolution changes of a system design in the short-cycled XP-like process, but they are largely ineffective in the long-cycled framework evolution process. Their study provides further evidence on the abilities and limitations of OO metrics in predicting maintenance effort.

2.1.2.5 Empirical Validation of Object-Oriented Metrics Using Machine Learning Techniques

Yida Mao, H.A. Sahraoui and H. Lounis [72] proposed an experiment to verify three hypotheses about the impact of three internal characteristics (inheritance, coupling and complexity) of OO applications on reusability. That verification was done through a machine-learning approach C4.5 [93]. To verify the hypotheses, they used six inheritance metrics, 18 design coupling metrics, 12 code coupling metrics and 13 complexity metrics. The results of their experiment show that the selected metrics can predict with a high level of accuracy the classes that can be potentially reusable. The four predictive models obtained give a satisfactory results (74% to 89% of accuracy).

In [13], a fuzzy-based approach for building interface stability estimation models for OO class libraries was presented. They used 13 object-oriented metrics.

2.1.2.6 Summary Remarks for Object-Oriented Metrics Studies

To the best of our knowledge, none of the studies investigated the validation of software metrics according to software architecture tiers. Moreover, neural network models are rarely used to validate the object-oriented metrics. To get a better predictive accuracy, we should use neural network as predictive model for N-tier applications.

CHAPTER 2

2.1.3 Error-seeding Models

A completely different type of model is built on firm statistical ground is the error-seeding model developed by Mills [77]. Gokhale et al.[45] classified error seeding models as data-domain models of software reliability models. The model involves "seeding" the program with a number of known errors. These errors are randomly inserted within the program and an assumption is made that the indigenous (original) and seeded errors are equally likely to be detected in subsequent testing efforts. By testing the program for some amount of time and determining which detected errors are indigenous or seeded, it is possible to estimate the number of original indigenous errors [82]. The effectiveness of error seeding was investigated in [76].

2.2 Static Analysis Tools

The static code analysis tools that compute various metrics have been implemented since last decade. These metrics are primarily size and complexity of various types (lines of code, Halstead, McCabe, etc.). Brian Marick [1983, 1992], Ted Shapin [1989], Thomas E. Dickey [1991], Dan Kozak [1992], Christopher Lott [1994], Brad Appleton [1995] and David Wheeler [2000] implemented tools to collect various kinds of size metrics from source codes [67].

Brian Renaud [67] implemented a collection of tools (control/halstead/kdsi/mccabe) around 1989 using C language. Another maintainability assessment tool for analyzing C programs (hp_mas) was developed under the sponsorship of Hewlett-Packard Corporate Engineering at the University of Idaho in 1992 using C language. Joerg Lawrenz [108] implemented the Ccount tool that collects 77 metrics containing bytes, lines, language keywords, comment bytes, semicolons, block length, nesting depth, and etc. in 1993 using C, yacc and lex. Roger Binns implemented the Cyclo tool that analyzes cyclomatic complexity of a piece of ANSI C or C++ code. It can generate a postscript flowgraph of the functions using c++ in 1993. A rule-based software metrics tool for standard C (metre) was implemented by Paul Long in 1994 using C language. These metrics tools have been re-compiled by Christopher Lott and they can be downloaded from his software engineering resource page [67].

CHAPTER 2

Andrew Cain and Rajesh Vasa have developed the Jmetric that calculates a set of object-oriented metrics from Java for a project at COTAR, Australia start from 1998 [53].

Tim Littlefair implemented the CCCC (C and C++ Code Counter) tool. That is a simple command line tool which parses files in the supported languages and builds a set of reports in HTML of numeric measurements (metrics) on those files. In that tool, metrics are divided into three groups. Procedural metrics include lines of code, lines of comment and McCabe's cyclomatic complexity. Metrics of OO design (proposed by Chidamber and Kemerer) include depth of inheritance tree, number of children, coupling between objects, and weighted methods per class. Structural metrics (based on the work of Henry and Kafura), include fan-in, fan-out and information flow [65].

Could et al. [46] has implemented a static analysis tool for SQL/JDBC applications. It verifies the correctness of dynamically generated query strings and finds known and unknown defects in programs that used JDBC. Another bug detection static analysis tool was implemented by Skevoulis and Jia [101] for Java programs.

Many distributors of metric software tools like the Distributive Software [32], the McCabe & Associate [74], the Quantitative Software Management (QSM) [88], the SET Laboratories [99], the Software Productivity Research Information Center [103], the Testwell [109], the Verifysoft Technology [114], the Power Software [86], the Positive Review Inc. [51], the M Squared technologies [69], the Predicate Logic Inc. [87], and the Software Engineering Research Center [98] etc. have developed various tools to increase the productivity of developments and to reduce the costs of software development using various types of software metrics.

As stated in the Chapter 1, the motivation of the development of our Metrics Extraction and Analyzer tool is that the generated metrics should be based on N-tier application architecture. This Metrics Extraction and Analyzer tool can generate the metrics for Business Tier and Data Access Tier source codes. Metrics extraction tools for the Data Access Tier have not been found in literature although the Data Access Tier is an essential tier in all applications.

CHAPTER 2

2.3 Neural Network Techniques for Building Predictive Models

Neural networks are information processing paradigm modeled after the processes of learning in the cognitive system and the neurological processing ability of the brain and capable of predicting new observations from other after executing a process of learning from existing data. Neural network is one of the Data Mining techniques. In the context of modeling predictive models, neural network models can offer significant improvement over traditional statistical methods because of their ability to capture nonlinear relationships associated with the use of non-compensatory decision rules. Moreover, neural network models do not require the same restrictive assumptions as in traditional statistical models about the relationship between the independent variables and dependent variable(s). Consequently, these models have already been very successfully applied in many diverse disciplines, including biology, psychology, statistics, mathematics, business, insurance, and computer science. Although neural network models have been used in diverse applications area, developing software metric-based neural network predictive models are limited. Neural networks can be used as predictive nets or classification nets. The following studies on metric-based predictive nets are found in literature.

2.3.1 Comparison with Other Models

Dolado [33] presented a validation of the Component-Based method for software size estimation using Multiple Linear Regression, Neural Networks, Genetic Programming. Their studies shows Neural Networks and Genetic Programming have worked well in approximating nonlinear data, whereas Multiple Linear Regression has failed to detect the relationships. In the case of linear relationships, they stated that some of the small improvements obtained by Genetic Programming compared to Multiple Linear Regression come at the expense of the simplicity of the equations, but the majority of the linear equations are rediscovered by Genetic Programming.

Neural networks have been earlier applied in dynamic software reliability growth prediction [57]. They introduced a new approach for static reliability modeling and compared its performance in the modeling of software reliability from software complexity in terms of the predictive quality and the quality of fit with more traditional regression modeling techniques.

They also [62] presented a neural network model for the detection of high-risk programs and compared its performance with that of a discriminant analytic model. For both of these approaches, they built models that predict high-risk modules based upon software complexity metric and fault data. The discriminant model used domain metrics derived from the complexity metric data. The data was collected from a large military telecommunications system. They also explored the behaviour of neural network software quality models [61].

In [60], they presented a discriminant model and a neural network model of the same large telecommunications system, classifying modules as not fault-prone or fault-prone. They modeled a subset of modules representing over seven million lines of code. The set consisted of those modules reused with changes from the previous release. The dependent variable was membership in the class of Fault-prone modules. The independent variables were principal components of nine software design metrics. They compared the neural-network model with a nonparametric discriminant model, and found the neural network model had better predictive accuracy.

2.3.2 Neural Network Models for Classification

Some researchers have used Neural Networks for classification in predictive models. They used Neural Networks to classify module as either fault-prone or not fault prone based on complexity software metrics.

Kapsu Kim and SungBack Hong [49] have studied classification of function block as either fault-prone or not fault prone based on 12 internal and external module complexity metrics using the neural network model. Internal metrics are those that measure an attribute like length, number of control statements, or number of tokens. External module complexity metrics try to quantify the module interactions using the assumption that the static interdependencies involved contribute to the overall complexity of the program units, and ultimately to that of the entire program. They can predict whether the newly added function block is fault prone or not fault prone. They used Factor Analysis prior to developing a neural network model.

Khoshgoftaar et al. [59] presented a case study of real-time avionics software to predict the testability of each module from static measurements of source code

CHAPTER 2

using 13 traditional software product metrics. They defined testability as the probability that a test case will fail if the program has a fault. They found that neural network is a promising technique for building predictive models, because they are able to model nonlinear ties in relationship.

Xu Zhiwei et al. have introduced using the Fuzzy Nonlinear Regression (FNR) modeling technique as a method for predicting classification of fault ranges in software modules [120]. They found that FNR modeling gives useful results using 24 traditional software product containing call graph metrics, control flow graph metrics and statement metrics.

Pizzi has proposed a method in predicting classification of the quality of software systems using a neural network classifier with metric inputs and subjective quality assessment as class labels [95]. Their result demonstrated that multi-layer perception classifier can be used in classification of software quality.

Basili et al. [7] constructed a predictive model of the cost of rework using the C4.5 system [93] for generating a logical classification model. They used measures of internal software product attributes as the predictor variables for the model. They used the model to generate prescriptive coding guidelines to improve programming practices.

In [1], different machine learning techniques have been investigated with regard to their capabilities to generate accurate correctability models. Nineteen complexity metrics have been used in their experiment.

A neural network has been shown to be a useful approach to test case effectiveness prediction [4]. Their model classified whether a test case can expose a fault or not. A neural network is neither system nor test case metric specific. Therefore, it can be used across a variety of testing criteria, test case and coverage metrics, and fault severity levels. Anderson et al. have presented experimental results on using a neural network for pruning a test case set while preserving its effectiveness.

2.3.3 Summary Remarks for Metric-based Neural Network Models

Although metrics-based neural network models have been used as classification nets (as discussed above in Section 2.3.2), developing software metric-based neural network predictive models is rare in the literature. We have introduced metric-based neural network predictive models for estimating software readiness in this research.

2.4 Summary Remarks

We would like to extract main important findings from literature. Some researchers have explored to estimate software readiness by gauging on the residual defects using software metrics. Residual defects are one of the most important factors that allow one to decide if a piece of software is ready to be released. One of the aim of our thesis is not only to predict the number of faults but also the number of code changes required to correct faults and the amount to time to make these changes depending on characteristics of codes for example coupling, cohesion and complexity. Therefore, we have chosen the reliability models that are based on structure of the programs using software metrics.

However, to the best of our knowledge, no model has been reported in the literature for the reliability models that are applicable for N-tier applications. As each layer serves distinct and separate tasks in N-tier applications, different sets of software metrics are needed for each tier. Therefore a new set of software metrics is required for N-tier applications.

Furthermore, none of the studies investigated the validation of software metrics according to software architecture tiers. Using two sets of data collected from the two commercial environments, we conducted an empirical validation of the metrics to examine their predictive capabilities.

Some researchers have explored that neural network model gives better prediction result than statistical models using traditional software metrics to predict software development faults. These metrics are not adequate for detecting object-oriented faults. To get a better predictive accuracy, we should use neural network as

CHAPTER 2

predictive model and traditional software metrics and object-oriented software metrics as independent variables.

Chapter 3

3. Software Quality Prediction Models for Object-Oriented Systems

3.1 Chapter Preface

As presented in Chapter 2, many object-oriented metrics have been proposed over the last decade. Prediction models using object-oriented design metrics can be used for obtaining assurances about software quality. In practice, quality estimation means either estimating reliability or maintainability. The estimated number of defects can also be normalized by a size measure to obtain a defect density estimate. Maintainability is typically measured as change effort. Change effort can mean either the average effort to make a change to a class, or the total effort spent on changing a class.

A variety of statistical techniques are used in software quality modeling. Models are often based on statistical relationships between measures of quality and measures of software metrics. However, relationships between static software metrics and quality factors are often complex and nonlinear, limiting the accuracy of conventional approaches. Artificial neural networks are adept at modeling nonlinear functional relationships that are difficult to model with other techniques, and thus, are attractive for software quality modeling.

We conduct our study in the object-oriented paradigm. However since the object-oriented paradigm exhibits different characteristics from the procedural paradigm, different software metrics have to be defined and used.

Our neural network model aims to predict object-oriented software quality by estimating the number of faults and the number of lines changed per class. We used software metrics including both object-oriented metrics and traditional complexity metrics. Object-oriented metrics used include inheritance related measures, cohesion measures, coupling measures and memory allocation measures.

We also introduce using Ward neural network and General Regression neural network to improve prediction result for estimating software quality. Ward neural network is a backpropagation network with different activation functions. They are applied to hidden layer slabs to detect different features in a pattern processed through a network to lead to better prediction. We use a Gaussian function in one hidden slab to detect features in the mid-range of the data and a Gaussian complement in another hidden slab to detect features for the upper and lower extremes of the data. Thus, the output layer will get different "views of the data". Combining the two feature sets in the output layer leads to a better prediction.

Another architecture that we have chosen is the General Regression Neural Network (GRNN). Specht [102] states that it is a memory-based network that provides estimates of continuous variables and converges to the underlying (linear or nonlinear) regression surface. This is a one-pass learning algorithm with a highly parallel structure. Even with sparse data in a multidimensional measurement space; the algorithm provides smooth transitions from one observed value to another.

3.2 Design of the Study

3.2.1 The choice of neural network architecture

We have performed a preliminary investigation on the library system which was developed by master students participating in Communication Software & Network (full-time/part-time) course offered by the Information Communication Institute of Singapore (ICIS) at the Nanyang Technological University by using commonly used network architectures. The students were grouped into fourteen teams of three students each. Each team was asked to develop a Library system that supports the borrow/return process of a library, and maintains members and book databases. Such an application domain had the advantage of being easily comprehensible and, therefore, system requirements could be easily interpreted by students regardless of their educational background. The students corrected and recorded their system based on the errors found during acceptance testing. The OO analysis/design method was used during the analysis and design phases. The Visual C++ programming language was used during the implementation. This experiment has been performed using OO metrics, N-tier metrics and SQL metrics. The details descriptions of these metrics are

CHAPTER 3

shown in Section 3.3.2, 3.4.2.4.2, 5.2, 5.3 and 5.4. We have presented the comparison of these results in Table 3.1. This experiment results shows that network structure GRNN and Ward are the best to use in software quality estimation experiments among these commonly used network structures.

Table 3.1 Comparison of prediction results using seven neural network structures

			Prediction Results for the Library System					
	Network Architecture		R-square	r	Avg error	MSE	RMSE	t values
3-tier	Back-propagation	Standard	0.677142	0.85927	0.974317	1.794343	1.33953	16.96543
		Jump	0.692586	0.862273	1.006827	1.708511	1.3071	17.19545
		Recurrent (input)	0.67978	0.835776	1.048566	1.779682	1.33404	15.37314
		Recurrent (hidden)	0.661355	0.835997	1.07261	1.882083	1.37189	15.38661
		Recurrent (output)	0.670577	0.848147	1.068226	1.830834	1.35308	16.16931
		Ward	0.718808	0.905377	0.991899	1.562781	1.25011	21.53482
	GRNN		0.745229	0.866163	0.687336	1.415936	1.18993	17.50397
OO	Back-propagation	Standard	0.601766	0.806847	1.176207	2.213264	1.48770	13.79368
		Jump	0.555207	0.774287	1.280172	2.472023	1.57226	12.35697
		Recurrent (input)	0.57861	1.572267	1.157779	2.341957	1.53034	12.80323
		Recurrent (hidden)	0.534026	0.776726	1.31365	2.589741	1.60926	12.45484
		Recurrent (output)	0.520224	0.761293	1.279909	2.666448	2.62422	11.8578

CHAPTER 3

		Ward	0.647257	0.809568	0.904521	1.960436	1.40015	13.92824
		GRNN	0.70565	0.849196	0.955187	1.635905	1.27902	16.24093
SQL	Back-propagation	Standard	0.414577	0.70966	1.501638	3.253599	1.80377	10.17284
		Jump	0.500652	0.83727	1.41509	2.775222	1.6659	15.46482
		Recurrent (input)	0.507027	0.757601	1.399361	2.73979	1.65523	11.72235
		Recurrent (hidden)	0.447877	0.701414	1.417259	3.06853	1.75172	9.938826
		Recurrent (output)	0.446	0.701587	1.383436	3.078959	1.75469	9.94366
		Ward	0.528833	0.875315	1.363075	2.618603	1.61821	18.28188
		GRNN	0.532874	0.738745	1.204762	2.596141	1.61125	11.06993

3.2.2 Neural network modeling

The first neural network architecture that we have chosen is the Ward Network [83] as shown in Figure 3.1. Backpropagation network is the most popular network for practical applications. We chose three layer Backpropagation Ward neural network because it is an effective network for most applications and trains much more quickly than 4 or 5 layer networks. We use three different activation functions. It has three slabs (slab2, slab3 and slab4) in the hidden layer. Hidden layers in a neural network are known as feature detectors. A slab is a group of neurons. Each slab in the hidden layer has a different activation function; this offers three ways of viewing the data. We use a linear function for the output slab (slab5). The hyperbolic tangent (tanh) function is used in one slab of hidden layer (slab3) because it is better for continuous valued outputs especially if the linear function is used on the output layer. The Gaussian function is used in another slab of the hidden layer (slab2). This function is unique, because unlike the others, it is not an increasing function. It is the classic bell shaped curve, which maps high values into low ones, and maps mid-range

CHAPTER 3

values into high ones. The Gaussian Complement is used in the third slab of the hidden layer (slab4) to bring out meaningful characteristics in the extremes of the data. We use a smaller learning rate 0.1 and momentum 0.1 as our network is a predictive network where outputs are continuous values rather than categories.

Figure 3.1. Ward Neural Network

Another neural network architecture that we have chosen is the General Regression Neural Network (GRNN) as shown in Figure. 3.2. GRNN is based on a one-pass learning algorithm with a highly parallel structure. GRNN is a powerful memory based network that could estimate continuous variables and converge to the underlying regression surface. The strength of GRNN is that it is able to deal with sparse data effectively. GRNNs feature fast training times, can model non-linear functions, and have been shown to perform well in noisy environments given enough data. Specht [102] has shown that the algorithm in GRNN is able to provide a smooth transition from one observed value to another, even with sparse data in a multidimensional measurement space. GRNN applications are able to produce continuous valued outputs. For GRNN networks, the number of neurons in the hidden layer (Slab2) is usually the number of patterns in the training set because each pattern in the training set is represented by one neuron. The number of neurons in the input layer (Slab1) is the number of inputs, and the number of neurons in the output layer (Slab3) corresponds to the number of outputs.

Figure 3.2. GRNN Network

The primary advantage of GRNN is the speed at which the network can be trained. There are no training parameters such as learning rate and momentum in

backpropagation network, but there is a smoothing factor that is applied after the network is trained. Smoothing factor is the only adjustable parameter in GRNN. Therefore, making overtraining is less likely in GRNN. The smoothing factor allows the GRNN to interpolate between patterns or spectra in the training set. The smoothing factor determines how tightly the network matches its predictions to the data in the training patterns. For GRNN networks, the smoothing factor must be greater than 0 and usually ranges from 0.01 to 1 with good results.

3.2.3 Principal component analysis

If a group of variables in a data set are strongly correlated, these variables are likely to measure the same underlying dimension (i.e., class property) of the object. Many object-oriented metrics have high correlation with each other. Principal components analysis transforms raw data into variables that are not correlated to each other. If correlated raw software metrics were used directly, the neural network models did not train satisfactorily, but transforming raw data with principal components analysis facilitated training. Principal component analysis (PCA) is a standard technique to identify the underlying, orthogonal dimensions that explain relations between the variables in a data set. Principal components (PCs) are linear combinations of the standardized independent variables. It is also a data reduction technique. The varimax rotation method was used in this study. It is an orthogonal rotation method that minimizes the number of variables that have high loadings on each factor [105]. It simplifies the interpretation of the factors. We selected only PCs whose Eigen values are larger than 1.0. (Principal components with Eigen values less than one are ignored as the components are accounting for less variance than the original variables).

3.3 Prediction of number of faults

3.3.1 Description of data

Faults appear when a program does not perform according to users' specification during testing and operations stages. The applications used in this prediction are three subsystems of HMI (Human Machine Interface) software, which is a fully networked Supervisory Control and Data Acquisition system [107]. This

CHAPTER 3

software, which consists of more than 200 subsystems and 3 million lines of code, has been used by many manufacturing companies for several years. Although each subsystem selected plays a different role in the system and performs a different functionality, they share some similar characteristics that meet our selection criteria. Subsystem A is a user interface-oriented program that allows customers to configure the basic product operations and device communications. It consists of 20 classes that define 256 new, re-defined or virtual functions, and approximately 5600 lines of code in length. Subsystem B is a real time data logging process that collects data as needed and logs data into the database, based on the user configuration. This subsystem defines 48 classes and 353 new, re-defined or virtual functions, comprising approximately 21,300 lines of code. Subsystem C is a communication-oriented program that acts as a router, not only delivering messages between processes within the same host but also forwarding messages to other hosts. This subsystem defines 29 classes and 293 new, re-defined or virtual functions and contains approximately 16,000 lines of code [107].

3.3.2 Selection of metrics

As discussed in Section 1, we are introducing the research on software defects prediction into the object-oriented paradigm using neural networks. We have selected the object-oriented metrics that have a strong relationship with software quality. To indicate the presence of software defects the following existing metrics [28, 107] are used.

Depth of Inheritance Tree (DIT) of a class is the length of the longest path from the class to the root in the inheritance hierarchy. This determines the complexity of a class based on its ancestors, since a class with many ancestors is likely to inherit much of the complexity of its ancestors. The deeper a class is in the hierarchy, the greater the number of methods it is likely to inherit making it more complex to predict its behavior. Therefore, the more likely it is to contain a fault.

Number of Children (NOC) measures the number of immediate descendants of a particular class. This measures an amount of potential reuse of the class. The more reuse a class might have, the more complex it may be, and the more classes are

CHAPTER 3

directly affected by changes in its implementation. This increases the magnitude of ripple effects. Therefore we selected the NOC metric to predict the number of faults.

Coupling Between Objects (CBO) is defined as the number of other classes to which a class is coupled. Coupling measures the degree of inter dependence among the components of a software system. High coupling makes a system more complex; highly interrelated modules are harder to understand, change or correct and thus likely to be more fault-prone.

Response For a Class (RFC) is the number of methods that can potentially be executed in response to a message received by an object of that class. The response set of a class consists of the set of M methods of the class, and the set of methods directly or indirectly invoked by methods in M. The number of methods that could potentially respond to a message indicates the complexity of that class. Therefore RFC can be used as a predictor variable for the number of faults.

Inheritance Coupling (IC) provides the number of parent classes to which a given class is coupled. A class is coupled to its parent class if one of its inherited methods is functionally dependent on the new or redefined methods in the parent class. When a data member, which is used by an inherited method, is modified by a new or redefined method, it is likely to introduce new faults into the inherited method.

Coupling Between Methods (CBM) provides the total number of new/redefined methods in which all the inherited methods are coupled. CBM measures the total number of function dependency relationships between the inherited methods and new/redefined methods. We have chose CBM to predict the number of faults because it can measure the functional dependency complexity at the methods level.

Weighted Methods per Class (WMC) is defined as a function of the number of all member functions and operators in each class. We have selected WMC to measure the complexity of an individual class.

Number of Object/Memory Allocations (NOMA) measures the total number of statements that allocate new objects or memories in a class. A class with more

CHAPTER 3

object/memory allocating activities tends to introduce more object management faults related to object copying, dangling reference, object memory usage faults and so on. Therefore NOMA can be used as a predictor variable for software quality.

First, we performed a preliminary analysis using multiple regressions. As shown in Appendix A, we got an R-square value of 0.856. R-square is a regression quality indicator that measures how much variance in the dependent variable is accounted for by the independent variables in the sample. About 85% of the variation in the criterion variable, the number of faults in the class, can be explained by the regression model with all predictors WMC, DIT, NOC, CBO, RFC, IC, CBM and NOMA. The adjusted R-square corrected for the number of predictors equals 0.843. So therefore, 84% of the total variance in the number of faults is accounted for by the metrics in the population. The observed significance level is less than 0.001. We can conclude that the prediction of the number of faults from the above metrics is possible.

3.3.3 Experiments

Relationships between static software metrics and quality factors are often complex and nonlinear, limiting the accuracy of conventional approaches. Artificial neural networks are adept at modeling nonlinear functional relationships that are difficult to model with other techniques, and thus, are attractive for software quality modeling.

First, each data pattern was examined for erroneous entries, outliers, blank entries and redundancy. The descriptive statistics for metrics and number of faults per class is summarized in Appendix A. We standardized the metrics to a mean of zero and a variance of one for each metric. Many raw software metrics have incompatible units of measures. This step converts all of them to the unit of one standard variation. After standardizing the metric data, we performed the principal component analysis. Table 3.2 presents the relationship between the original object-oriented metrics and the domain metrics for HMI system.

CHAPTER 3

Table 3.2. Rotated Principal Components for HMI system

Metrics	PC1	PC2	PC3
WMC	0.9068	-0.0544	-0.1768
DIT	-0.0358	-0.0400	0.9286
NOC	-0.0274	0.8710	-0.0694
CBO	-0.0043	0.8508	0.0471
RFC	0.9399	-0.0818	-0.0919
IC	0.6452	0.1678	0.5140
CBM	0.8636	0.0811	0.2932
NOMA	0.6216	-0.1029	0.4508
Eigenvalues	3.2560	1.5390	1.4620
% Variance	40.7030	19.2380	18.2790
Cummulative % Variance	40.7030	59.9400	78.2190

For HMI system, PCA identified three sets of principal components (PCs), which capture 40.7%, 19.24%) and 18.28% respectively of the data set variance, which gives a representation of about 78.22%) of the population. Table 3.2 shows the coefficient measure for each rotated component, with the coefficients larger than 0.6 set in boldface. The Eigen value, the percentage of the data set described, and the cumulative variance percentages are also shown. Based on the analysis of the coefficients associated with each metric within each of the three sets of rotated components, the PCs are interpreted as follows.

The first principal component shows high correlation between metrics with RFC, WMC, CBM, IC and NOMA. The second principal component is highly correlated with NOC and CBO. The third principal component shows highest correlation with DIT.

We divided our data into training, testing, and production sets using 3:1:1 ratio, which is the commonly accepted proportion used by most neural network researchers. We randomly extracted 19 patterns for the test set and another 19 patterns for the production set. The remaining 59 patterns are used as training set. We used the production data set to evaluate model performance.

The dependent variable was the number of faults per class and the independent variables were the three principal components identified above (out of eight software

CHAPTER 3

metrics). We used both the Ward network and GRNN network for predicting the number of defects. The Ward neural design summary is presented in Table 3.3.

Table 3.3. Ward neural network architecture used

	Slab1	Slab2	Slab3	Slab4	Slab5
No. of neurons	3	4	4	4	1

The GRNN usually requires one hidden node for each training sample. Therefore we used 97 neurons in the hidden layer, 3 neurons in the input layer and one neuron in the output layer as shown in Table 3.4. In this study, the value of the smoothing factor is 0.05507813.

Table 3.4. GRNN neural network architecture used

	Input layer	Hidden layer	Output layer
No. of neurons	3	97	1

3.3.4 Experiment results

To measure the goodness of fit of the model, we use the coefficient of multiple determination (R-square), the coefficient of correlation (r), mean squared error, mean absolute error, minimum absolute error and maximum absolute error. These statistical measures are shown in Table 3.5. The correlation of the predicted change and the observed change is represented by the coefficient of correlation (r). An r value of 0.9476 in Ward neural network and 0.9531 in GRNN network represent high correlation between the predicted value and the observed value. The number of observations is 97. The significance level is indicated by the p value [41]. Most of software quality prediction studies in the literature used significance testing with p-value [78]. A commonly accepted p value is 0.05. In our experiment, the two tailed probability p value is less than 0.001. This shows a high degree of confidence for the successful validations. The results clearly indicate a close relationship between metrics (independent variables) and the number of faults per class in software applications (dependent variable).

CHAPTER 3

Table 3.5. Experimental result for HMI system

	Ward	GRNN
R-square	0.8715	0.9077
r (correlation coefficient)	0.9476	0.9531
r- square	0.8979	0.9084
Mean squared error	1.5840	1.1380
Mean absolute error	0.8230	0.7650
Min absolute error	0.0010	0.0000
Max absolute error	6.2110	4.2950
t values	28.88816	28.8815
p values	<0.001	<0.001

3.3.5 Results from a 10-cross-validation study

We performed a 10-cross-validation of the prediction neural network models as shown in Table 3.3 and Table 3.4. It is a common technique to evaluate learning algorithms on a dataset [15]. The 97 data points were randomly split into 10 partitions or folds of roughly equal size (eight partitions of 10 data points each, one partition of nine data points and one partition of eight data points). For each fold, the following processing is performed. For each iteration, the selected fold is become the test set, and the other 9 folds are combined into the training set. Create a neural network and train it on the training set and test it on the test set. We performed the process 10 times and took a mean accuracy to obtain the estimated performance of the neural network. The results are summarized in Table 3.6 for the GRNN network and in Table 3.7 for the Ward neural network.

CHAPTER 3

Table 3.6. Results from 10-cross validation for GRNN network

Partition	R-square	r squared	Correlation coefficient r
1	0.9151	0.9170	0.9576
2	0.8863	0.8888	0.9428
3	0.8563	0.8623	0.9286
4	0.8963	0.8988	0.9481
5	0.8485	0.8534	0.9238
6	0.9486	0.9493	0.9743
7	0.9355	0.9365	0.9677
8	0.6425	0.6992	0.8362
9	0.9137	0.9154	0.9567
10	0.9580	0.9582	0.9789
Average	0.8801	0.8879	0.9415

Table 3.7. Results from 10-cross validation for Ward Network

Partition	R-square	r squared	Correlation coefficient r
1	0.8834	0.8838	0.9401
2	0.8749	0.8793	0.9377
3	0.7417	0.8178	0.9043
4	0.8593	0.8699	0.9327
5	0.8593	0.8699	0.9327
6	0.8724	0.8762	0.936
7	0.8765	0.8915	0.9442
8	0.8765	0.8915	0.9442
9	0.8825	0.8887	0.9427
10	0.8701	0.8779	0.9369
Average	0.8596	0.8746	0.9351

The average value of the correlation between predicted faults and actual faults is 0.9415 in the GRNN network and 0.9351 in the Ward network. They are statistically significant. The average squared multiple correlation is 0.88008 in the GKNN network and 0.8596 in the Ward network. Therefore about 88% and 86% of the variance, respectively, in the number of faults can be accounted for by the two predictors.

3.4 Prediction of maintenance effort

3.4.1 Description of data

This investigation attempted to predict the maintenance effort. The commercial software product QUES(Quality Evaluation System) data is used in this investigation [64]. The maintenance effort is measured by counting the number of lines changed per class. A line change could be an addition or a deletion. A change of the content of a line is counted as a deletion followed by an addition. The QUES system was designed and developed with Class-Ada.

3.4.2 Selection of metrics

We have selected the software metrics that have a strong relationship with software maintainability. To predict the maintenance effort, DIT, RFC and WMC that are previously defined in Section 3.3.2 and the following metrics are used.

Message Passing Coupling (MPC) gives an indication of how many messages are passed among objects of the classes. The number of messages sent out from a class indicates how dependent the implementation of the local methods is on the methods in other classes.

Lack of Cohesion in Methods (LCOM) is the number of pairs of methods in the class using no attributes in common, minus the number of pairs of methods that do. If this difference is negative, LCOM is set to zero. The cohesion of a class is characterized by how closely the local methods are related to the local instance variables in the class. It is harder to maintain a class that has a larger LCOM metric value because, if all the methods defined in a class access many independent sets of data structures encapsulated in the class, the class may not be well designed and partitioned. LCOM can be used to predict the maintainability of the class.

Data Abstraction Coupling (DAC) is the number of attributes in a class that have as their type another class. This metric measures the coupling complexity caused by abstract data type (ADT). The number of variables having ADT type may indicate

CHAPTER 3

the number of data structures dependent on the definitions of other classes. It is more difficult to maintain the class that has more ADT's.

The number of local methods (NOM) defined in a class indicates the operation property of a class. The more methods a class has, the more complex will be the class's interface.

Size metrics have been used as software metrics for a long time. Two size metrics are used in this prediction. The first size metric (SIZE1) is the traditional line of code metric, which is calculated by counting the number of semicolons in a class. The second size metric (SIZE2) is the total number of attributes and methods of a class.

We performed the preliminary analysis using multiple regressions. As shown in Appendix B, we got an R-square value of 0.734. About 73% of the variation in the criterion variable maintenance effort can be explained by the regression model with all predictors DIT, MPC, RFC, LCOM, DAC, WMC, NOM, SIZE1 and SIZE2. The adjusted R-square is 0.694. So therefore, 69% of the total variance in the number of faults is accounted for by the metrics in the population. The observed significance level is less than 0.001. We can conclude that the prediction of maintenance effort from the above metrics is possible.

3.4.3 Experiments

As in this experiment, first, each data pattern was examined for erroneous entries, outliers, blank entries and redundancy. The descriptive statistics for metrics and number of lines changed per class is summarized in Appendix B. After standardizing the metric data, we performed the principal component analysis. Table 3.8 presents the relationship between the original object-oriented metrics and the domain metrics for QUES system.

CHAPTER 3

Table 3.8. Rotated principal components for QUES system

metrics	PC1	PC2	PC3
DIT	0.060	0.027	0.966
MPC	-0.023	0.966	0.037
RFC	0.877	0.333	0.043
LCOM	0.869	-0.156	0.059
DAC	0.796	0.027	0.427
WMC	0.832	0.258	-0.27
NOM	0.971	-0.132	0.097
SIZE1	0.812	0.475	-0.089
SIZE2	0.963	-0.093	0.190
Eigenvalues	5.384	1.388	1.248
% Variance	59.826	15.424	13.863
Cummulative % Variance	59.826	75.250	89.113

For the QUES system, PCA identified three PCs, which capture 89% of the data set variance; Table 3.8 shows for each rotated component the coefficients of the measure, with coefficients larger than 0.6 set in boldface. The Eigen value, the percentage of the data set variance each PC describes, and the cumulative variance percentage are also provided. Based on the analysis of the coefficients associated with each metric within each of the three rotated components, the PCs are interpreted as follows.

The first component is highly correlated with NOM, SIZE2, RFC, LCOM, WMC, SIZE1 and DAC. NOM is the best representative, however, because it is less correlated with the other two components. The second component is most highly correlated with MPC. The third component is most highly correlated with DIT. This suggests that NOM, MPC and DIT metrics should be focused on in further analysis for this system.

We divided the data into training, testing, and production sets using a 3:1:1 ratio. Test set is used to prevent over training the network so they will generalize well. We used the production data set to evaluate model performance. The network's results can be tested with the data the network has never seen before.

CHAPTER 3

We used the Ward network and the GRNN network for predicting the number of changes. Table 3.9 shows the summary of the Ward network design. In our General Regression neural network design, there were 71 neurons in the hidden layer, 3 neurons in the input layer and one neuron in the output layer as shown in Table 3.10. In this study, the value of the smoothing factor is 0.02460938.

Table 3.9. Ward neural network architecture used for QUES system

	Slab1	Slab2	Slab3	Slab4	Slab5
No. of neurons	3	3	3	3	1

Table 3.10. GRNN neural network architecture used

	Input layer	Hidden layer	Output layer
No. of neurons	3	71	1

3.4.4 Experiment results

The summary results in terms of the goodness of fit of the model are shown in Table 3.11. The correlation of the predicted change and the observed change is represented by the coefficient of correlation (r). The coefficient of correlation values of 0.747 in the Ward neural network and 0.8590 in the GRNN network represent high correlations between the actual versus predicted output. The number of observations is 71. Two tailed probability p values less than 0.001 shows a high degree of confidence for the successful validations. We conclude that the impact of model prediction is valid in the population.

In the evaluation of the accuracy of the prediction of maintenance effort models we also included measures based on the magnitude of the relative error (MRE):

$$\text{MRE} = |\text{actual effort} - \text{predicted effort}| / \text{actual effort}$$

MRE based accuracy measures are meaningful for the maintenance managers [54]. MMRE is the mean magnitude of relative error of a prediction relative to a data set. It is regularly used to compare the accuracy of cost estimation models and

CHAPTER 3

modeling techniques [16, 52, 68]. To overcome the shortcoming of MMRE which may be strongly influenced by a few very high MRE values, we included the median magnitude of relative error (MdmRE) in this evaluation.

In addition, to evaluate model performances we computed the following variants of the measure PRED [31, 54, 68]:

$$\text{PRED}_{25} = \% \text{ of the observations with RE } \leq 0.25$$

$$\text{PRED}_{50} = \% \text{ of the observations with RE } \leq 0.50$$

As shown in Table 3.1 1, GRNN network model can predict better than Ward network, although the performances of Ward network can also be considered rather acceptable. In particular, GRNN network predicts about 76% of the cases within a relative error less than 25% (PRED_{25}) and almost 89% of the cases with a relative error less than 50% (PRED_{50}). Also, the mean relative error is 19.52%. Maintenance managers may, in most cases and especially for small maintenance tasks, accept a relative error between the actual and predicted effort of about 50% [68]. Accurate prediction is difficult: at the beginning of a software development or maintenance process, an error of 100% in the estimates can still be considered 'good' and an error of 32% 'outstanding' [115]. In the effort estimation literature the prediction performances of a model are considered good whenever the estimates are within 25% of actual at least 75% of the time [31]. From the results presented above, the metrics chosen in this study appear to be useful in predicting maintenance effort.

CHAPTER 3

Table 3.11. Experimental result for QUES system

	Ward	GRNN
R-square	0.5545	0.7220
r (correlation coefficient)	0.7470	0.8590
r- square	0.5580	0.7379
Mean squared error	817.004	509.7900
Mean absolute error	20.7820	12.1820
Min absolute error	0.0940	0.0000
Max absolute error	114.1610	109.3850
MMRE	44.97%	19.52%
MdMRE	30.00%	13.38%
PRED ₂₅	49.30%	76.06%
PRED ₅₀	67.61%	88.73%
t values	9.3290	13.9848
p values	<0.001	<0.001

3.4.5 Results from a 10-cross-validation study

Results from the 10-cross-validation of the prediction neural network models are shown in Table 3.12 and Table 3.13. The 71 data points were randomly split into 10 partitions or folds of roughly equal size (nine partitions of seven data points each and one partition of eight data points),

Table 3.12. Results from 10-cross validation for GRNN network

Partition	R-square	r squared	Correlation coefficient r
1	0.7553	0.7787	0.8825
2	0.7984	0.8286	0.9103
3	0.7539	0.8476	0.9207
4	0.6040	0.6100	0.7810
5	0.6032	0.6084	0.7800
6	0.7759	0.8149	0.9027
7	0.7215	0.7393	0.8598
8	0.7345	0.7932	0.8906
9	0.5709	0.5752	0.7584
10	0.7963	0.8004	0.8946
Average	0.7114	0.7396	0.8581

CHAPTER 3

Table 3.13. Results from 10-cross validation for Ward network.

Partition	R-square	r squared	Correlation coefficient r
1	0.5723	0.5924	0.7697
2	0.5046	0.5915	0.7691
3	0.5510	0.5962	0.7721
4	0.6472	0.655	0.8093
5	0.544	0.5619	0.7496
6	0.5787	0.5921	0.7695
7	0.6277	0.6382	0.7989
8	0.5517	0.5154	0.7179
9	0.5048	0.5153	0.7178
10	0.5247	0.5415	0.7359
average	0.5607	0.5799	0.7609

The average value of the correlation between maintenance effort and actual maintenance effort is 0.8581 in GRNN network and 0.7609 in the Ward network. They are statistically significant. The average squared multiple correlation is 0.7114 in the GRNN network and 0.5607 in the Ward network. Therefore about 71% and 56%, respectively, of the variance in the maintenance effort can be accounted for by the two predictors.

3.5 Conclusion

The neural network modeling techniques used in this study are applicable to improving the quality of software products. This chapter presents GRNN and Ward neural network models of the Human Machine Interface system and Quality Evaluation System, predicting the number of software defects and the maintenance effort. The independent variables were principal components of software design metrics. Table 3.5 and Table 3.11 summarize the prediction results of the neural network models. The two tailed probability p values are less than 0.001. That shows a high degree of confidence for the successful validations.

In the prediction of the number of faults experiment, the correlation between predicted faults and actual faults is 0.9531 in the GRNN network and 0.9476 in the Ward network. In the maintainability prediction experiment, the correlation between

CHAPTER 3

predicted maintenance effort and actual maintenance is 0.8590 in the GRNN network and 0.747 in the Ward network. They are statistically significant. The GRNN network model is found to predict more accurately than the Ward network model. Their comparison of prediction results for production sets are shown in Figure 3.3 and Figure 3.4. Although we included measures based on the magnitude of the relative error (MRE) in the evaluation of the accuracy of the prediction of maintenance effort models, we could not include these measures in the prediction of number of faults models because of containing zero value of number of actual faults in HMI data set as shown in Appendix A.

From the results presented above, the object-oriented metrics chosen in this study appear to be useful in predicting software quality. These software metrics are significantly related to the number of faults and the maintenance effort. These prediction models can be used to forecast the group membership of modules from a subsequent release or a similar system developed in the same environment.

Figure 3.3. Comparison chart for number of faults prediction

CHAPTER 3

Figure 3.4. Comparison chart for number of line changes prediction

Chapter 4

4. Prediction of Software Development Faults in PL/SQL Files

4.1 Chapter Preface

Software metrics have been used as a quantitative means of assessing software development process as well as the quality of software products. Many researchers have studied the correlation between software design metrics and the likelihood of occurrence of software faults. They classified the software modules (or classes) as not fault-prone or fault-prone modules (or classes) and predicted the number of faults in modules (or classes) using various software metrics [13, 14, 15, 17, 34, 36, 37, 39, 40, 47, 59, 96 etc.].

Although database applications are essential to every organization, studies on product measures for static database operation statements are rarely been studied and validated. Structured Query Language (SQL) is the standard language for relational database management systems. The relationship between the fault occurrence for database applications and SQL metrics is studied in this chapter. We proposed new SQL metrics that have strong relationship with faults and then performed empirical validation for these metrics.

In our study, we analyzed the fault reports of development projects involving database applications using PL/SQL code and found that faults are related to the number of SQL statements and the complexity of SQL statements. SQL commands are mainly composed in the PL/SQL files to perform database operations. SQL complexity can be measured using product metrics such as the number of table names in from-clauses: the number of join-queries criteria in where-clauses.

This chapter aims at exploring the relationships between product measures for database operations statements and the probability of fault detection in system during testing and maintenance. First, we have defined SQL metrics. Based on actual project defect data of commercial applications (collected from the journal files that contain

CHAPTER 4

the documentation of all changes in source files), the SQL metrics will be empirically validated by analyzing their relationship with the probability of fault detection across SQL files. In this study, the General Regression Neural Network (GRNN) and Ward Neural Network are used to predict faults in PLISQL codes using proposed SQL metrics.

4.2 SQL Metrics

Database application constitutes one of the largest and most important software domains in the world [96]. Some of classes or modules in those applications are responsible for handling database accesses. We analyzed the fault reports of these classes or modules and found that faults are related to the number of SQL statements, which are invoked From a class or module; and the complexity of SQL statements. For example, retrieving wrong database records. In such cases developers need to check and modify the corresponding SQL statements to correct the error. To be able to predict the number of such faults for these classes or modules are very important in developing database applications.

The following SQL metrics are defined and used in this study. Metrics having weak relationships with fault occurrences, such as the number of Data Definition Language (DDL) commands and Data Control Language (DCL) commands are omitted in this study.

Table 4.1. Proposed SQL metrics

SQL metrics	Description
NSC	Number of select commands
NIUO	Number of insert/update operations
NDO	Number of delete operations
NT	Number of table in from-clause
NSCC	Number of search condition criteria in where-clause
NJQ	Number of join-queries criteria

4.3 Data Collection

The expenment data are collected from a set of warehouse management applications that is developed using C, JAM and PL/SQL languages. This set of applications has more than a thousand source files of C, JAM and PL/SQ codes and

CHAPTER 4

uses Oracle database. The warehouse system has been customized and used by many companies. Data access faults were collected from the journal files that contain the documentation of all changes in source files such as status of module, start date, end date, developer, nature of changes, etc. Data on six SQL metrics (see Table 4.1) were extracted from 108 PL/SQL files of the above mentioned warehouse application using metric extraction tool that we developed using VC++ incorporating MKS LEX & YACC utilities as embedded languages. This data set is shown in Appendix C.

4.4 Experiment

Firstly, each data pattern was examined for erroneous entries, outliers, blank entries and redundancy. The descriptive statistics for metrics and number of faults per class is summarized in Appendix D. We standardized the SQL metrics to a mean of zero and a variance of one for each metric. Many raw software metrics have incompatible units of measures. This step converts all of them to a unit of one standard variation. After standardizing SQL metric data, we performed principal component analysis (PCA). It is a standard technique to identify the underlying, orthogonal dimensions that explain relations between the variables in the data set. The varimax rotation method was adopted in this study. Table 4.2 presents the relationship between the original SQL metrics and the domain metrics, based on experiment data extracted from the warehouse management applications.

Table 4.2. Principal Components

SQL metrics	PC1	PC2
NSC	.962	.116
NIUO	.029	.872
NDO	.052	.859
NT	.979	.101
NSCC	.967	-.038
NJQ	.823	.008
Eigenvalues	3.500	1.524
% Variance	58.336	25.394
Cumulative % Variance	58.336	83.730

CHAPTER 4

PCA identified two sets of principal components (PCs), which capture 58.34% and 25.39% respectively of the data set variance, which gives a representation of about 84% of the population. Table 4.2 shows the coefficient measure for each rotated component, with coefficients larger than 0.8 set in boldface. The Eigen value, the percentage of data set each PC describes, and the cumulative variance percentage are also shown. Based on the analysis of the coefficients associated with each metric within each of the two sets of rotated components, the PCs are interpreted as follows:

The first set of principal components shows high correlation between metrics NT, NSCC, NSC and NJQ. However NT is a better representative, because it is less correlated with the other components. The second set of principal components shows high correlation between metrics NIUO and NDO. We therefore chose NT and NIUO metrics further analysis.

We divided our data into training, testing, and production sets using 3:1:1 ratio, which is the commonly accepted proportion used by most neural network researchers. We extracted 21 patterns for the test set and another 21 patterns for the production set. The remaining 66 patterns are used as training set. We used the production data set to evaluate model performance.

The dependent variable was the number of software faults and the independent variables were the two principal components identified above (out of the six SQL metrics). We used both Ward network and GRNN network for predicting faults. Table 4.3 shows the summary of Ward network design whereas Table 4.4 shows the structure of GRNN network used.

Table 4.3. Ward neural network architecture used

	Slab1	Slab2	Slab3	Slab4	Slab5
No. of neurons	2	3	3	3	1

CHAPTER 4

Table 4.4. GRNN architecture used

	Input layer	Hidden layer	Output layer
No. of neurons	2	108	1

For GRNN network, there were 108 neurons in hidden layer, two neurons in input layer and one neuron in output layer.

4.5 Experimental Results

To measure the goodness of fit of the model, we use the coefficient of multiple determination (R-square), the coefficient of correlation(r), r-square, mean squared error, mean absolute error, minimum absolute error and maximum absolute error.

Table 4.5. Experimental results

	Ward Net	GRNN
R-square	0.7403	0.8284
r	0.8717	0.9167
r-square	0.7598	0.8404
Mean squared error	1.7480	1.1552
Mean absolute error	0.7370	0.5557
Min absolute error	0.0060	0.0000
Max absolute error	5.7160	6.3380
t value	18.3105	13.7638
P value	<0.0001	<0.0001

Table 4.5 shows the values that are obtained from the two neural network models used for our experiments. The correlation of the predicted number of faults and the observed number of faults is represented by the coefficient of correlation (r). The r value of 0.871658 in Ward neural network and 0.915713 in GRNN network represents high correlations between the actual versus predicted output. The number of observations is 108. The significance level is indicated by p value. A commonly accepted p value is 0.05. In our experiment, a two tailed probability p values is less than 0.0001. This shows a high degree of confidence for the successful validations. The results clearly indicate close relationship between metrics NSC, NIUO, NDO, NT, NSCC and NJQ (independent variables) and predictable faults in software applications (dependent variable).

CHAPTER 4

4.6 Results from a 10-cross-validation study

We performed a 10-cross-validation of the prediction neural network model in Table 4.6 and Table 4.7. As presented in Chapter 3, it is a common technique to evaluate learning algorithms on a dataset. The 108 data points were randomly split into 10 partitions or folds of roughly equal size (eight partitions of 11 data points each, two partitions of ten data points each). For each iteration, the selected fold is become the test set, and the other 9 folds are combined into the training set. Create a neural network and train it on the training set and test it on the test set. We performed 10 times and took a mean accuracy to obtain the estimated performance of the neural network. The results are summarized in Table 4.6 for the GRNN network and in Table 4.7 for the Ward neural network respectively.

Table 4.6. Results from 10-cross validation for GRNN network

Partition	R-square	r squared	Correlation coefficient r
1	0.8241	0.8293	0.9107
2	0.7056	0.7662	0.8753
3	0.9292	0.9299	0.9643
4	0.7475	0.7675	0.8761
5	0.9064	0.9702	0.9525
6	0.9055	0.9086	0.9532
7	0.722	0.7722	0.8787
8	0.8871	0.8913	0.9441
9	0.5167	0.7634	0.8737
10	0.7893	0.8014	0.8952
Average	0.7933	0.8337	0.9124

CHAPTER 4

Table 4.7 Results from 10-cross validation for Ward Network

Partition	R-square	r squared	Correlation coefficient r
1	0.7853	0.7955	0.8919
2	0.7589	0.7888	0.8881
3	0.7804	0.7896	0.8886
4	0.7861	0.7943	0.8912
5	0.7639	0.7937	0.8909
6	0.7769	0.7986	0.8937
7	0.7769	0.7986	0.8937
8	0.5121	0.6810	0.8252
9	0.6333	0.6895	0.8304
10	0.7828	0.7946	0.8914
Average	0.7356	0.7724	0.8785

The average value of the cotrelation between predicted faults and actual faults is 0.9124 in the GRNN network and 0.8785 in the Ward network. They are statistically significant. The average squared multiple correlation is 0.79334 in the GRNN network and 0.73566 in the Ward network. Therefore about 79% and 74% of the variance, respectively, in the number of faults can be accounted for by the two predictors.

4.7 Conclusion

We studied the relationship between fault occurrence for database applications and SQL metrics. First we proposed SQL metrics that have strong relationship with Faults and then performed empirical validation for these metrics. We analyzed the fault reports kept by project teams of developing database applications using PLISQL code and found that faults are related to the number of SQL statements and the complexity of SQL statements. The relationship between the fault occurrence for database applications and SQL metrics has been empirically validated in this study. From the results prcsnted above, our proposed SQL metrics in this study appear to be useful in predicting faults in PL/SQLtiles.

We have compared this approach to a multiple regression analysis approach. By comparing empirical resultsincluding R-square and mean squared error, neural

CHAPTER 4

networks give better prediction results than Regression analysis. Regression analysis was carried out using SPSS software and regression analysis result is shown in Appendix D.

The results suggest the actual data has better conformity with GRNN prediction than with Ward. Ward neural network is a backpropagation network with different activation functions. Although backpropagation network is the most popular network for practical applications and a very powerful tool, it has sensitivities of backpropagation (e.g. selection of random number seed to select weight). GRNN has no training parameters such as learning rate and momentum in backpropagation network, but there is a smoothing factor that is applied after the network is trained. Smoothing factor is the only adjustable parameter in GRNN, making overtraining less likely in GRNN. The smoothing factor allows the GRNN to interpolate between patterns or spectra in the training set. The smoothing factor determines how tightly the network matches its predictions to the data in the training patterns. So, GRNN gives better prediction especially for production set.

These findings paved the way for future research into using neural network for predicting software maintainability. In addition, our research results also provide a new avenue for software project manager to determine the readiness of software under development.

Chapter 5

5. Estimating Software Readiness using Predictive Models

5.1 Chapter Preface

Defect tracking as a proxy method is used to predict software readiness in this study. Remaining defects are one of the most important factors that allow one to decide if a piece of software is ready to be released. By comparing the predicted number of faults and number of faults discovered in testing, software managers can decide whether the software are ready to be released or not. Our predictive models can predict: (i) the number of faults (defects), (ii) the number of lines required to be repaired to correct these defects and (iii) the amount of time (in minutes) to make the changes in respective classes using product metrics as independent variables. These metrics are selected depending on the nature of source code with regards to architecture layers, types of faults and contribution factors of these metrics.

The use of neural network model with a genetic training strategy is introduced to improve prediction results for estimating software readiness in this study. This genetic net combines a genetic algorithm with a statistical estimator to produce a model which also shows the usefulness of inputs. With many variables, it is quite difficult to find the exact combination of contributions of each of the variables that creates the best predictions [83]. The genetic algorithm provides the answer [44]. The genetic algorithm finds the best set of importance of input values on an arbitrary scale of 0 to 1. The importance of input values are a relative measure of how significant each of the inputs is in the predictive model. Since the sum over all inputs is approximately 1, these importance values may be thought of as the percent contribution to the model of the respective variables [83]. We can then decide to remove the variables with the lowest Relative Importance of Inputs value.

In times of global competition the companies require global information system through the internet. As internet applications are much more complex than conventional information system new architectures have been developed. The N-tier application architecture supports a flexible and reusable application for distribution to any number of client interfaces. Additionally, new development methods (component-

CHAPTER 5

based system design), new middle ware (CORBA, Java RMI, message-oriented middleware), and new database technology (object-relational database systems) have become applicable in the enterprise main-stream [104]. Nowadays, software houses widely used N-tier architecture approach and successfully implemented software systems by current technologies such as .NET and J2EE. Although they are widely used in nowadays, none of the prediction of number of software development faults (defects) studies is based on architecture layers yet. This research introduces predictive models that are based on N-tier software development architecture exploiting the new technologies

This system is aimed to be useable in N-tier applications. N-tier application architecture contains any number of levels arranged above another, each serving distinct and separate tasks as shown in Figure 5.1. Conceptually, each tier is treated as a black box with well-defined interfaces [81]. For example, re-definition of the storage strategy or change to the new DBMS won't influence the other tiers. Since all tiers have clear separation, this allows the developers to plug each layer in and out without too many hassles and without limiting the technology used at each tier [25]. In the past, two-tier applications that are also known as client/server applications were commonplace. In some cases, the only service provided by the server was that of a database server. In those situations, the client was then responsible for data access, applying business logic, converting the results into a format suitable for display, displaying the intended interface to the user, and accepting user input. The client/server architecture is generally easy to deploy at first, but is difficult to upgrade or enhance, and is usually based on proprietary protocols, typically proprietary database protocols. It also makes reuse of business and presentation logic difficult. Finally, and perhaps most importantly in the era of the Web, two-tier applications typically do not prove very scalable and are therefore not well suited to the Internet [25].

Figure 5.1. A Typical N-Tier Model

Our research team has analyzed the fault reports of the Warehouse Application System (WAS) and the Hwafuh Information System (HIS) and classified the faults. These software fault reports are recorded by CA1BGmbH, Murrhardt Company and Hwafuh, Myanmar software development team during system test and maintenance stages in recent years for the Warehouse Application System and the Hwafuh Information System respectively. Therefore, our prediction model is divided into three parts: (1) prediction model for Presentation Logic Tier (2) prediction model for Business Tier ants: (1) prediction model for Data Access Tier because each layer serves distinct and separate tasks in N-tier application architecture.

The classification scheme was based on the nature of these faults and their relevance to the application architecture. For example, the faults (defects) coming from the Business Tier are strongly related to OO features and are introduced by features such as inheritance and polymorphism. The Data Access Tier is mainly used to interface with database. Therefore, the faults (dependent variable) occurs in the

CHAPTER 5

Data Access Tier are strongly related to the codes that are used to interface with the database. The Data Access Tier allows the creation of a set of useful and specific routines to be able to perform insert, select, delete and update actions on database tables. It does not usually contain business logic or presentation items. So, predictive models for the Data Access Tier differ from other tier models depending on independent variable (SQL complexity metrics). Existing object-oriented metrics and complexity software metrics are used in the Business Tier prediction model. New sets metrics have been proposed for the Presentation Logic Tier and the Data Access Tier. These metrics are validated using real world application data which are collected from the Warehouse Management System (WMS) and the Hwafuh Infomation System (HIS). From the experiments results, proposed metrics in this study appear to be useful in predicting software readiness. These software metrics are significantly related to the number of faults and the maintenance effort which are important factors for deciding software readiness. As such, our models can he used as one of the tools to assist software mangers in deciding whether the software is ready to be released or not.

Most of the software quality models e.g.[14, 15, 23, 28, 34, 36, 39, 42 etc.] for object-oriented systems predict whether a software module contains fault(s) by using traditional statistical models. Very few object-oriented software quality models predict the number of faults and maintainability (number of lines). For example, the experiment done by Wei Li and S. Henry [64] predict maintainability in terms of number of line and the experiment done by Ping Yu et al. [118]predict the number of faults in programs using object-oriented software metrics as independent variables. The dependent variables such as number of faults, number of lines and required time (in minutes) to correct these errors are correlated each other. As shown in Appendix F, they have significant positive Pearson's correlation coefficients. It is clearly seen that the more faults (defects) contain in the class, the more maintenance cost requires. Therefore, our models predict all three correlated variables to support software managers to decide software readiness.

The remainder of this chapter is organized as follows: the metrics used in our predictive models as independent variables are described in Sections 5.2, 5.3 and 5.4. The empirical studies for the Data Access Tier, the Business Tier and the presentation

Logic Tier as well as their results and discussion are presented in Section 5.5. Our conclusions, from this empirical study, and future research directions are given in Section 5.6.

5.2 Software metrics for the Data Access Tier

The Data Access Tier focuses on storing and retrieving data from a data source, typically a relational database management system. The Data Access Tier contains generic methods to interface with data. For example, these codes contain a method for creating and opening a Connection object (internal), and another for creating and using a Command object, along with a stored procedure (with or without a return value), etc. The Data Access Tier is mainly used to interface with database. In other words, it allows the creation of a set of useful and specific routines to be able to perform insert, select, delete and update actions on database tables. It does not usually contain business logic or presentation items. This tier contains a set of specific methods that are used to perform all of the database interactions. Therefore, the metrics for the Data Access Tier are different from other tiers. SQL commands can be issued interactively or embedded in a host language. For example: a method of a class may pass a select command as a parameter to a DataSet object. In data-intensive applications, it is quite common for the implementation code to dynamically construct database query strings and execute them. For example, a typical Java servlet web service constructs SQL query strings and dispatches them over a JDBC connector to an SQL-compliant database [46].

SQL commands are mainly composed in the Data Access Tier classes to perform tasks such as update data on a database, or retrieve data from a database. The complexity of SQL commands is strongly related to the faults of Data Access Tier. The following Data Access Tier metrics are proposed and used in this study. Metrics having weak relationship with fault occurrence, such as the number of Data Definition Language (DDL) commands and Data Control Language (DCL) commands, are omitted in this study.

- **Total Number of Select Commands (TNSC)**

CHAPTER5

It measures the total number of select commands used in a class or a module. Select command is an executable command that can be issued interactively or embedded in a host language. TNSC is given by

$$\text{TNSC}(c) = |\text{Select}(c)|$$

where a class c is a Data Access Tier class and $\text{Select}(c)$ is the set of select statements in class c .

- **Total Number of Insert/Update Operations (TNIUO)**

It measures the total number of invocation of insert/update operations in a class or module. TNIUO is given by

$$\text{TNIUO}(c) = |\text{Update}(c)| + |\text{Insert}(c)|$$

where $\text{Update}(c)$ is the set of update statements in class c and $\text{Insert}(c)$ is the set of insert statements in class c .

- **Total Number of Delete Operations (TNDO)**

It measures the total number of invocation of delete operations in a class or module. TNDO is given by

$$\text{TNDO}(c) = |\text{Delete}(c)|$$

where $\text{Delete}(c)$ is the set of delete statements in class c .

- **Average Number of Search Condition Criteria of where-clause (ANSC)**

Most relational database queries retrieve only a portion of the records contained in a table. The where-clause qualifies the query command statement to limit the data retrieved to specific rows in the table. It is generally used with a select statement to specify search criteria for retrieving rows of data from a table or group of tables. The where-clause can also appear in delete and update command statements. A predicate states selection criteria, which is applied to each row of data values in the

CHAPTER 5

table being queried by the select statement. The higher number of selection criteria may cause higher complexity of SQL statement. ANSC is given by

$$ANSC(c) = \frac{\sum Criteria(Select \cup Update \cup Delete(c))}{|Select(c)| + |Update(c)| + |Delete(c)|}$$

where $Select(c)$ is the set of select statements in class c , $Update(c)$ is the set of update statements in class c , $Delete(c)$ is the set of delete statements in class c and $Criteria(Select \cup Update \cup Delete(c))$ is the number criteria in select statements or update statements or delete statements in class c .

- **Total Number of Sub-Queries (TNSQ)**

A sub-query consists of two or more ordinary queries nested in such a way that the results of each inner query are used in the comparison test for the selection clause of the next outer query (or another command statement). This metric is very important in measuring the complexity of SQL statements. TNSQ is given by

$$TNSQ(c) = |Subquery(c)|$$

where $Subquery(c)$ is the set of sub-query statements in class c .

- **Total Number of Group-By-clause (TNGB)**

When a query statement includes a group-by clause, the select-clause for that query may list one or more aggregate functions (SUM, COUNT, AVG, etc.) operating on groups of data values in other columns. The group-by clause contains a column-list argument that must include all simple column names that appear in the select-clause. Simple column names are called grouping columns, since they group together records with identical data in that column. This record grouping allows mathematical operations to be performed on the other columns specified as arguments in the aggregate functions. Records can also be grouped by any valid expression appearing in the select-clause argument. In these cases, the expression must also be referenced in the group-by-clause by a whole number representing its relative column position in the select-clause (or output result). This metric counts the total number of select

CHAPTER 5

statements containing group-by-clause invoked from a particular application class. TNGB is given by

$$TNGB(c) = |Groupby(c)|$$

where Groupby(c) is the set of select statements containing group by clause in class c.

5.3 Software metrics for the Business Tier

The Business Tier is the brain of applications and basically contains things like the business rules, data manipulation, etc. Some of the functionalities associated with this tier are transaction processing, user authentication, data validation and application logging. It controls the logic and flow of the application as shown in Figure 5.1. This layer does not have codes to access the database or codes for the user interface. As this layer mainly contains object-oriented codes, object-oriented metrics are used for this tier. There is great interest in the use of the object-oriented approach to software engineering. Many measures have been proposed and evaluated in the literature to capture the structural quality of object-oriented (OO) code and design [15, 19, 23, 28, 34, 36, 39, 42 etc]. The following existing traditional complexity metrics and object-oriented metrics are used in the prediction model for the Business Tier.

- **Lack of Cohesion in Methods (LCOM)**

Cohesion is an important attribute corresponding to the quality of the abstraction captured by the call under consideration. Good abstractions typically exhibit high cohesion. Classes with low cohesion often result in unnecessary complexities. Lack of Cohesion in Methods is the 100% minus the average percentage of the number of class methods accessing each class instance variable. A lower percentage means higher cohesion between class data and methods.

- **Number of Parents (NOP)**

Inheritance is a mechanism whereby one object acquires characteristics from one, or more, other objects. Some object-oriented languages support multiple

CHAPTER 5

inheritances, i.e. an object may acquire characteristics directly from two, or more, different objects. The number of parents metric measures the number of classes that a given class directly inherits from.

- **Number of Children (NOC)**

This measures the number of immediate descendants of a particular class. This measures an amount of potential reuse of the class. The more reuse a class might have, the more complex it may be, and the more classes are directly affected by changes in its implementation.

- **Depth of Inheritance Tree (DIT)**

The DIT of a class is the length of the longest path from the class to the root in the inheritance hierarchy. This determines the complexity of a class based on its ancestors, since a class with many ancestors is likely to inherit much of the complexity of its ancestors. The deeper a class is in the hierarchy, the greater the number of methods it is likely to inherit making it more a complex task to predict its behavior.

- **Coupling Between Objects (CBO)**

Coupling refers to the degree of inter-dependence among the components of a software system. Good software design should obey the principle of low coupling. Strong coupling makes a system more complex; highly interrelated modules are harder to understand, change or correct. By minimizing coupling, one can avoid propagating errors across modules. CBO is defined as the number of other classes to which it is coupled.

- **Response For a Class (RFC)**

RFC is the number of methods that can potentially be executed in response to a message received by an object of that class. The response set of a class consists of the set of M methods of the class, and the set of methods directly or indirectly invoked by methods in M .

CHAPTER 5

- **Number of Methods Added (NMA)**

The NMA is the number of new methods in a class, neither inherited, nor overriding.

- **Number of Attributes (NA)**

The Number of Attributes (NA) is the number of attributes in a class (excluding inherited ones).

- **Weighted Methods per Class (WMC)**

The Weighted Methods per Class (WMC) is defined as the number of all member functions and operators in each class.

- **Average Method Complexity (AMC)**

The Average Method Complexity (AMC) provides the average McCabe's cyclomatic complexity of methods in a class. The more control flows a class's methods have, the harder it is to understand them, and thus, to maintain them. A method with a low cyclomatic complexity is generally better. This may imply decreased testing and increased comprehensibility or that those decisions are deferred through message passing, not that the method is not complex.

5.4 Software metrics for the Presentation Logic Tier

This layer provides an interface for the end user with the application. It works with the results/output of the Business Tier to handle the transformation into something usable and readable by the end user. It consists of windows forms, dialogs and ASP documents etc. The following two software metrics are proposed for Presentation Logic Tier.

- **Number of User Interface Objects (NUIO)**

The NUIO is the measure of user interface objects or controls which handle input operations from user and output operation to users. For example, textbox objects, button objects and label objects etc. The NUIO is given by

$$NUIO(c) = |InterfaceObjects(c)|$$

where $InterfaceObjects(c)$ is the set of interface objects which are declared in the Presentation Logic Tier class c .

- **Total Number of Messages (TNM)**

This metric measures the total number of interface objects' messages of presentation layer classes. For example, BN_CLICKED message, LBN_DBCLICKED message of CListBox and CButton interfaceobjects. The TNM is given by

$$TNM(c) = |Messages(c)|$$

where $Message(c)$ is the set of messages which are handled by interface objects of the Presentation Logic Tier class c .

5.5 Empirical study

The genetic training strategy of NeuroShellPredictor is used in this study. The genetic net combines a genetic algorithm with a statistical estimator to produce a model which also shows the usefulness of inputs. Genetic algorithms (GAS) seek to solve optimization problems using the methods of evolution, specifically survival of the fittest. The functioning of the genetic estimator is based upon the General Regression Neural Net (GRNN) [102]. Genetic learning stores every set of inputs and related output in the training data. When the net is applied to a new set of inputs, the new inputs are compared to every set of stored inputs. Depending upon how close the match is, the output for each training row is weighted. Closer matches receive higher weights, and inputs that are farther away from the training inputs receive lower weights. The predicted output for the new set of inputs is a "weighted" average of all of the outputs with which the network was trained.

Because of the methodology used in the genetic method of training, out of sample evaluation sets are not necessary. The genetic method uses a "one hold out" strategy both during training and afterwards when evaluating new data. If the method is called upon to produce an output from a particular pattern of inputs X , then it never

CHAPTER 5

looks at X if X is in the training set, as long as enhanced generalization is turned on. This is true at all times during training. The genetic method is much like a "nearest neighbor" predictor or classifier. Its output is a kind of weighted average of the close neighbors' outputs in the training set. So another way of saying this is that if evaluating X , X is never considered to be in the neighborhood. Most other neural nets do not work this way. They look at the entire training set when training everything in the training set. Therefore, they are essentially looking at the entire training set when applied to other data later [26, 84].

5.5.1 Data collection

In our study, we use the term 'defect' or 'fault' which is detected during system testing. By comparing the predicted number of faults and number of faults discovered in testing, software managers can decide whether the software or software components are ready to be released or not. The experiment data is collected from two applications systems. The first application is the warehouse management applications (WMA) that is developed using C, JAM (JYACC Application Manager) and PL/SQL languages. This set of applications has more than a thousand source files of C, JAM (JYACC Application Manager) and PL/SQL codes and uses the Oracle database. The warehouse system has been customized and used by many companies. Data Access Tier defects were collected from the journal files that contain the documentation of all changes in source files such as status of module, start date, end date, developer, nature of changes, etc. Data on software metrics were extracted from 103 PL/SQL files of the above mentioned warehouse application using the metric extraction tool that was developed using VC++ incorporating MKS LEX & YACC utilities as embedded languages. The descriptive statistics for WMS is shown in Table 5.1.

CHAPTER 5

Table 5.1. Descriptive statistics for the Data Access Tier of the WMS system data set

	Min	Max	Mean	Median	Std dev
TNSC	0	57.00	7.2621	3.00	10.6991
NIUO	0	17.00	1.9029	1.00	2.5264
NDO	0	5.00	0.3398	0.00	0.8581
ANSC	0	7.00	2.0536	1.75	1.3077
TNSQ	0	5.00	0.1262	0.00	0.5888
TNGB	0	8.00	0.1553	0.00	0.9976
DEFECTS	0	5.00	0.3204	0.00	0.8542

The second applications used in this prediction are subsystems of an application system which is used in the Hwafuh(Myanmar) company, which is a fully networked informationsystem. The Hwafuh Information System (HIS) contains more than 300 classes and approximately ,1,000,000 lines of codes. The experiment data was collected from the payroll subsystem, time record subsystem, human resource function subsystem and piccc calculation subsystems. All the subsystems are implemented using Microsoft Visual C++. The HIS system software development team recorded maintenance records for test case failures in change request forms. A test case failure may contain one or more defects and need to change/correct within a single class or among classes. Each defect is fixed when it is discovered. The change request form contains test failure description, class names, defect description, number of line changes, dale and time of start and finish, the amount of time(in minutes) lo correct these defects. The descriptive statistics for HIS is shown in Table 5.2, 5.3 and 5.4.

Table 5.2. Descriptive statistics For Presentation Logic tier of HIS system data set

	Min	Max	Mcan	Median	Std dev
TNSC	3	24.00	8.0517	6	4.2444
TNIUO	0	18.00	4.9655	5	4.3122
DEFECTS	0	10.00	1.4483	1	2.1041
LINES	0	152.00	10.6724	2	24.2186
MINUTES	0	280.00	31.0172	6	56.8819

CHAPTER5

Table 5.3. Descriptive statistics for the Data Access Tier of HIS system data set

	Min	Max	Mean	Median	Std dev
TNSC	1	15.00	3.1892	2	2.9799
TNIUO	0	4.00	1.7027	2	1.1271
TNDO	0	1.00	.16220	0	0.3737
ANSC	0	1.80	.75970	1	.54294
DEFECTS	0	14.00	2.1622	1	2.9013
LINES	0	198.00	28.6216	4	51.0475
MINUTES	0	243.00	47.3243	25	66.9224

Table 5.4. Descriptive statistics for the Business tier of HIS system data set

	Min	Max	Mean	Median	Std dev
LCOM	0	100.00	57.3064	65.0	31.4579
DIT	0	2.00	0.8613	1.0	0.4083
NOP	0	1.00	0.8382	1.0	0.3694
CBO	0	10.00	1.7630	1.0	2.2142
NOC	0	6.00	0.0983	0.0	0.5671
RFC	2	33.00	9.0462	7.0	5.7383
NMA	2	33.00	8.2486	6.0	5.4561
NA	0	24.00	4.4104	3.0	4.5366
WMC	2	33.00	8.1965	6.0	5.4673
AMC	0	8.00	1.7167	1.0	1.2083
DEFECTS	0	11.00	1.3988	1.0	2.0735
LINES	0	152.00	9.9942	2.0	20.7105
MINUTES	0	280.00	27.7457	5.0	49.6990

5.5.2 Experiment for the Data Access Tier

5.5.2.1 Experiment 1

The warehouse application system data was used for the prediction of data access faults (dcfcts) of Data Access Tier. Only the number of defects prediction has done in Experiment 1 because maintenance cost (in lines and minutes) data is not available for WMS system. Six software metrics were extracted from 103 PL/SQL files. It contains TNSC, TNDO, TNIUO, TNGB, TNSQ and ANSC metrics. The dependent variable was the number of Data Access Tier Faults and the independent variables were the six software metrics identified above. First, each data pattern was

CHAPTER 5

examined for erroneous entries, outliers, blank entries and redundancy. A threshold value was set 1000 for maximum number of generations without improvement. After 1621 generations, the optimized coefficient of multiple determination (K-square) value 0.737046 was arrived at. Therefore, about 74% of the variation in the number of faults can be accounted by the six predictors.

To measure the goodness of fitness of the model, coefficient of multiple determination (R-square), the coefficient of correlation (r), mean squared error (MSE) and root mean squared error (RMSE) were used. These statistical measures are shown in Table 5.5. The correlation of the predicted change and the observed change is represented by the coefficient of correlation (r). An r value of 0.860836 represents high correlations between the actual versus predicted output. The number of observations is 103. In this experiment, a two tailed probability p value is less than 0.0001. This shows a high degree of confidence for the successful validations. The results clearly indicate a close relationship between Data Access Tier metrics (independent variables) and the number of Data Access Tier faults/defects (dependent variable).

Table 5.5. Experimental result for the WMA system

R-square	0.737046
r (correlation coefficient)	0.860836
Avg error	0.266629
MSE	0.190009
RMSE	0.435900
t values	17.000590
p values	<0.0001

Table 5.6 shows the relative importance of inputs Data Access Tier metrics. It displays a list of input metrics and a corresponding number which indicates the importance of the variable in predicting the output. The Relative Importance of Inputs values signify the importance of the variables. The weight values range from 0 to 1. Weights near 0 signify the least important variables, while weights near 1 signify the most important. The genetic method gives more reliable importance factors than the neural method does. The Genetic Training Strategy uses a "genetic algorithm" or survival of the fittest technique to determine a weighting scheme for the

CHAPTER 5

inputs. The genetic algorithm tests many weighting schemes until it finds the one that gives the best predictions for the training data

From the above experiment, conclusion can be made the total number of sub-queries is the most important factor to increase the complexity of SQL statements

Table 5.6. Relative importance of Data Access Tier metrics for the WMA system

Metrics	Relative Importance
TNSQ	0.369
ANSC	0.185
TNGB	0.159
TNIUO	0.155
TNSC	0.129
TNDO	0.004

5.5.2.2 Experiment 2

The Hwafuh Information System (HIS) was used for prediction of data access faults, the number of line changed per class and required time (in minutes) to correct these data access tier faults. Six software metrics were extracted from 37 Data Access Tier classes.

- Selection of Metrics

Four Data Access Tier metrics (TNSC, TNDO, TNIUO and ANSC) form proposed six Data Access Tier metrics are selected in this experiment. TNGB and TNSQ metrics could not be collected because corresponding SQL constructs are seldom used in the HIS system.

- Prediction of number of defects

The relative importance of Data Access Tier metrics prediction of the number of faults is shown in Table 5.7. Average number of search condition criteria of where-clause (ANSC) and Number of insert /update operations (TNIUO) are found to be more important for number of faults prediction for the Data Access Tier. That shows the complexity of search condition criteria is highly related to occurrence of faults in this Tier. From database operations statements, insert/update operations is more

CHAPTER 5

important than other operations (delete and select). TNSC metric is the least important metric for the number of faults prediction.

- **Prediction of number of lines changed per class**

In this prediction, the TNSC metric is the most important inputs in this model as shown in Table 5.7. The amount of Select operation containing in Data Access Tier class is highly related to number of lines changes in that class. The ANSC and the TNIUO metrics are also important for prediction of number of lines changed. The metrics regarding with delete operations is the least important for this prediction.

- **Prediction of required time (in minutes) to change**

The ANSC and the TNIUO metrics are also most important for prediction of required time (in minutes) to change. In this experiment shows that insert/update database operations are more important for prediction of maintenance effort than other database operations: The complexity of selection criteria is also important factor in this model.

Although only 37 rows of data have been tested in Experiment 2 the two-tailed P values are less than 0.0001. By conventional criteria, these p values are considered to be statistically significant.

As shown in Table 5.8, r values are 0.788908 for prediction of the number of faults, 0.792298 for prediction of the number of lines changed per class and 0.818494 for prediction of required time (in minutes) to change have received. These results show a close relationship between Data Access Tier metrics (independent variables) and dependent variables (the number of faults per class and the maintenance cost). The values of coefficient of multiple determination are 0.733111 for prediction of the number of faults, 0.624663 for the prediction of the number of lines changed per class and 0.664182 for prediction required time (in minutes) to change. Therefore about 73%, 62% and 66% of the variance, respectively, in the number of faults and the maintenance cost can be accounted for by these predictors.

CHAPTEK 5

Table 5.7. Relative importance of Data Access Tier metrics

Prediction (number of)	Relative Importance of Data Access Tier Metrics			
	ANSC	TNIUO	TNDO	TNSC
Defects	0.490	0.483	0.026	0.001
Lines	0.217	0.280	0.081	0.422
Minutes	0.451	0.462	0.002	0.085

Table 5.8 Experiments 2 results for the Data Access Tier

Prediction (number of)	Prediction Results from the Data Access Tier						
	R-square	r	Avg error	MSE	RMSE	t values	p values
Defects	0.733111	0.788908	1.113268	2.185798	1.478445	7.595036	<0.0001
Lines	0.624663	0.792298	17.79677	951.6382	30.84863	7.682408	<0.0001
Minutes	0.664182	0.818494	26.30481	1463.352	38.25378	8.428446	<0.0001

5.5.3 Experiment for Business Tier

The Hwafuh Information System (HIS) was used for prediction of data access Faults, the number of line changed per class and required time (in minutes) to correct these Business Tier faults. Software metrics were extracted from 173 Business Tier classes.

Selection of Metrics

As discussed in Section 5.3, we have selected the Chidamber and Kmcrcr object oriented metrics [28] and complexity metrics. These existing metrics are already validated in various studies [14, 15, 23, 19, 28, 34, 36, 39, 42 etc]. After testing several combinations of models using Business Tier software metrics using subsystems of the Hwafuh Information system data, the following models are best for prediction of number of faults, the number of line changes and required time (in minutes) respectively.

Prediction of number of defects

This model contains NOP, RFC, DIT, NMA, NOC, NA, CBO and WMC metrics and their relative importance for prediction of the number of faults is shown in Table 5.9. From Business Tier metrics, inheritance metrics (NOP, DIT), RFC, coupling metrics (CBO), NA and NMA are mainly important for number of faults prediction in this tier.

Prediction of number of lines changed per class

This model contains NA, RFC, AMC, DIT, CBO, NMA, NOP, WMC and NOC metrics and their relative importance for prediction of the number of lines changed per class is shown in Table 5.9. From Business Tier metrics, NA, RFC, complexity metrics (AMC) and inheritance metrics (DIT) are mainly important for number of lines changed per class prediction in this tier.

Prediction model for required time (in minutes)

This model contains DIT, LCOM, NOP, RFC, NA, CBO, NMA and AMC metrics, and their relative importance for prediction of the number of required time (in minutes) to change is shown in Table 5.9. As in fault prediction experiment, NOP, DIT and RFC are mainly important. To oppose to fault prediction, LCOM is relative important for this prediction. NMA metrics is the least important for prediction of required time (in minutes) to change.

For 173 observations, 2 the two-tailed P values are less than 0.0001. By conventional criteria, these p values are considered to be statistically significant. As shown in Table 5.10, r values are 0.8345 for prediction of the number of faults, 0.8169 for prediction of the number of lines changed per class and 0.7681 for prediction of required time (in minutes) to change have received. These results show a close relationship between Business Tier metrics (independent variables) and dependent variables (the number of faults per class and the maintenance cost). The values of coefficient of multiple determination are 0.6921 for prediction of the number of faults, 0.6623 for the prediction of the number of lines changed per class and 0.5895 for prediction required time (in minutes) to change. Therefore about 70 %,

CHAPTER 5

66% and 59% of the variance, respectively, in the number of faults and the maintenance cost can be accounted for by these predictors.

Table 5.9. Relative importance of Business Tier metrics

Prediction (number of)	Relative Importance of Business Tier Metrics									
	NOP	RFC	DIT	LCOM	NA	CBO	NMA	NOC	AMC	WMC
Defects	0.178	0.198	0.162	-	0.177	0.084	0.102	0.037	-	0.063
Lines	0.010	0.234	0.115	-	0.432	0.063	0.015	0.002	0.124	0.005
Minutes	0.127	0.240	0.219	0.137	0.122	0.110	0.002	-	0.045	-

Table 5.10. Experiments results for the Business Tier

Prediction (number of)	Prediction Results from the Business Tier						
	R-square	r	Avg error	MSE	RMSE	t values	p values
Defects	0.6921	0.8345	0.8257	1.3160	1.1472	19.8079	<0.0001
Lines	0.6623	0.8169	7.2002	143.9963	11.9998	18.5215	<0.0001
minutes	0.5895	0.7681	20.2564	1008.1710	31.7517	15.6843	<0.0001

5.5.4 Experiment for Presentation Logic Tier

Presentation tier classes (58 classes) were collected from the Hwafuh Information System.

Selection of Metrics

As they mainly include dialog classes, form classes and view classes etc, the number of user interface objects (NUIO) metrics and the total number of messages (TNM) metrics were proposed and used in this study. Table 5.11 displays the importance of the input Presentation Tier metrics in predicting the number of faults, number of line change and required time (in minutes). Their relative importance is roughly equal for experiments of Presentation Logic Tier.

CHAPTER 5

Prediction of number of faults

After performing 380 generalizations, an R-square value of 0.6740655 was received using the genetic learning strategy of NeuroShellpredictor as shown in Table 5.12. Therefore about 67% in the number of faults can be explained by two predictors. An r value of 0.828182 represents high correlations between the actual versus the predicted number of faults for 58 observations.

Prediction of number of line changed per class

After performing 156 generalizations, an R-square value of 0.53181 was received. Therefore about 53% in the number of lines changed per class can be explained by two predictors as shown in Table 5.12. An r value of 0.740294 represents high correlations between the actual versus the predicted number of lines for 58 observations. Table 5.11 displays the importance of the input presentation layer metrics in predicting the number of lines changed per class.

Prediction model for required time (in minutes)

After performing 116 generalizations, an R-square value of 0.598468 was received. Therefore about 60% in required time (in minutes) to change per class can be explained by two predictors as shown in Table 5.12. An r value of 0.799905 represents high correlations between the actual versus the predicted the number of minutes for 58 observations. Table 5.11 displays the importance of the input Presentation Tier metrics in predicting required time(in minutes) to change per class.

From the results presented above, proposed Presentation Logic Tier metrics in this study appear to be useful in predicting software readiness. In Presentation Logic Tier experiment, high r values show a close relationship between Presentation Logic Tier metrics (independent variables) and dependent variables (the number of faults per class and the maintenance cost). About 67 %, 53% and 60% of the variance, respectively, in the number of faults and the maintenance cost can be accounted for by these predictors. The relative importance both Presentation Logic Tier metrics are nearly the same

CHAPTER 5

Table 5.1 1. Relative importance of Presentation Logic Tier metrics

Prediction (number of)	Relative Importance of Presentation Logic Tier Metrics	
	NUIO	TNM
Defects	0.505	0.495
Lines	0.538	0.462
Minutes	0.434	0.566

Table 5.12. Experiments results for the Presentation Logic Tier

Prediction (number of)	Prediction Results from the Presentation Logic Tier						
	R-square	r	Avg error	MSE	RMSE	t values	p values
Defects	0.674067	0.828182	0.94906	1.418063	1.190824	11.05799	<0.0001
Lines	0.53181	0.740294	7.765123	269.8776	16.42795	8.240345	<0.0001
Minutes	0.598468	0.799905	25.01111	1276.78	35.73206	9.974457	<0.0001

Table 5.13. Comparative performance data for the models

Model	Prediction	Tiers					
		Data Access		Business		Presentation	
		R-square	RMSE	R-square	RMSE	R-square	RMSE
Regression	Defects	0.274415	2.43772	0.3643	1.6484	0.514112	1.453955
	Lines	0.180226	45.59028	0.3587	16.5375	0.442837	17.92105
	Minutes	0.156954	60.61049	0.2208	43.7434	0.199984	50.43681
Genetic Net	Defects	0.733111	1.478445	0.6921	1.1472	0.674067	1.190824
	Lines	0.624663	30.84863	0.6623	11.9998	0.53181	16.42795
	Minutes	0.664182	38.25378	0.5895	31.7517	0.598468	35.73206

5.6 Conclusion and Discussion

The followings are the three main purposes of this research:

- to support software managers to decide software readiness by predicting the number of defects and required time (in minutes) and line to correct these defects
- to build predictive models based on software development architecture layers
- to introduce a genetic net approach for predictive models

The genetic training strategy of NeuroShell Predictor is used in this study. The Genetic Training Strategy uses a "genetic algorithm" or survival of the fittest technique to determine a weighting scheme for the inputs. The genetic algorithm tests many weighting schemes until it finds the one that gives the best predictions for the training data. Data patterns from WMS (103 patterns) were used for the prediction of the number of faults in the Data Access Tier prediction model. This study used 37 Data Access Tier classes, 173 Business Tier classes and 58 for Presentation Logic Tier classes from HIS to predict the number of Faults, the number of lines changed and time required (in minutes). For these numbers of patterns, the genetic training method is much better because the evaluation set is not necessary in the genetic training method. The genetic method uses a "one hold out" strategy both during training and afterwards when evaluating new data [26]. Moreover, the genetic method generally gives more reliable importance factors than the neural method does [26]. We have compared this approach to a multiple regression analysis approach. The comparative performance data for the models are shown in Table 5.13. By comparing empirical results including K-square and KMSE, Genetic Nets give better prediction results than Regression analysis. Regression analysis has done by using SPSS software and detail regression analysis results are shown in Appendix E.

As shown in Table 5.7, SQL complexity metrics (average number of search condition criteria of where-clause) is important for all three predictions for Data Access Tier. That shows the complexity of SQL is highly related to occurrence of faults and maintenance effort level in this Tier. From database manipulation operations statements, insert/update operations is more important than other operations (i.e. delete and select) in prediction of number of faults and required time

CHAPTER 5

(in minutes) to correct. As shown in Table 5.9, for business tier source codes, object-oriented metrics such as RFC, DIT and NA are important for all three predictions but the importance of traditional software metrics (the average McCabe's cyclomatic complexity metrics) and other OO metrics such as LCOM and NOC are relatively less significant. It was also round that the number of all member function metrics (WMC) will lose its importance if WMC counts all member function including simple function such as put and get methods.

As shown in Table 5.1 1, the relative importance both Presentation Logic Tier metrics are nearly the same. From the results presented above, software metrics corresponding to architecture layers chosen in this study appear to be useful in predicting software readiness. Although the results for the Presentation Logic Tier are not very good by comparing with the Data Access Tier, Presentation Logic Tier metrics can be stepping stone for future derivation of Presentation Tier Logic since they yield promising results in this experiment. We intend to extend this investigation with a wide range of application and proposing a new set of Presentation Logic Tier metrics regarding various types of user interface objects.

Chapter 6

6. An Implementation of Metrics Extraction and Analyzer Tool

6.1 Chapter Preface

In this study, the metrics extraction and analyzer tool have been implemented for Business Tier and Data Access Tier source codes. The primary aim of the implementation of the metrics extraction and analyzer tool is to facilitate generation of software metrics which are used as independent variables in software quality prediction models. A secondary goal is to gather and present information on the attributes of source code to assist the programmers in managing the software projects. This tool is able to generate the principal components, correlation analysis report and descriptive statistics report. These analysis reports are essential reports for the study of software measurement and software quality management.

Software metrics have been proposed as potential tools in attempts to improve the quality of computer software, and to help in the control of software projects. This research focus on those metrics that can be programmatically computed from software source code and on the context of using an automated tool for gathering of source code metrics.

The aim of our tool is to generate metrics for the Business Tier and the Data Access Tier. It can automatically identify the principal components from generated software metrics. Principal component analysis (PCA) is a standard technique to identify the underlying, orthogonal dimensions that explain relations between the variables in a data set. Principal components (PCs) are linear combinations of the standardized independent variables. Principal components analysis can be used to reduce the number of independent variables in analysis and to detect structure in the relationships between software metrics.

6.2 Generated Software Metrics

The generated software metrics are tabulated in Table 6.1 and 6.2. As described in Chapter 5, they are aimed to be applicable for main two tiers of N-tier applications: Business Tier and Data Access Tier.

Table 6.1. Data Access Tier metrics

metrics	Description
TNSC	Total number of select commands
TNIUO	Total number of insert/update operations
TNDO	Total number of delete operations
ANSC	Average number of search condition criteria of where-clause
TNSQ	Total number of sub-queries
TNGB	Total number of group-by-clause

Table 6.2. Business Tier metrics

metrics	Description
LCOM	Lack of Cohesion in Methods
NOP	Number of Parents
NOC	Number of Children
DIT	Depth of Inheritance Tree
CBO	Coupling between Objects
RFC	Response for a class
NMA	Number of methods added
NA	Number of Attributes
WMC	Weighted Methods per Class
AMC	Average Method Complexity

6.3 Architecture Description of Metric Extraction and Analyzer Tool

The architecture includes two main components, the metric extraction module and the principal component analysis module as shown in Figure 6.1

Figure 6.1. Architecture description

6.3.1 Metric extraction module

The function of the metric extraction module is to collect software metrics from the source codes that are written in VC++, C++ and Java. It was implemented using VC++ incorporating MKS LEX & YACC utilities as embedded languages.

It contains two phases. The first phase scans the source code and generates class information. Its class diagram is shown in Figure 6.2. It contains two parts: a lexical analyzer and syntax analyzer. A lexical analyzer or scanner uses patterns that match strings in the input and converts the strings to tokens. It consists of three sections: definition, translation and function. The definition section is for defining the names of the regular expressions. The translation section contains the regular expressions that lcx attempts to match. It then returns the token associated with that expression. The input to the lexical analyzer is a java or VC++ or C++ source code. The lexical analyzer returns the token as a string of characters.

A syntax analyzer or parser uses grammar rules that allow it to analyze tokens and extract class information from the source code. The parser consists of three sections: declarations, grammar rules and functions. The declaration section describes the identifiers for the language which needs to be parsed. It also includes the data type of each identifier. The data types of the grammar rules in the declaration portion can

CHAPTER 6

also be specified. The grammar rules section defines legal statements in the input file. We use Java, C++ and VC++ grammar rules for the syntax analyzer. The parser module also stores class information of the input source code in the database while parsing. The class information stored is later used for generating software metrics.

Figure 6.2. Class diagram for class information generation sub system

The second phase reads the class information from the database and calculates the metrics. The Data model for class information is shown in Figure 6.3. Output is reported in the HTML format and the Microsoft Excel format as shown in Figure 6.4. This system also produces detail report for class information in HTML format.

CHAPTER 6

Figure 6.3. Data model for class information generation sub system

CHAPTER 6

Class Information - Microsoft Internet Explorer

Address: E:\output\myMetrics.html

Object-Oriented Metrics

ClassID	Class Name	DIT	AID	CLD	NOP	NOP (AID)	NOA	NOD	NOC	NOM	NA	DAC
1	Breeder	2	2.00	0	1	1	2	0	0	1	0	0
2	BreedingPipeline	2	2.00	0	1	2	2	0	0	12	6	1
3	BreedingSource	1	1.00	1	1	2	1	2	2	13	6	0
4	Clique	1	1.00	1	1	1	1	1	1	0	0	0
5	DefaultsForm	0	0.00	1	0	0	0	1	1	0	0	0
6	ECDDefaults	1	1.00	0	1	1	1	0	0	1	1	0
7	Evaluator	2	2.00	0	1	1	2	0	0	3	2	1
8	EvolutionState	2	2.00	0	1	1	2	0	0	8	40	7
9	Evolve	0	0.00	0	0	0	0	0	0	2	15	0
10	Exchanger	2	2.00	0	1	1	2	0	0	6	0	0
11	Finisher	2	2.00	0	1	1	2	0	0	1	0	0
12	Fitness	1	1.00	0	1	1	1	0	0	8	1	0
13	Group	1	1.00	1	1	1	1	2	2	1	0	0

Figure 6.4. Software metrics for Business Tier

Microsoft Excel - Data Access Tier Metrics.xls

	A	B	C	D	E	F	G	H	I
	ClassID	Class Name	TNSC	TNDO	TNIUO	NSC	ANSC	TNSQ	TNGB
1	1	BookType	1	0	0	1	1	0	0
3	2	BorrowableCopy	6	0	0	6	1	1	0
4	3	loan	1	2	2	8	1.6	0	0
5	4	Reservation	1	1	1	3	1	0	0
6	5	Return	2	1	0	5	1.67	0	0

Figure 6.5. Software metrics for Data Access Tier

6.3.2 Principal component analysis module

As presented in Section 3.3.2, if a group of variables in a data set are strongly correlated, these variables are likely to measure the same underlying dimension (i.e., class property) of the object. Some software metrics have high correlation with each other as shown in Table 6.3. Principal Components analysis transforms raw data into variables that are not correlated to each other. The varimax rotation method was used in this study. The varimax rotation enhances the interpretability of the principal

CHAPTER 6

components. With the varimax rotation, each component correlates high with a smaller number of variables and low on the other variables. It minimizes the number of variables that have high loadings on each factor. It simplifies the interpretation of the factors. This module generates the principal components whose Eigen values are larger than 1.0. It reads the metrics from the database and transforms them into principal components that are not correlated to each other. The principal components can be used to reduce the number of variables and to detect structure in the relationships between software metrics. It also produces the correlation analysis report and the descriptive statistics report. Sample outputs for principal components are shown in Table 6.4 and Table 6.5. The Eigen value, the percentage of the data set variance each PC describes, and the cumulative variance percentage are also provided in these tables.

Table 6.3. Correlation metrics for Data Access Tier metrics

	TNSC	TNIUO	TNDO	ANSC	TNSQ	TNGB
TNSC	1.000	.687	.160	.203	.530	.272
TNIUO	.687	1.000	.644	.022	.656	.271
TNDO	.160	.644	1.000	-.114	.513	-.036
ANSC	.203	.022	-.114	1.000	.181	.118
TNSQ	.530	.656	.513	.181	1.000	.049
TNGB	.272	.271	-.036	.118	.049	1.000

6.4 An Analysis of Metrics Extracted Using the Tool Developed

The sample data has been collected from the subsystems of an application system which is used in the Hwafuh (Myanmar) company, which is a fully networked information system. The experiment data was collected from the payroll subsystem, time record subsystem, human resource function subsystem and piece calculation subsystems. All the subsystems are implemented using Microsoft Visual C++.

CHAPTER 6

Table 6.4. Generated principal components for Data Access Tier metrics

Metrics	PC1	PC2
TNSC	.767	.335
TNIUO	.932	-.094
TNDO	.660	-.573
ANSC	.169	.677
TNSQ	.828	-.097
TNGB	.294	.595
Eigenvalues	2.693	1.271
% Variance	44.883	21.189
Cummulative % Variance	44.883	66.072

For Data Access Tier metrics, PCA identified two PCs, which capture 66.07% of the data set variance; Table 6.3 shows for each rotated component the coefficients of the measure, with the coefficients larger than 0.5 set in boldface. About 66.07% of the total variance in the six standardized variables is attributable to the first two components. The remaining four components together accounts for 33.93% of the total variance. Based on the analysis of the coefficients associated with each metric within each of the two rotated components, the PCs are interpreted as follows:

The first component is highly correlated with TNSC, TNIUO, TNDO and TNSQ. It explains a variance of 2.693 which is 44.883% of the total variance of six. TNIUO is the best representative, however, because it is less correlated with the other component.

The second component is most highly correlated with ANSC. It explains a variance of 1.271 which is 21.189% of the total variance of six. This suggests that TNIUO and ANSC metrics should be focused on in further analysis for this system.

CHAPTER 6

Table 6.5. Generated principal components for Business Tier metrics

Metrics	PC1	PC2	PC3
LCOM	-.171	-.068	.582
DIT	-.122	.948	-.010
NOP	-.111	.950	-.003
CBO	.250	.116	.597
NOC	-.098	-.461	-.024
RFC	.982	.009	.001
NMA	.992	-.034	-.002
NA	.069	-.213	.801
WMC	.991	-.039	-.005
AMC	-.075	.196	.683
Eigenvalues	3.070	2.118	1.804
% Variance	30.695	21.181	18.044
Cummulative% Variance	30.695	51.876	69.920

For Business Tier metrics, PCA identified three PCs as shown in Table 6.5. About 69.92% of the total variance in the ten standardized variables is attributable to the first three components. The remaining seven components together accounts for 30.08% of the total variance. Based on the analysis of the coefficients associated with each metric within each of the three rotated components, the PCs are interpreted as follows:

The first component is highly correlated with RFC, NMA and WMC. It explains a variance of 3.070 which is 30.695% of the total variance of 10. Thus, the first component can be labeled a complexity component.

The second component is most highly correlated with DIT and NOP. It explains a variance of 2.118 which is 21.181% of the total variance of ten. This component is highly related to inheritance properties.

The third component is most highly correlated with LCOM, CBO, NA and AMC. It explains a variance of 1.804 which is 18.044% of the total variance of ten. Thus, the third component can be labeled a cohesion-coupling component. These automatically generated analysis reports efficiently provide the study of software

CHAPTER 6

measurement and software quality management. Therefore principal component analysis tool has been embedded in this Metrics Extraction and Analyzer Tool.

6.5 Conclusion

In this study, we have implemented the metrics extraction and analyzer tool for Business Tier and Data Access Tier source codes. The existing object-oriented metrics are used for the Business Tier source codes. The proposed metrics are used for the Data Access Tier source code. These metrics are validated using real world application data which are collected from the warehouse management system (WMS) and the Hwafuh information system (HIS) as presented in Chapter 5. This system can be extended for other language like C# and other advanced languages which support N-tier applications.

Chapter 7

7. Conclusion and Recommendations

7.1 Conclusion

This chapter reviews the research presented in this thesis. The contributions of the research are summarized. The finding in relation to the initial research question is also presented. Finally, some suggestions on future research arising from this research are discussed.

These are the main contributions of this thesis

- Introducing the use of neural networks in software quality estimation using object-oriented design metrics and evaluation of that prediction model using real word applications; the Human Machine Interface System and Quality Evaluation System. These works have been published in [91, 111].
- Proposing a new SQL metrics for prediction of the software development faults in database application and validation of the metrics using the Warehouse Application Data. General Regression Neural Network and Ward Neural Network are used to evaluate the capability of this set of SQL metrics in predicting the number of faults in database applications. These works have been published in [89,90].
- Building a novel software readiness model on N-tier application architecture basis. This work has been published in [92]. The revised and enhanced versions are submitted to [112, 113].

We have also implemented a prototype system for the metric extraction and analyzer tool which is capable of generating software metrics which are used as independent variables in software quality prediction models.

7.2 Findings in Relation to the Initial Research Questions

In section 1.3 of this thesis, the primary research questions for this research are posted. These questions are repeated below and findings in relation to each are presented.

- Can we build a predictive model that is able to answer three questions'?
 - (i) How many faults (defects) are remaining the programs?
 - (ii) How many lines are required to be repaired to correct these errors?
 - (iii) How much time is required for the above purpose?

As presented in Chapter 5, our predictive models predict the number of faults, the number of lines to correct errors and required time (in minutes) for N-tier applications. The prediction results are tabulated in Table 5.8, 5.10 and 5.12. The small two-tailed probability p values ($p < 0.0001$) confirm, with a high degree of confidence, that these dependent variables can be predicted from our predictive models. The correlation of the predicted variables and the observed variables is represented by the correlation coefficient (r). The high r values represent high correlations between the actual versus predicted output. The results of the analysis show that our predictive models can answer these three questions.

- Can these models be applied to N-tier architecture based applications?

The prediction model is divided into three parts: (1) prediction model for Presentation Logic Tier (2) prediction model for Business Tier and (3) prediction model for Data Access Tier because each layer serves distinct and separate tasks in N-tier application architecture. The classification scheme was based on the nature of faults and their relevance to the application architecture. Predictive models need to use different software metrics as independent variables. Existing object-oriented metrics and complexity software metrics are used in the Business Tier prediction model. New sets metrics have been proposed for the Presentation Logic Tier and Data Access Tier. These models are validated using N-tier application data which are collected from warehouse management system (WMS) and Hwafuh information

CHAPTER 7

system (HIS). From the experiments results as presented in the Chapter 5, these models can be applied to N-tier architecture based applications.

- How can we predict the number of faults (defects) in PL/SQL files embedded in database applications'?

Firstly, we have studied the relationship between the Fault occurrence in PLISQL codes for database applications and static measures for database operations statements. We proposed new SQL metrics that have strong relationship with faults and then performed empirical validation for these metrics the General Regression Neural Network (GRNN) and Ward Neural Network. From the results presented in Chapter 4, our proposed SQL metrics in this study appear to be useful in predicting faults in PLISQL files.

- Can artificial neural networks and genetic training strategy be applied to predicting software quality for object-oriented systems?

In Chapter 3, we presented GRNN and Ward neural network models of the Human Machine Interface system and Quality Evaluation System, in predicting the number of software defects and the maintenance effort. Table 3.5 and Table 3.11 summarized the prediction results of the neural network models. They gave better prediction result than regression analysis which results are shown in Appendix A and B.

As shown in Table 3.11, GRNN network model can predict better than Ward network, although the performances of Ward network can also be considered rather acceptable. In particular, GRNN network predicts about 76% of the cases within a relative error less than 25% (PRED25) and almost 89% of the cases with a relative error less than 50% (PRED50). Also, the mean relative error is 19.52%. These results give very good results comparing with other studies [68]. From these results neural networks are useful in predicting maintenance effort.

In Chapter 5, we use genetic net to show the importance of input values. It is a relative measure of how significant each of the inputs is in the predictive model. We can then decide to remove the variables with the lowest Relative Importance of Inputs

value. We have compared this approach to a multiple regression analysis approach. By comparing empirical results as shown in Table 5.13 including R-square and RMSE, Genetic Nets give clearly better prediction results than Regression analysis

7.3 Other Findings

As shown in Table 5.7, SQL complexity metrics (average number of search condition criteria of where-clause) is important for all three predictions for Data Access Tier. That shows the complexity of SQL is highly related to occurrence of faults and maintenance effort level in this Tier. From database manipulation operations statements, insert/update operations is more important than other operations (i.e. delete and select) in prediction of number of faults and required time (in minutes) to correct. As shown in Table 5.9, for business tier source codes, object-oriented metrics such as RFC, DIT and NA are important for all three predictions but the importance of traditional software metrics (the average McCabe's cyclomatic complexity metrics) and other OO metrics such as LCOM and NOC are relatively less significant. It was also found that the number of all member function metric (WMC) will lose its importance if WMC counts all member function including simple function such as put and get methods. The relative importance of metrics of prediction models for N-tier applications is summarized in Table 7.1

Table 7.1. The relative importance of metrics for N-tier applications

Tier	Metrics	Relative importance
Data Access Tier	ANSC, TNIUO, NDO, TNSC	important for all three predictions (defects, lines, minutes)
	TNIUO	more important than other database operations statements
Business Tier	NOP, RFC, DIT, NA, CBO, NMA	important for all three predictions
	LCOM	not important for defects and lines predictions
	NOC, WMC	not important for minutes (time/effort) predictions
	AMC	not important for defects predictions
Presentation Logic Tier	NUIO, TNM	important for all three predictions

CHAPTER 7

As presented in Chapter 3 and 4, the results suggest the actual data has better conformity with GKNN prediction than with Ward. Ward neural network is a backpropagation network with different activation functions. Although backpropagation network is the most popular network for practical applications and a very powerful tool, it has sensitivities of backpropagation (e.g. selection of random number seed to select weight). GRNN has no training parameters such as learning rate and momentum in backpropagation network, but there is a smoothing factor that is applied after the network is trained. Smoothing factor is the only adjustable parameter in GKNN, making overtraining less likely in GRNN. The smoothing factor allows the GRNN to interpolate between patterns or spectra in the training set. These make GKNN able to give better prediction especially for production set.

7.4 The Cross Section Study

In this thesis, three prediction models have been built for three types of applications system: (1) object oriented application (2) PL/SQL application and (3) N-tier application. The comparison of prediction of results is shown in Table 7.2. The correlation between the predicted value and the observed value is represented by the coefficient of correlation (r). The applications' complexities used in this study have shown in terms of the total number of conditional branching, the total number of loop, total number of function calls and total number of parameters.

Table 7.2 Comparison of prediction results for three applications

		OO application	PL/SQL application	N-tier application		
				Business tier	Data access tier	Presentation logic tier
Prediction results	r	0.9531	0.9167	0.8345	0.860836	0.828182
	t	0.9084	0.8404	0.8257	17.000590	11.05799
	P value	<0.001	<0.0001	<0.0001	<0.0001	<0.0001
Applications' Size and Complexity	Total No. of lines	4955	20952	84943	19982	19952
	No. of files	97	108	173	103	58
	No. of variables	965	4968	2941	4429	754

CHAPTER 7

	(global +local)					
	Conditional branching	720	1156	1285	1100	157
	Looping	672	112	1199	110	187
	No. of function calls	2204	4297	3930	4038	1300
	No. of parameters	4508	8900	6560	8090	2560
	Total complexity	8104	14465	12974	13338	4204
	Average Complexity	83.54639	149.1237	133.7526	137.5052	43.34021
	Complexity per loc	0.16208	0.68881	0.152635	0.6669	0.144966

In terms of predictability, the best value of correlation between the predicted value and the observed value is obtained from OO application and the second best value is obtained from PL/SQL files and the third best value is obtained from N-tier application. In terms of size for the applications that are used in this study, N-tier application is the largest and the PLISQL files are the smallest. In terms of complexity for the applications that are used in this study, N-tier application is the most complex and the OO application the least complex one. In term of r value, large and complex applications seem to be reduced accuracy of predictability because of their high uncertainties.

7.5 Threads to Validity and Limitation

7.5.1 Construct validity and limitation

The dependent variable used in this study is the number of faults, the number of lines as well as the time (measured by number of minutes) required to be repaired to correct these faults during acceptance testing. Assuming testing was performed properly and thoroughly, the number of faults in a class, the required number of lines and the required number of minutes needed for repair during acceptance testing should be valid measures. However, there is always a possibility that subjective

human biases may creep in at the data recording phase when the software developers measure and record their observations into the journal files. This in turn could have led to different results in the NN training and prediction. This is one of the limitations of this study.

Several software metrics are used as the independent variables in this study. However, since testing fault data is needed to perform the analysis, measurement was performed only after implementation was completed. If measurement had taken place, say, before implementation started, different measurement data could have been obtained because of the uncertainty inherent to early design information. This in turn could have led to different results in the statistical analyses. Therefore, I make it sure to collect the data from the source files electronically after acceptance testing. The construct validity of the independent variable can be considered satisfactory.

7.5.2 Internal validity and limitation

The analysis performed here is correlational in nature. I have demonstrated that several of the measures investigated had a statistically and practically significant relationship with the number of fault during testing in Section 3.3.4, 3.4.4, 4.5 and 5.5. They only provide empirical evidence of it. Only controlled experiments, where the measures would be varied in a controlled manner and all other factors would be held constant, could really demonstrate causality. However, such a controlled experiment would be difficult to run since varying coupling, cohesion, and inheritance in a system, while preserving its functionality, is difficult in practice. Some attempts to do so are reported in [21].

7.5.3 External validity and limitation

Threats to external validity are conditions that limit the ability to generalize the results. The system developed for the Presentation Logic Tier is rather small: containing 58 classes. That fact may restrict the generalizability of the presentation tier results. This is one of the limitations of this study. We chose the genetic training strategy for these numbers of patterns. The genetic training method trains everything in an out-of-sample mode; it is essentially doing a "one-hold-out" technique, also called "jackknife" or "cross validation". This method is therefore

extremely effective when there are no many patterns to train although the genetic training method takes a little longer to train [83]. A genetic algorithm solves the problems by creating a population or group of possible solutions to the problem. The genetic algorithm tests many weighting schemes until it finds the one that gives the best predictions for the training data. Genetic learning stores every set of inputs and related output in the training data. When the net is applied to a new set of inputs, the new inputs are compared to every set of stored inputs. Depending upon how close the match is, the output for each training row is weighted. Closer matches receive higher weights, and inputs that are farther away from the training inputs receive lower weights. The predicted output for the new set of inputs is a "weighted" average of all of the outputs with which the network was trained. Therefore, we introduced using the genetic training strategy to obtain the reliable results.

7.6 Recommendations for Further Research

We have proposed a new predictive model which is applicable for N-tier applications which support global computing in the Internet. This is an interesting direction. Currently, most of the software quality models do not consider tier basis software metrics. This investigation can be extended for a wide range of N-tier applications and proposing a new set of Presentation Logic Tier metrics regarding various types of user interface objects. This can be another future research area.

In this study, we have implemented the metrics extraction and analyzer tool for Business Tier and Data Access Tier source codes to support software measurement and software quality management. It can analyze Java or VC++ or C++ source code. This system can be extended for other language like C# and other advanced languages which support N-tier applications.

Author's Publications

- [1] Mie Mie Thet Thwin and Tong-Seng Quah, "Application of Neural Networks for Estimating Software Quality Prediction Using Object-Oriented Metrics", Journal of Systems and Software, vol. 76, no. 2, 2005, pp. 147-156.
- [2] Tong-Seng Quah and Mie Mie Thet Thwin, "Prediction of Software Development Faults in PLISQL Files using Neural Network Models", Journal of Information and Software Technology, vol. 46, No. 8, pp 519-523, 2004.
- [3] Tong-Seng Quah and Mie Mie Thet Thwin, "Prediction of Software Development Faults in PLISQL Files using Genetic Nets", WSEAS Transactions on Computers, vol. 3, no.1, 2004, pp. 228-233.
- [4] Tong-Seng Quah and Mic Mic Thet Thwin, "Application of neural networks for software quality prediction using object-oriented metrics", Proceedings of IEEE Ninetccnth International Conference on Software Maintenance, ICSM 2003, 22-26 Sept., 2003, pp. 116 – 125.
- [5] Mic Mic Thet Thwin and Tong-Seng Quah, "Application of Neural Networks for Estimating Software Maintainability Using Object-Oriented Metrics", Fifteenth International Conference on Software Engineering and Knowledge Engineering, San Francisco, June 2003, Pp. 69-73.
- [6] Mie Mie Thet Thwin and Tong-Scng Quah, "Application of Neural Network for Predicting Software Development Faults Using Object-Oriented Design Metrics", Prceceding of 9th IEEE Internatinal Conference on Neural Information Processing, vol.5, 2002, pp. 2312-2316.
- [7] Jon T. S. Quah and Mie Mie Thet Thwin, "Prediction of Software Readiness Using Neural Network", Proceedings of 1st IEEE International Conference on Information Technology & Applications (ICITA 2002), Australia, 2002, pp. 25-28.

AUTHOR'S PUBLICATIONS

In process

- [1] Mie Mie Thet Thwin and Tong Seng Quah, "Estimating Software Readiness Using Predictive Models", Journal of Information Sciences. (Revised version submitted on 17-Nov-2005)

- [2] Mie Mie Thet Thwin and Tong Seng Quah, "Estimating Software Readiness for N-tier Applications", Journal of Data & Knowledge Engineering (Revised version submitted on 12-Dec-2005)

- [3] Mie Mie Thet Thwin and Tong-Seng Quah, "An Implementation of Metrics Extraction and Analyzer Tool", International Journal on Artificial Intelligence Tools (IJAIT). (Submitted on 5-Sep-2004)

Bibliography

- [1] M.A. de Almeida, H. Lounis and W.L. Melo, "An investigation on the use of machine learned models for estimating correction costs" Proceedings of the 20th international Conference on Software Engineering, 1998, pp. 473–476.
- [2] Sultan H. Aljahdali et al., "Prediction of Software Reliability: "A Comparison between Regression and Neural Network Non-Parametric Models", Proceedings of ACS/IEEE International Conference on Computer Systems and Applications, 2001, pp. 470-473.
- [3] M. Alshayeb and Li Wei, "An empirical validation of object-oriented metrics in two different iterative software processes", IEEE Transactions on Software Engineering, vol. 29, no. 11, Nov. 2003, pp.1043 – 1049.
- [4] C. Anderson, A. von Mayrhauser, Tom Chen, "Assessing neural networks as guides for testing activities", Proceedings of the 3rd international Software Metrics Symposium, 1996, pp. 155 –165.
- [5] R.K. Bandi, V.K. Vaishnavi and D.E. Turk, "Predicting maintenance performance using object-oriented design complexity metrics", IEEE Transactions on Software Engineering, vol. 29, no. 1, Jan. 2003, pp.77 – 87.
- [6] J. Bansiya and C.G. Davis, "A Hierarchical Model for Object-Oriented Design Quality Assessment," IEEE Transactions on Software Engineering, vol. 28, no. 1, Jan. 2002, pp. 4- 17.
- [7] V.R. Basili, S.E. Condon, K. El Emam, R.B. Hendrick, and W. Melo, "Characterizing and modeling the cost of rework in a library of reusable software components", Proceedings of the international conference on Software engineering, 1997, pp. 282–291.
- [8] V.K. Basili, et al., "A Validation of Object-Oriented Design Metrics as Quality Indicators", IEEE Transactions on Software Engineering, vol. 22, 1996, pp. 751-761.
- [9] Victor R. Basili and Barry Perricone, "Software Errors and Complexity: An Empirical Investigation", Communication of the ACM, vol. 27, 1984, pp. 42-52.

BIBLIOGRAPHY

- S. Benlarbi, W.L. Melo, "Polymorphism measures for early risk prediction", Proceedings of the 1999 International Conference on Software Engineering, 1999, pp. 334-344.
- E.V. Berard, "Bringing testing into the fold [software testing]", IEEE Software, vol. 13, no. 3, May 1996, pp.91 - 92, 95.
- J.M. Bieman, B.-K. Kang, "Cohesion and Reuse in an Object-Oriented System", Proceeding of ACM Symposium on Software Reusability (SSR'94), 1995, pp. 259-262.
- Mounir Boukadoum, Houari A. Sahraoui and Hakim Lounis, "Machine Learning Approach to Predict Software Evolvability using fuzzy binary trees", Preceeding of International Conference on Artificial Intelligence, 2001.
- L Briand, W.L Melo, J. Wust, "Assessing the applicability of fault-proneness models across object-oriented software projects", IEEE Transactions on Software Engineering, vol. 28 pp. 706-720, 2002.
- L. Briand, J. Wiist, John W. Daly and V. Porter, "Exploring the Relationships between Design Measures and Software Quality in Object-Oriented Systems", Journal of Systems and Software, vol. 51, pp. 245-273, 2000.
- L. Briand, T. Langley and I. Wiecezorek, "A replicated assessment and comparison of common software cost modeling techniques", Proceedings of International Conference on Software Engineering, 2000, pp. 377-386.
- L. Briand, J. Daly, and J. Wiist, "A Unified Framework For Coupling Measurement in Object-Oriented Systems," IEEE Transactions on Software Engineering, vol. 25, no. 1, pp. 91-121, 1999.
- L.C. Briand, J. Wiist, S. Ikonomovski and H. Lounis, "Investigating Quality Factors in Object-Oriented Designs: an Industrial Case Study", Proceedings of the 21st IEEE International Conference on Software Engineering ICSE, 1999, pp. 345-354.
- L. Briand, J. Daly, J. Wuest. "A Unified Framework for Cohesion Measurement in Object-Oriented Systems", Empirical Software Engineering - An International Journal, vol. 3, no. 1, 65-117, 1998.
- L. Briand, J. Daly, V. Porter, J. Wuest, "Predicting Fault-Prone Classes based on Design Measures in Object-Oriented Systems", Proceedings of IEEE International Symposium on Software Reliability Engineering (ISSRE), 1998.

BIBLIOGRAPHY

- [21] Lionel Briand, Prem Devanbu and Walcelio Melo, "An investigation into coupling measures for C++", Proceedings of the 19th International Conference on Software Engineering, 1997, pp. 412-421.
- [22] S. Brocklehurst, P.Y. Chan, B. Littlewood and J. Snel, "Recalibrating software reliability models", IEEE Transactions on Software Engineering, vol. 16 Issue: 4, 458-469, 1990.
- [23] M. Cartwright and M. Shepperd, "An Empirical Investigation of Object-Oriented Software System", IEEE Transactions on Software Engineering, vol. 26, pp. 786-796, 2000.
- [24] A. Chatzigeorgiou, "Mathematical assessment of object-oriented design quality", IEEE Transactions on Software Engineering, vol. 29, no. 11, Nov. 2003, pp. 1050 – 1053.
- [25] Robert Chartier, "Application Architecture: An N-Tier Approach", <http://www.15seconds.com>.
- [26] C.H. Chen, "Fuzzy Logic and Neural Network Handbook", New York, N.Y.: McGraw-Hill, Inc., 1996.
- [27] S.R. Chidamber, D.P. Darcy, and C.F. Kemerer, "Managerial use of metrics for object-oriented software: an exploratory analysis", IEEE Transactions on Software Engineering, vol. 24, no. 8, Aug. 1998, pp. 629 – 639.
- [28] S.R. Chidamber, and C.F. Kemerer, "A Metrics Suite for Object Oriented Design", IEEE Transactions on Software Engineering, vol. 20, pp. 476-493, 1994.
- [29] S.R. Chidamber, and C.F. Kemerer, "Towards a Metrics Suite for Object Oriented Design", Proceeding of the 6th ACM Conference on Object Oriented Programming, Systems, Languages and Applications (OOPSLA), pp. 197-211, 1991.
- [30] W.C. Chu, Jucz-Nan Chen, Chun-Yuan Lee and Y. Hongji, "Implementing an agent system using N-tier pattern-based framework", Proceedings of 25th Annual International Computer Software and Applications Conference, 2001, pp. 451 – 456.
- [31] S. Conte, H. Dunsmore and V. Shen, "Software Engineering Metrics and Models", Benjamin/Cummings, 1986.
- [32] Distributive Software, <http://www.distributive.com>.

BIBLIOGRAPHY

- J.J. Dolado, "A validation of the component-based method for software size estimation", IEEE Transactions on Software Engineering, vol. 26 , no. 10 ,2000, pp. 1006 – 1021.
- K. E. Emam, W. Melo, and C.M. Javam, "The Prediction of Faulty Classes Using Object-Oriented Design Metrics", Journal of Systems and Software, Elsevier Science, Vol. 56, No. 1, pp. 63-75, March 2001.
- K. E. Emam, S. Benlarbi, N. Goel, and S. N. Rai, "The confounding effect of class size on the validity of object-oriented metrics", IEEE Transactions on Software Engineering, Vol. 27, No. 7, pp. 630-650, July 2001.
- L. Etzkorn, H. Delugach, "Towards a semantic metrics suite for object-oriented design", Proc. 34th International Conference on Technology of Object-Oriented Languages and Systems, 2000, pp. 71 –80, 2000.
- N.E. Fenton and N. Ohlsson, "Quantitative analysis of faults and failures in a complex software system", IEEE Transactions on Software Engineering, vol. 26,2000, pp. 797-814.
- Norman E. Fenton and Martin Neil, "Software metrics: success, failures and new directions", The journal of Systems and Software, vol. 47, 1999, pp. 149-157. [Fenton 1999]
- F. Fioravanti, "A metric framework for the assessment of object-oriented systems", Proceedings of IEEE International Conference on Software Maintenance, 2001, pp. 557–560.
- F. Fioravanti, P. Nesi, A study on fault-proneness detection of object-oriented systems", Preceding of Fifth European Conference on Software Maintenance and Reengineering, 2001, pp. 121 –130.
- John E. Frenund, Frank J. Williams, Benjamin M. Perles, "The Elementary Business Statistics- The Modern Approach", Prince-Hill, 1993.
- D. Glasberg, K. El Emam, W. Melo, and N. Madhavji, "Validating Object-oriented Design Metrics on a Commercial Java Application". Technical Report, NRCIERB-1080, NRC 44146,2000.
- A. L. Goel, "Software Reliability Models: Sssumptions, Limitations, and Applicabilty", IEEE Transactions on Software Engineeering, vol.11, no. 12, 1985.

BIBLIOGRAPHY

- [44] D.E Goldberg, "Genetic Algorithms in Search, Optimization, and Machine Learning", Reading, Mass: Addison-Wesley, 1989.
- [45] S. Gokhale, P.N. Marinos and K.S. Trivedi, "Important Milestones in Software Reliability Modeling," Proceeding of Software Engineering and Knowledge Engineering (SEKE 96) , 1996, pp. 345-352.
- [46] C. Gould, Su Zhendong and P. Devanbu," JDBC checker: a static analysis tool for SQL/JDBC applications", Proceedings of 26th International Conference on Software Engineering, 2004. ICSE 2004. , 23-28 May 2004, pp. 697 – 698.
- [47] Todd L. Graves, Alan F. Karr, J.S. Marron, and Harvey Siy, "Predicting Fault Incidence Using Software Change History", IEEE Transactions on Software Engineering, vol. 26, no. 7,2000, 653-661.
- [48] M. Hitz, and B. Montazeri, "Chidamber and Kemerer's metrics suite: a measurement theory perspective", IEEE Transactions on Software Engineering, vol. 22, 1996, pp. 267-271.
- [49] Kapsu Kim and SungBack Hong, "Identifying fault prone modules: an empirical study in telecommunication system", Proceedings of the Second Euromicro Conference on Software Maintenance and Reengineering, 1998, pp.179 – 183.
- [50] S.A. Hossain and R.C. Dahiya, "Estimating the parameters of a non-homogeneous Poisson-process model for software reliability", IEEE Transactions on Reliability, vol. 42, no. 4, 1993, pp. 604-612.
- [51] Janco Associates, Inc., <http://www.psrinc.com/metrics.htm>.
- [52] R. Jeffery, M. Ruhc and I. Wiecek, "A comparative study of two software development cost modeling techniques using multi-organizational and company-specific data", Journal of Information and Software Technology, vol. 42, no.14, 2000. pp. 1009–1016.
- [53] JMetric, <http://www.it.swin.edu.au/projects/jmetric/products/jmetric/>
- [54] M. Jorgensen, "Experience with the accuracy of software maintenance task effort prediction models", IEEE Transactions on Software Engineering, vol. 21, no. 8, 1995, pp. 674 – 681.
- [55] T. Kamiya, Kusumoto, S., K. Inoue, "Prediction of fault-proneness at early phase in object-oriented development", Proceedings of the Second IEEE

BIBLIOGRAPHY

International Symposium on Object-Oriented Real-Time Distributed Computing. 1999, pp. 253-258.

N. Karunanithi, D. Whitley, and Y.K. Malaiya, "Prediction of software reliability using connectionist models", IEEE Transactions on Software Engineering, vol. 18, pp. 563-574, 1992.

N. Karunanithi et al., "Prediction of software reliability using neural networks", Proceedings IEEE International Symposium. Software Reliability Engineering, pp 124-130, 1991.

R Kempkens, P Rosch, L Scott, J Zettel, "A multi-layer multi-view architecture for software engineering environment," Information and Software Technology, vol. 42,2000, pp 141-149.

T. M. Khoshgoftaar, E.B. Allen, Z. Xu, "Predicting testability of program modules using a neural network", Proceedings of the 3rd IEEE Symposium on Application-Specific Systems and Software Engineering Technology, 2000, pp. 57-62.

T. M. Khoshgoftaar, E. B. Allen, J. P. Hudepohl, S. J. Aud, "Application of neural networks to software quality modeling of a very large telecommunications system", Neural Networks, IEEE Transactions on ,vol. 8, no. 4, pp. 902 – 909, July 1997.

T.M. Khoshgoftaar, R.M. Szabo and P.J. Guasti, "Exploring the behaviour of neural network software quality models", Software Engineering Journal , vol. 10, no. 3, 1995, pp. 89 – 96.

T.M. Khoshgoftaar, D.L. Lanning and A.S. Pandya, "A Comparative Study of Pattern Recognition Techniques for Quality Evaluation of Telecommunications Software", IEEE Journal on Selected Areas in Communications, vol. 12, 1994, pp. 279-291.

N. Li, "Measurement and Enhancement of Software Reliability Through Testing", Ph. D. thesis, Colorado State University, 1996.

Wei Li and S. Henry, "Object-Oriented Metrics that Predict Maintainability", Journal of Systems and Software, vol. 23, 1993, pp. 111-122.

Tim Littlefair, "An Investigation into the use of Software Code Metrics in the Software Development Environment", Ph.D. Thesis, Edith Cowan University, June 2001.

BIBLIOGRAPHY

- [66] Mark Lorenz, Jeff Kidd, "Object-Oriented Software Metrics", Prentice Hall Object-Oriented Series, 1994.
- [67] Christopher M. Lott, <http://www.chris-lott.org/resources/cmetrics/>.
- [68] A. De Lucia, E. Pompella and S. Stefanucci, "Assessing effort estimation models for corrective maintenance through empirical studies", Journal of Information and Software Technology, vol. 47, no. 1, 2005, pp. 3-15.
- [69] M Squared Technologies, <http://msquaredtechnologies.com>.
- [70] Y.K. Malaiya and J. Denton, 'Estimating the number of Residual Defects', HASE '98, 3rd IEEE Int'l High-Assurance Systems Engineering Symposium, Maryland, USA, November 13-14, 1998.
- [71] Y. K. Malaiya, N. Karunanithi, P. Verma, "Predictability measures for software reliability models", Proceedings of Fourteenth Annual International Computer Software and Applications Conference, 1990, pp. 7-12.
- [72] Yida Mao, H.A. Sahraoui, and H. Lounis, "Reusability hypothesis verification using machine learning techniques: a case study", Proceedings of the 13th IEEE International Conference on Automated Software Engineering, pp. 84-93, 1998.
- [73] K. Matsumoto, K. Inoue, T. Kikuno, K. Torii, "Experimental evaluation of software reliability growth models", Eighteenth International Symposium on Fault-Tolerant Computing, 1998, pp. 148-153.
- [74] McCabe & Associates, Inc., <http://www.mccabe.com/>.
- [75] S. McConnell, "Gauging software readiness with defect tracking", IEEE Software, Vol. 14, no. 3, 1997, pp. 136, 135.
- [76] Brian Meek and K.K. Siu, "The effectiveness of error seeding", ACM Sigplan Notices, vol. 24, pp 81-89, 1989.
- [77] H.D. Mills, "On the Statistical Validation of Computer Programs," FSC-72-6015, IBM Federal Systems Div., Gaithersburg, Md., 1972.
- [78] James Miller, "Statistical significance testing--a panacea for software technology experiments?" Journal of Systems and Software, vol. 73, no. 2, pp. 183-192, October 2004.
- [79] J.D. Musa, et al., "Software Reliability-Measurement, Prediction, Applications", New York: McGraw-Hill, 1987.

BIBLIOGRAPHY

- J.C. Munson, and T. M. Khoshgoftaar, "The detection of fault-prone programs", IEEE Transactions on Software Engineering, vol. 18, pp. 423-433, 1992.
- Glenford J. Myers, "The Art of Software Testing", John Wiley & Sons, Inc., 2004.
- Glenford J. Myers, "Software reliability: principles and practices", John Wiley & Sons, Inc., 1976.
- NeuroShell Predictor Help, Ward Sysms Group, Inc. <http://www.wardsystems.com>.
- NeuroShell 2 Help, Ward Sysms Group, Inc. <http://www.wardsystems.com>.
- Duncan Pemberton, "VOCAL: Interactive System & Tool Support", Cooperative Systems Engineering Group Technical Reports, 1997.
- Power Software, <http://www.powersoftware.com/>.
- Predicate Logic, Inc., <http://www.predicate.com>.
- QSM, Quantitative Software Management Inc., <http://www.qsm.com/>.
- Tong-Seng Quah, Mie Mie Thet Thwin, "Prediction of Software Development Faults in PL/SQL Files Using Neural Network Models", Information and Software Technology, vol. 46, No.8, pp. 519-523, 2004.
- Tong-Seng Quah and Mie Mie Thet Thwin, "Prediction of Software Development Faults in PL/SQL Files using Genetic Nets", WSEAS Transactions on Computers, vol. 3, no.1, 2004, pp. 228-233.
- Tong-Seng Quah and Mie Mie Thet Thwin, "Application of neural networks for software quality prediction using object-oriented metrics", Proceedings of IEEE International Conference on Software Maintenance, ICSM 2003, 22-26 Sept., pp. 116 – 125, 2003.
- Jon T. S. Quah and Mie Mie Thet Thwin, "Prediction of Software Readiness Using Neural Network", Proceedings of 1st IEEE International Conference on Information Technology & Applications (ICITA 2002), Australia, 2002, pp. 25-28.
- J. R. Quinlan, "C4.5: programs for machine learning", Morgan Kaufmann Publishers Inc., 1993.

BIBLIOGRAPHY

- [94] A.P. Nikora and J.C. Munson, "Developing fault predictors for evolving software systems", Proceedings of Ninth International Software Metrics Symposium, 2003. pp. 338 – 350.
- [95] N.J. Pizzi, A.R. Summers and W. Pedrycz. "Software quality prediction using median-adjusted class labels", Proceedings of the 2002 International Joint Conference on Neural Networks, vol. 3 , 2002, pp. 2405 – 2409.
- [96] Raghu Ramakrishnan, and Johannes Gehrke, "Database Management Systems", McGRAW-Hill, 3rd Edition, 2003.
- [97] Ralf ReiBing, "Towards a Model for Object-Oriented Design Measurement", Proceedings of the 5th International ECOOP Workshop on Quantitative Approaches in Object-Oriented Software Engineering, pp. 71-84,2001.
- [98] SERC (The Software Engineering Research Center), <http://www.cs.bsu.edu/homepages/metrics/tools/>,
- [99] SET Laboratories, Inc., <http://www.molalla.net/~setlabs/>.
- [100] David J. Sheskin and David Sheskin, "Handbook of Parametric and Nonparametric Statistical Procedures", Chapman & Hall/CRC, 2000.
- [101] S. Skevoulis and Jia Xiaoping, "Generic invariant-based static analysis tool for detection of runtime errors in Java programs", Proceedings of 37th International Conference on Technology of Object-Oriented Languages and Systems, 2000. TOOLS-Pacific 2000, pp. 36 – 44.
- [102] D.F Specht, "A general regression neural network", IEEE Transactions on Neural Networks, vol. 2, Issue: 6, pp. 568-576, 1991.
- [103] SPR (Software Productivity Research) , <http://www.spr.com/>.
- [104] Hans-Peter Steiert, "Towards a component-based n-Tier CIS-architecture"; Proceedings of the third international workshop on Software architecture, 1998, pp. 137 – 140.
- [105] James Stevens, "Applied multivariate statistics for the social sciences", Mahwah, N.J. Lawrence Elbaum Associates, 1996.
- [106] R. Subramanyam and M. S. Krishnan, "Empirical analysis of CK metrics for object-oriented design complexity: implications for software defects", IEEE Transactions on Software Engineering, vol. 29 , no. 4 , April 2003, pp. 297 – 310.

BIBLIOGRAPHY

- [107] Mei-Huei Tang, Ming-Hung Kao, Mei-Hwa Chen, "An empirical study on object-oriented metrics", Proceedings of the Sixth IEEE International Symposium on Software Metrics, pp. 242-249, 1999.
- [108] testingfaqs.org (An information resource for software testers), <http://www.testingfaqs.org/t-static.htm>.
- [109] Testwell Oy (Ltd), <http://www.testwell.fi/>.
- [110] William Thomas, Alex Delis, and Victor R. Basili, "An Analysis of Errors in Reuse-Oriented Development Environment", The Journal of Systems and Software, vol. 38, 1997, pp. 211-224.
- [111] Mie Mie Thet Thwin and Tong-Seng Quah, "Application of Neural Networks for Estimating Software Quality Prediction Using Object-Oriented Metrics", Journal of Systems and Software, vol. 76, no. 2, 2005, pp. 147-156.
- [112] Mie Mie Thet Thwin and Tong Seng Quah, "Prediction of Software Readiness Using Predictive Models", Journal of Information Sciences. (Revised version submitted on 17- Nov- 2005)
- [113] Mie Mie Thet Thwin and Tong Seng Quah, "Prediction of Software Readiness for N-tier Applications", Journal of Data & Knowledge Engineering. (Submitted on 12-Dec-2005)
- [114] Verifysoft Technology GmbH, <http://www.verifysoft.com/en.html>.
- [115] S. Vicinanza, T. Mukhopadhyay and M. Prietula, "Software effort estimation: an exploration study of export performance", Journal of Information System Research, vol. 2, no. 4, 1991, pp. 243–262.
- [116] Alen Wood, "Predicting Software Reliability", IEEE Computer, vol. 29, no. 11, pp. 69-77.
- [117] S. Yamada, M. Ohba and S. Osaki, "S-Shaped Reliability Growth Modeling for Software Error Detection", IEEE Transactions on Reliability, 1983, pp. 475-484.
- [118] Ping Yu, T. Systa, and H. Muller, "Predicting fault-proneness using OO metrics. An industrial case study", Proceedings of 6th European Conference on Software Maintenance and Reengineering, 2002, pp. 99 –107, 2002.
- [119] Xuemei Zhang, Xiaolin Teng and Hoang Pham, "Considering fault removal efficiency in software reliability assessment", IEEE Transactions on Systems, Man and Cybernetics, vol. 33, no. 1, 2003, pp. 114-120.

BIBLIOGRAPHY

- [120] Xu Zhiwei, T. M, Khoshgoftaar and E.B. Allen, "Prediction of software faults using fuzzy nonlinear regression modeling", Fifth IEEE International Symposium on High Assurance Systems Engineering, 2000, pp. 281 -- 290.

Appendix A Descriptive statistics and Regression Analysis for HMI system

Descriptive statistics of the HMI system data set

	Min	Max	Mean	Median	Std dev
WMC	1	89	8.9175	6	10.32762
DIT	0	5	1.2887	1	1.03036
NOC	0	17	0.4639	0	1.90428
CBO	0	28	2.319588	2	3.554769
RFC	0	397	35.79381	14	58.77498
IC	0	2	0.43299	0	0.557285
CBM	0	9	1.134021	0	1.955914
NOMA	0	8	1	1	1.399405
FAULTS	0	28	2.061856	1	3.529089

Model Summary

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate	Change Statistics				
					R Square Change	F Change	df1	df2	Sig. F Change
1	.925 ^a	.856	.843	1.39971	.856	65.283	8	88	.000

a. Predictors: (Constant), NOMA, CBO, DIT, IC, NOC, WMC, CBM, RFC

Coefficients^a

Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.
		B	Std. Error	Beta		
1	(Constant)	-.200	.296		-.677	.500
	WMC	.119	.029	.349	4.053	.000
	DIT	-.152	.168	-.044	-.903	.369
	NOC	.094	.090	.051	1.049	.297
	CBO	-.071	.047	-.072	-1.500	.137
	RFC	.020	.006	.328	3.403	.001
	IC	.054	.434	.009	.124	.901
	CBM	.464	.152	.257	3.059	.003
	NOMA	.261	.140	.103	1.856	.067

a. Dependent Variable: FAULTS

Appendix B Descriptive statistics and Regression Analysis for QUES system

Descriptive statistics of the QUES system data set

	Min	Max	Mean	Median	Std dev
DIT	0	4	1.915493	2	0.527703
MPC	2	42	17.74648	17	8.328468
RFC	17	156	54.43662	40	32.62065
LCOM	3	33	9.183099	5	7.308136
DAC	0	25	3.43662	2	3.91237
WMC	1	83	14.95775	9	17.05993
NOM	4	57	13.40845	6	12.00426
SIZE2	4	82	18.02817	10	15.20805
SIZE1	115	1009	274.3099	211	172.3721
CHANGE	6	217	64.22535	52	43.12745

Model Summary

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate	Change Statistics				
					R Square Change	F Change	df1	df2	Sig. F Change
1	.856 ^a	.734	.694	23.84591	.734	18.663	9	61	.000

a. Predictors: (Constant), SIZE1, DIT, MPC, LCOM, DAC, WMC, RFC, NOM, SIZE2

Coefficients^a

Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.
		B	Std. Error	Beta		
1	(Constant)	-3.423	16.622		-.206	.838
	DIT	6.685	7.532	.082	.888	.378
	MPC	-.659	.688	-.127	-.957	.342
	RFC	.497	.327	.376	1.520	.134
	LCOM	-4.129	1.441	-.700	-2.865	.006
	DAC	-13.822	4.046	-1.254	-3.417	.001
	WMC	-1.471	.499	-.582	-2.948	.005
	NOM	-4.236	3.699	-1.179	-1.145	.257
	SIZE2	5.857	3.671	2.065	1.595	.116
	SIZE1	.358	.056	1.430	6.358	.000

a. Dependent Variable: CHANGE

Appendix C SQL metrics and principal components data from PL/SQL files of warehouse management applications

	NSC	NIUO	NDO	NT	NSCC	NJQ	faults	PC1	PC2
alternate.sql	3	3	0	3	8	0	2	-0.35725	0.06913
archive.sql	29	2	8	36	62	15	7	2.0681	2.80832
aisle_locked.sql	2	0	0	2	3	0	1	-0.38584	-0.50034
asrs_util.sql	13	3	0	15	35	4	5	0.50049	0.02293
balance.sql	19	3	2	19	52	0	2	0.40464	0.836
bk_obj_l.sql	2	0	0	2	4	0	1	-0.38064	-0.50165
bk_use.sql	12	3	0	13	31	2	3	0.26302	0.04715
book_ihex_by_ihdt.sql	2	1	0	2	2	0	1	-0.402	-0.30815
book_itcx_by_itrf.sql	2	2	0	2	4	0	1	-0.40258	-0.1199
book_ohex_by_ohdt.sql	4	4	0	4	4	0	2	-0.35351	0.26865
book_ohlx_by_ohdt.sql	2	2	0	2	2	0	1	-0.41297	-0.11727
book_stocktake_data.sql	1	1	0	1	2	0	2	-0.43751	-0.31155
book_usrf_by_aprf_sl rf.sql	0	1	0	0	0	0	1	-0.48341	-0.31232
cancel_exist_redt.sql	1	1	0	1	3	0	2	-0.43231	-0.31286
capacity.sql	10	0	0	10	14	0	2	-0.04462	-0.4876
check_buff.sql	7	0	0	7	29	0	2	-0.07315	-0.51747
cancel_exist_redt.sql	1	1	0	1	3	0	2	-0.43231	-0.31286
chg_fifo.sql	6	4	1	6	11	0	1	-0.26357	0.65139
chg_loc.sql	0	1	0	0	0	0	0	-0.48341	-0.31232
chk_dbl_deep_swp.sql	6	0	0	7	23	3	2	0.12338	-0.54718
chk_liquid_item.sql	1	0	0	2	4	2	2	-0.23498	-0.52742
conf_wodt.sql	2	4	0	2	3	0	1	-0.42971	0.26317
config.sql	1	0	0	1	2	0	1	-0.42654	-0.50243
etc.sql	6	0	0	7	16	1	2	-0.07713	-0.51437

	NSC	NIUO	NDO	NT	NSCC	NJQ	faults	PC1	PC2
front2rear.sql	8	0	0	8	17	0	2	-0.10003	-0.49833
gct_timpri.sql	13	0	0	13	19	0	2	0.08788	-0.48397
handle_trg.sql	4	0	0	4	4	0	1	-0.30963	-0.49486
ifc_ihdt_create.sql	1	1	0	1	1	0	1	-0.44271	-0.31024
ifc_ihdt_delete.sql	4	0	2	5	7	1	1	-0.22985	0.2609
ifc_ildt_create.sql	4	1	0	4	6	0	1	-0.3102	-0.30661
ifc_itrf_create.sql	6	6	2	6	9	0	2	-0.31336	1.42089
ifc_ohdt_create.sql	1	1	0	1	1	0	2	-0.44271	-0.31024
ifc_ohdt_delete.sql	4	0	2	5	7	1	1	-0.22985	0.2609
ifc_oldt_create.sql	4	1	0	4	6	0	1	-0.3102	-0.30661
info.sql	1	0	0	2	4	2	1	-0.23498	-0.52742
is_asrs.sql	1	0	0	1	2	0	1	-0.42654	-0.50243
item.sql	5	0	0	5	19	0	2	-0.19614	-0.51114
lga900.sql	27	0	0	27	125	0	9	1.13601	-0.57549
lga920.sql	3	0	0	3	7	0	2	-0.32954	-0.50219
lgb010.sql	0	2	0	0	0	0	1	-0.49438	-0.12144
lgb020.sql	2	2	0	2	3	0	1	-0.40778	-0.11859
lgb030.sql	2	2	0	2	3	0	1	-0.40778	-0.11859
lgb050.sql	0	2	0	0	0	0	1	-0.49438	-0.12144
lgb060.sql	1	1	0	1	1	0	1	-0.44271	-0.31024
lgb090.sql	3	0	1	3	4	0	1	-0.36259	-0.11313
lgb091.sql	1	2	1	1	2	0	1	-0.46594	0.26445
lgb092.sql	0	0	1	0	0	0	1	-0.4899	-0.11807
lgb200.sql	1	3	0	1	0	0	1	-0.46984	0.07283
lgb300.sql	2	2	0	2	3	0	1	-0.40778	-0.11859
lgb310.sql	3	2	0	3	5	0	1	-0.36187	-0.11781
lgb330.sql	0	2	0	0	0	0	1	-0.49438	-0.12144
lgb370.sql	1	2	0	1	1	0	1	-0.45368	-0.11936

	NSC	NIUO	NDO	NT	NSCC	NJQ	faults	PC1	PC2
lgb380.sql	2	1	1	2	4	0	2	-0.40906	0.07435
lgb390.sql	5	5	5	5	15	0	2	-0.35908	2.37412
lgb395.sql	3	0	3	3	6	0	1	-0.38711	0.6545
lgb500.sql	25	23	11	25	21	0	3	0.07998	8.18073
lgb600.sql	4	4	0	4	5	0	1	-0.34831	0.26734
lgb601.sql	5	3	0	5	9	0	1	-0.28104	0.07461
lgb603.sql	5	2	0	5	20	0	1	-0.21288	-0.1307
lgb610.sql	15	10	0	15	29	0	1	0.10118	1.41848
lgb610.sql	4	0	0	4	8	0	1	-0.28884	-0.50011
lgb730.sql	5	0	0	5	5	0	1	-0.26893	-0.49278
lgb735.sql	11	3	0	14	47	3	3	0.42685	0.01347
lgb736.sql	9	0	0	11	33	3	2	0.29892	-0.54885
LGB737.sql	3	0	0	3	6	0	1	-0.33474	-0.50088
lgb743.sql	8	1	0	8	48	0	3	0.05016	-0.34812
lgb745.sql	4	0	0	4	8	0	1	-0.28884	-0.50011
lgb750.sql	9	0	0	12	16	6	3	0.47377	-0.56073
lgb751.sql	3	1	0	3	5	0	2	-0.3509	-0.30869
lgb795.sql	4	0	0	4	7	0	1	-0.29403	-0.4988
lgb800.sql	0	2	0	0	0	0	1	-0.49438	-0.12144
lgb810.sql	0	1	0	0	0	0	1	-0.48341	-0.31232
lgb820.sql	3	3	0	3	3	0	1	-0.38324	0.07569
lgb830.sql	2	2	0	2	3	0	1	-0.40778	-0.11859
lgb840.sql	4	2	0	4	4	0	2	-0.33157	-0.1131
lgb850.sql	2	1	0	2	3	0	2	-0.39681	-0.30946
lgb860.sql	17	3	0	17	40	0	3	0.30616	0.0747
lgb861.sql	8	2	0	8	14	0	1	-0.13757	-0.11264
lgb862.sql	1	2	0	2	7	5	2	0.00487	-0.18504
lgb870.sql	6	4	0	10	21	4	2	0.20229	0.21092

	NSC	NIUO	NDO	NT	NSCC	NJQ	faults	PC1	PC2
lgb872.sql	0	1	1	0	0	0	1	-0.50087	0.0728
lgb880.sql	13	3	0	13	31	0	8	0.11736	0.07292
lgb881.sql	2	2	0	3	9	5	3	0.05077	-0.18427
lgb882.sql	10	0	0	10	42	0	6	0.10095	-0.52434
lgb883.sql	10	0	0	11	37	2	3	0.25612	-0.54014
lgb890.sql	3	0	1	3	4	0	1	-0.36259	-0.11313
lgb891.sql	1	2	1	1	2	0	0	-0.46594	0.26445
lgb893.sql	1	3	0	1	2	0	0	-0.45945	0.07021
lgb900.sql	3	1	3	3	7	0	0	-0.39288	0.84406
lgb901.sql	2	2	0	2	4	0	1	-0.40258	-0.1199
lgb910.sql	0	1	1	0	0	0	0	-0.50087	0.0728
lgb920.sql	60	0	0	62	75	2	14	2.24587	-0.41891
lgb921.sql	24	6	0	26	50	4	4	0.93609	0.61325
lgb922.sql	9	0	0	9	9	0	3	-0.10612	-0.48444
lgb925.sql	22	0	0	22	28	0	8	0.4542	-0.46521
lgb928.sql	46	11	0	48	101	2	5	1.76334	1.59908
lgb940.sql	28	1	0	28	100	0	4	1.03057	-0.34841
lgb950.sql	5	2	0	5	9	0	1	-0.27007	-0.11627
lge900.sql	123	0	0	139	506	20	16	8.43901	-0.96541
lgt951.sql	22	1	0	23	37	1	5	0.58912	-0.29669
lgt952.sql	18	0	0	28	81	12	5	1.74289	-0.67751
lgh350.sql	5	0	0	5	5	0	2	-0.26893	-0.49278
lgu210.sql	1	1	1	1	1	0	1	-0.46016	0.07489
lgu220.sql	1	1	0	1	1	0	1	-0.44271	-0.31024
lgu900.sql	21	0	0	25	97	6	11	1.33796	-0.62498
lgu901.sql	27	0	0	29	131	4	7	1.52954	-0.6281
lgu910.sql	23	14	0	23	65	0	4	0.52849	2.16193
lmot_shift_get_by_id.sql	1	0	0	1	4	0	1	-0.41614	-0.50505

Appendix D Descriptive statistics and regression Analysis for SQL metrics from PL/SQL files of warehouse management applications

Descriptive statistics of SQL metrics from PL/SQL files

	Min	Max	Mean	Median	Std dev
NSC	0	123	8.166667	4	14.76134
NIUO	0	23	1.805556	1	3.027779
NDO	0	11	0.444444	0	1.474603
NT	0	139	8.814815	4	16.32277
NSCC	0	506	22.2037	6	54.12995
NJQ	0	20	1.018519	0	2.896925
FAULTS	0	16	2.305556	1	2.606456

Model Summary

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate	Change Statistics				
					R Square Change	F Change	df1	df2	Sig. F Change
1	.872 ^a	.760	.745	1.31538	.760	53.188	6	101	.000

a. Predictors: (Constant), NJQ, NIUO, NDO, NSC, NSCC, NT

Coefficients^a

Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.
		B	Std. Error	Beta		
1	(Constant)	1.171	.162		7.215	.000
	NSC	.742	.256	4.202	2.902	.005
	NIUO	-.129	.053	-.149	-2.438	.016
	NDO	-.082	.109	-.046	-.754	.453
	NT	-.548	.255	-3.430	-2.151	.034
	NSCC	-.011	.008	-.230	-1.454	.149
	NJQ	.409	.160	.455	2.563	.012

a. Dependent Variable: FAULTS

Appendix E Regression analysis for N-tier applications

Business Tier

Experimental result for number of fault prediction

Model Summary

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate	Change Statistics				
					R Square Change	F Change	df1	df2	Sig. F Change
1	.604 ^a	.364	.325	1.70348	.364	9.283	10	162	.000

a. Predictors: (Constant), AMC, RFC, NOC, LCOM, CBO, DIT, NA, WMC, NOP, NMA

Coefficients^a

Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.
		B	Std. Error	Beta		
1	(Constant)	-1.148	.596		-1.926	.056
	LCOM	-.001	.004	-.011	-.165	.869
	DIT	-.795	.986	-.156	-.806	.421
	NOP	1.415	1.082	.252	1.308	.193
	CBO	.105	.069	.112	1.532	.128
	NOC	.322	.246	.088	1.307	.193
	RFC	.075	.067	.207	1.112	.268
	NMA	.566	.617	1.489	.917	.361
	NA	.187	.036	.410	5.212	.000
	WMC	-.551	.581	-1.452	-.948	.344
	AMC	.126	.127	.073	.966	.326

a. Dependent Variable: DEFECTS

Experimental result for number of lines prediction

Model Summary

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate	Change Statistics				
					R Square Change	F Change	df1	df2	Sig. F Change
1	.599 ^a	.359	.319	17.08976	.359	9.060	10	162	.000

a. Predictors: (Constant), AMC, RFC, NOC, LCOM, CBO, DIT, NA, WMC, NOP, NMA

Coefficients^a

Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.
		B	Std. Error	Beta		
1	(Constant)	-2.470	5.978		-.413	.680
	LCOM	-.025	.045	-.038	-.557	.578
	DIT	-4.330	9.890	-.085	-.438	.662
	NOP	-.544	10.859	-.010	-.050	.960
	CBO	.148	.689	.016	.215	.830
	NOC	-.447	2.470	-.012	-.181	.857
	RFC	.072	.673	.020	.108	.914
	NMA	12.482	6.192	3.288	2.016	.045
	NA	1.828	.361	.401	5.068	.000
	WMC	-11.670	5.826	-3.081	-2.003	.047
	AMC	1.079	1.277	.063	.844	.400

a. Dependent Variable: LINES

Experimental result for number of required time (in minutes) prediction

Model Summary

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate	Change Statistics				
					R Square Change	F Change	df1	df2	Sig. F Change
1	.470 ^a	.221	.173	45.20415	.221	4.591	10	162	.000

a. Predictors: (Constant), AMC, RFC, NOC, LCOM, CBO, DIT, NA, WMC, NOP, NMA

Coefficients^a

Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.
		B	Std. Error	Beta		
1	(Constant)	-12.917	15.812		-.817	.415
	LCOM	-.206	.119	-.130	-1.722	.087
	DIT	-34.001	26.160	-.279	-1.300	.196
	NOP	48.030	28.722	.357	1.672	.096
	CBO	.768	1.823	.034	.421	.674
	NOC	10.178	6.534	.116	1.558	.121
	RFC	1.643	1.781	.190	.922	.358
	NMA	-7.660	16.378	-.841	-.468	.641
	NA	3.871	.954	.353	4.057	.000
	WMC	7.331	15.410	.806	.476	.635
	AMC	5.985	3.379	.146	1.771	.078

a. Dependent Variable: MINUTES

Data Access Tier

Experiment 1

Experimental result for number of fault prediction

Model Summary

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate	Change Statistics				
					R Square Change	F Change	df1	df2	Sig. F Change
1	.635 ^a	.403	.365	.68047	.403	10.789	6	96	.000

a. Predictors: (Constant), TNGB, TNSQ, NDO, ANSC, NSC, NIUO

Coefficients^a

Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.
		B	Std. Error	Beta		
1	(Constant)	.131	.141		.930	.355
	NSC	.042	.007	.522	5.732	.000
	NIUO	.032	.032	.094	1.004	.318
	NDO	.069	.080	.069	.857	.393
	ANSC	-.098	.054	-.151	-1.809	.074
	TNSQ	-.166	.121	-.114	-1.372	.173
	TNGB	.164	.075	.191	2.191	.031

a. Dependent Variable: DEFECTS

Experiment 2

Experimental result for number of fault prediction

Model Summary

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate	Change Statistics				
					R Square Change	F Change	df1	df2	Sig. F Change
1	.524 ^a	.274	.184	2.62126	.274	3.026	4	32	.032

a. Predictors: (Constant), ANSC, NDO, TNSC, TNIUO

Coefficients^a

Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.
		B	Std. Error	Beta		
1	(Constant)	1.776	.858		2.071	.047
	TNSC	.067	.165	.069	.407	.687
	TNIUO	-1.500	.676	-.583	-2.218	.034
	NDO	1.536	1.305	.198	1.177	.248
	ANSC	3.261	1.581	.610	2.063	.047

a. Dependent Variable: DEFECTS

Experimental result for number of lines prediction

Model Summary

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate	Change Statistics				
					R Square Change	F Change	df1	df2	Sig. F Change
1	.425 ^a	.180	.078	49.02281	.180	1.759	4	32	.162

a. Predictors: (Constant), ANSC, NDO, TNSC, TNIUO

Coefficients^a

Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.
		B	Std. Error	Beta		
1	(Constant)	22.885	16.039		1.427	.163
	TNSC	1.247	3.086	.073	.404	.689
	TNIUO	-25.916	12.650	-.572	-2.049	.049
	NDO	-2.880	24.409	-.021	-.118	.907
	ANSC	61.019	29.563	.649	2.064	.047

a. Dependent Variable: LINES

Experimental result for number of required time (in minutes) prediction

Model Summary

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate	Change Statistics				
					R Square Change	F Change	df1	df2	Sig. F Change
1	.396 ^a	.157	.052	65.17389	.157	1.489	4	32	.229

a. Predictors: (Constant), ANSC, NDO, TNSC, TNIUO

Coefficients^a

Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.
		B	Std. Error	Beta		
1	(Constant)	46.038	21.323		2.159	.038
	TNSC	.314	4.103	.014	.076	.940
	TNIUO	-33.884	16.818	-.571	-2.015	.052
	NDO	2.948	32.450	.016	.091	.928
	ANSC	75.697	39.302	.614	1.926	.063

a. Dependent Variable: MINUTES

Presentation Logic Tier

Experimental result for number of fault prediction

Model Summary

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate	Change Statistics				
					R Square Change	F Change	df1	df2	Sig. F Change
1	.717 ^a	.514	.496	1.49308	.514	29.097	2	55	.000

a. Predictors: (Constant), NUIO, TNM

Coefficients^a

Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.
		B	Std. Error	Beta		
1	(Constant)	-1.565	.454		-3.444	.001
	TNM	.236	.047	.475	4.980	.000
	NUIO	.225	.047	.461	4.826	.000

a. Dependent Variable: DEFECTS

Experimental result for number of lines prediction

Model Summary

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate	Change Statistics				
					R Square Change	F Change	df1	df2	Sig. F Change
1	.665 ^a	.443	.423	18.40332	.443	21.857	2	55	.000

a. Predictors: (Constant), NUIO, TNM

Coefficients^a

Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.
		B	Std. Error	Beta		
1	(Constant)	-21.913	5.600		-3.913	.000
	TNM	2.656	.583	.466	4.555	.000
	NUIO	2.255	.574	.402	3.929	.000

a. Dependent Variable: LINES

Experimental result for number of required time (in minutes) prediction

Model Summary

Model	R	R Square	Adjusted R Square	Std. Error of the Estimate	Change Statistics				
					R Square Change	F Change	df1	df2	Sig. F Change
1	.447 ^a	.200	.171	51.79409	.200	6.874	2	55	.002

a. Predictors: (Constant), NUIO, TNM

Coefficients^a

Model		Unstandardized Coefficients		Standardized Coefficients	t	Sig.
		B	Std. Error	Beta		
1	(Constant)	-18.575	15.761		-1.178	.244
	TNM	3.612	1.641	.270	2.201	.032
	NUIO	4.130	1.615	.313	2.557	.013

a. Dependent Variable: MINUTES

Appendix F Correlations between faults, required time (in minutes) and lines

(1) Presentation Tier

Correlations

		faults	minutes	lines
faults	Pearson Correlation	1	.709**	.802**
	Sig. (2-tailed)	.	.000	.000
	N	58	58	58
minutes	Pearson Correlation	.709**	1	.483**
	Sig. (2-tailed)	.000	.	.000
	N	58	58	58
lines	Pearson Correlation	.802**	.483**	1
	Sig. (2-tailed)	.000	.000	.
	N	58	58	58

** . Correlation is significant at the 0.01 level (2-tailed).

(2) Data Access Tier

Correlations

		faultsDA	linesDA	minutesDA
faultsDA	Pearson Correlation	1	.893**	.783**
	Sig. (2-tailed)	.	.000	.000
	N	37	37	37
linesDA	Pearson Correlation	.893**	1	.654**
	Sig. (2-tailed)	.000	.	.000
	N	37	37	37
minutesDA	Pearson Correlation	.783**	.654**	1
	Sig. (2-tailed)	.000	.000	.
	N	37	37	37

** . Correlation is significant at the 0.01 level (2-tailed).

(3) Business Tier

Correlations

		faultsB	minutesB	linesB
faultsB	Pearson Correlation	1	.735**	.706**
	Sig. (2-tailed)	.	.000	.000
	N	178	178	178
minutesB	Pearson Correlation	.735**	1	.564**
	Sig. (2-tailed)	.000	.	.000
	N	178	178	178
linesB	Pearson Correlation	.706**	.564**	1
	Sig. (2-tailed)	.000	.000	.
	N	178	178	178

** . Correlation is significant at the 0.01 level (2-tailed).

Appendix G Statistical measures

(1) Coefficient of correlation (r)

Coefficient of correlation (r) (Pearson's Linear Correlation Coefficient) is a statistical measure of the strength of the relationship between the actual versus predicted outputs.

$$r = \frac{s_{xy}}{\sqrt{s_{xx} \cdot s_{yy}}}$$

$$s_{xx} = \sum x^2 - \frac{1}{n}(\sum x)^2$$

$$s_{yy} = \sum y^2 - \frac{1}{n}(\sum y)^2$$

$$s_{xy} = \sum xy - \frac{1}{n}(\sum x)(\sum y)$$

where n equals the number of patterns, x refers to the set of actual outputs, and y refers to the predicted output.

(2) Coefficient of multiple determination (R^2)

Coefficient of multiple determination (R^2) is a statistical indicator. It compares the accuracy of the model to the accuracy of a trivial benchmark model wherein the prediction is just the mean of all of the samples.

$$R^2 = 1 - \frac{\sum (y - \hat{y})^2}{\sum (y - \bar{y})^2}$$

where y is the actual value for the dependent variable, $\hat{y}$ is the predicted value of y and $\bar{y}$ is the mean of the y values.

(3) The Adjusted R square

The adjusted R square measures the proportion of the variation in the dependent variable accounted for by the explanatory variables.

$$\text{Adjusted } R^2 = 1 - (1 - R^2) \frac{(n-1)}{(n-k-1)}$$

where: R^2 is the coefficient of multiple determination, n is the number of patterns and k is the number of independent variables.