Mississippi State University
Scholars Junction

Theses and Dissertations Theses and Dissertations

5-8-2004

An empirical evaluation of information theory-based software metrics in comparison to counting-based metrics: case-study approach

Rajiv Govindarajan

Follow this and additional works at: https://scholarsjunction.msstate.edu/td

Recommended Citation
Govindarajan, Rajiv, "An empirical evaluation of information theory-based software metrics in comparison to counting-based metrics: case-study approach" (2004). *Theses and Dissertations*. 507.
https://scholarsjunction.msstate.edu/td/507

This Graduate Thesis - Open Access is brought to you for free and open access by the Theses and Dissertations at Scholars Junction. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of Scholars Junction. For more information, please contact scholcomm@msstate.libanswers.com.

AN EMPIRICAL EVALUATION OF INFORMATION THEORY-BASED SOFTWARE METRICS IN COMPARISON TO COUNTING-BASED METRICS: A CASE-STUDY APPROACH

By

Rajiv Govindarajan

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

Mississippi State, Mississippi

May 2004

Copyright by

Rajiv Govindarajan

2004

AN EMPIRICAL EVALUATION OF INFORMATION THEORY-BASED SOFTWARE

METRICS IN COMPARISON TO COUNTING-BASED METRICS:

A CASE-STUDY APPROACH

By

Rajiv Govindarajan

Approved:

Edward B. Allen
Assistant Professor of Computer Science and Engineering
(Major Professor)

David A. Dampier
Assistant Professor of Computer Science and Engineering
(Committee Member)

Thomas Philip
Professor of Computer Science and Engineering
(Committee Member)

Susan M. Bridges
Professor of Computer Science and Engineering
Graduate Coordinator
Department of Computer Science and Engineering

A. Wayne Bennett
Dean of the Bagley College of Engineering

Name: Rajiv Govindarajan

Date of Degree: May 8, 2004

Institution: Mississippi State University

Major Field: Computer Science

Major Professor: Dr. Edward B. Allen

Title of Study: AN EMPIRICAL EVALUATION OF INFORMATION THEORY-BASED SOFTWARE METRICS IN COMPARISON TO COUNTING-BASED METRICS: A CASE-STUDY APPROACH

Pages in Study: 98

Candidate for Degree of Master of Science

The field of software engineering embraces measurement, analysis and modeling of software. Software metrics are often based on counting, whereas this thesis adopts information theory. The goal of this research is to show that information theory-based metrics proposed by Allen can be useful for software development projects compared to counting-based metrics. Briand, et al. have defined five families of measures based on counting the elements of a graph. This research considers a hypergraph system. Parallel Mathematical Library Project (PMLP) was used as the case study. Abstract semantic graphs were generated for the C++ source files of PMLP in the form of nodes $\times$ hyperedges tables, which are measured for counting and information theory-based measures. Analysis showed that information theory-based metrics provide fine-grained distinctions among the modules,

compared to the counting-based metrics. The case study measurements conformed to the properties proposed by Briand et al. as well.

DEDICATION

To My Family, Friends and God.

ACKNOWLEDGMENTS

This work was supported in part by grant CCR-0098024 from the National Science Foundation. The findings and opinions in this thesis belong solely to the author and are not necessarily those of the sponsor.

Datrix® is a registered trademark of Bell Canada. I would like to thank Bell Canada for providing the license for the use of their product Datrix.

I thank Dr. Edward B. Allen for providing me with useful ideas, his support, and for directing me in the right channel in my approach to research. I would like to thank my researchmate Sampath Gottipati for his valuble suggestion on the research and for developing the Measurement tool. I would like to thank Shane Fox and Shea Fox for evaluating the CPPX tool and preparing tools to be compatible with it. I would also like to thank Archana Chilukuri for providing valuable details about the case study. Last but not least, I would like to thank all the members of the Empirical Software Engineering Research Group for their comments and suggestion on the research.

TABLE OF CONTENTS

Page

DEDICATION ii

ACKNOWLEDGMENTS iii

LIST OF TABLES vii

LIST OF FIGURES ix

LIST OF SYMBOLS AND ABBREVIATIONS x

CHAPTER

I. INTRODUCTION 1

1.1 Hypothesis 3
1.2 Research Questions 4
1.3 Relevance 5
1.4 Overview 6

II. RELATED WORK 7

2.1 Measurement Framework 7
2.2 Information Theory-Based Software Metrics 10
2.3 Validation Framework 11

III. DEFINITIONS OF METRICS 14

3.1 Definitions Based on Information Theory 17
3.1.1 Information Size of a System 21
3.1.2 Information Size of a Module 22
3.1.3 Information Complexity of a System 22
3.1.4 Information Complexity of a Module 22
3.1.5 Information Coupling of a Modular System 23
3.1.6 Information Coupling of a Module 23
3.1.7 A Complete Modular System 23

CHAPTER Page

3.1.8 Information Cohesion of a Modular System 23
3.1.9 Information Cohesion of a Module 24
3.2 Definitions Based on Counting 24
3.2.1 Counting Size of a System 24
3.2.2 Counting Size of a Module 25
3.2.3 Counting Complexity of a System 25
3.2.4 Counting Complexity of a Module 25
3.2.5 Counting Coupling of a Modular System 25
3.2.6 Counting Coupling of a Module 26
3.2.7 Counting Cohesion of a Modular System 26
3.2.8 Counting Cohesion of a Module 26
3.3 Revised Definition of Counting Cohesion 27
3.3.1 Counting Cohesion of a System 27
3.3.2 Counting Cohesion of a Module 28

IV. ANALYSIS OF NEGATIVE COMPLEXITY 29

4.1 Gottipati's Conjecture 29
4.2 Analysis of Negative Complexity: A Revised Conjecture 30

V. TOOLS 37

5.1 Datrix 39
5.2 CPPX 40
5.3 Reorganizer 40
5.4 Abstraction Extractor 41
5.5 Duplicates-System Filter 41
5.6 Nodes $\times$ Hyperedges Generator 42
5.7 Measurement 42
5.8 Statistical Analysis System (SAS) 42
5.9 Microsoft Excel 43

VI. METHODOLOGY 44

6.1 Methodology 44
6.2 Abstraction Issues 45
6.3 Measurement Issues 46

VII. CASE STUDY 48

CHAPTER Page

7.1 Solvers . . . 51
7.2 Graph . . . 52
7.3 SP_BLAS . . . 56

VIII. STATISTICAL ANALYSIS . . . 64

8.1 Preliminary Data Analysis . . . 64
8.2 Correlation Analysis . . . 68
8.3 Comparison of Metrics for Suitability in Software Quality Modeling . . . 77
8.4 Insight into Software Development Processes . . . 80
8.4.1 Analysis of Zero Cohesion . . . 80
8.4.2 Analysis of High Correlation between Size Metrics . . . 82

IX. ANALYSIS OF RESEARCH QUESTIONS . . . 83

9.1 Research Questions . . . 83
9.2 Threats to Validity . . . 88
9.2.1 Threats to Internal Validity . . . 89
9.2.2 Threats to External Validity . . . 89

X. CONCLUSIONS . . . 91

10.1 Evaluation of the Hypothesis . . . 91
10.2 Contribution . . . 94
10.3 Future Work . . . 95

REFERENCES . . . 96

LIST OF TABLES

TABLE Page

1.1 Alternative Software Metrics 3

3.1 An Example Nodes × Hyperedges Table 15

3.2 Properties of Size of a System 16

3.3 Properties of Size of a Module 16

3.4 Properties of Complexity of a System 18

3.5 Properties of Complexity of a Module 18

3.6 Properties of Coupling of a System 19

3.7 Properties of Coupling of a Module 20

3.8 Properties of Cohesion of a System 20

3.9 Properties of Cohesion of a Module 21

4.1 Counter Example 30

4.2 Two Modules with Variable Number of Nodes 31

4.3 Three Modules with Variable Number of Nodes 32

4.4 A System with Greater than Two Nodes 34

5.1 Tools 37

7.1 PMLP Directories for a System 49

7.2 System Measurements of Information Theory-Based Metrics 50

TABLE Page

7.3 System Measurements of Counting-Based Metrics 51

7.4 Sample Nodes × Hyperedges Table for Solvers 53

7.5 Module Level Information Measurements for Solvers 54

7.6 Module Level Counting Measurements for Solvers 55

7.7 Sample Nodes × Hyperedges Table for Graph 57

7.8 Module Level Information Measurements for Graph 58

7.9 Module Level Counting Measurements for Graph 58

7.10 Sample Nodes × Hyperedges Table for SP_BLAS 59

7.11 Module Level Information Measurements for SP_BLAS 60

7.12 Module Level Counting Measurements for SP_BLAS 62

8.1 Summary Statistics for Graph 65

8.2 Summary Statistics for Solvers 66

8.3 Summary Statistics for SP_BLAS 67

8.4 Correlation among Solvers Module Measurements 72

8.5 Correlation among SP_BLAS Module Measurements 73

8.6 Analysis of Information Size and Counting Size at Module Level for Solvers . 75

8.7 Analysis of Information Size and Counting Size at Module Level for Graph . 76

LIST OF FIGURES

FIGURE Page

5.1 Tools Architecture 38

8.1 Information Module Size *vs.* Counting Module Size of Solvers 69

8.2 Information Module Complexity *vs.* Counting Module Complexity of Solvers 69

8.3 Information Module Coupling *vs.* Counting Module Coupling of Solvers . . . 70

8.4 Information Module Size *vs.* Counting Module Size of SP_BLAS 70

8.5 Information Module Complexity *vs.* Counting Module Complexity of SP_BLAS 71

8.6 Information Module Coupling *vs.* Counting Module Coupling of SP_BLAS . 71

8.7 Comparison of Information *vs.* Counting System Size 78

8.8 Comparison of Information *vs.* Counting System Complexity 78

8.9 Comparison of Information *vs.* Counting System Coupling 79

LIST OF SYMBOLS AND ABBREVIATIONS

ASG Abstract Semantic Graph

CPPX C/C++ Fact Extractor

H Entropy of a probability distribution

i, j Indexes for row in S, $i = 0, ..., n$ and similarly j

k Index for a module in **S**, $k = 1, ..., n_M$

l Index for a pattern of values on a row

$L(i)$ A function that determines the label of a row

$L_i(j)$ A function that determines the label of a row j in S_i

log Logarithm, base 2

m_k module k. Nodes in a module and their incident hyperedges

MS Modular system. **S** partitioned into modules

$MS^{(n)}$ Complete graph. Complete graph with n nodes in one module

$MS^{\star}$ Intermodule hyperedges graph. Nodes in *MS* and intermodule hyper-edges

MS° Intramodule hyperedges graph. Nodes in *MS* and intramodule hyper-edges

n The number of nodes in system, **S**

n_e Number of hyperedges in system, **S**

n_{e_k} Number of hyperedges incident to nodes in module m_k

n_{ec} Number of hyperedges in a complete graph of a system, **S**.

n_{eck} Number of hyperedges in a complete graph of module, m_k

n_{intra_e} Number of intramodule hyperedges in system, **S**

n_{inter_e} Number of intermodule hyperedges in system, **S**

$n_{inter_e_k}$ Number of intermodule hyperedges incident to module, m_k

n_k Number of nodes in the module, m_k

n_M The number of modules in *MS*

p Probability mass function

S System. Abstraction of software (nodes and hyperedges)

S System graph. **S** plus environment node, represented by nodes $\times$ hyperedges table

$\mathbf{S}^{\#}$ Hyperedges-only graph. Hyperedges in **S** and end points

SAS Statistical Analysis System

$\mathbf{S}_i$ Node subgraph. Nodes in $\mathbf{S}^{\#}$ and hyperedges incident to node i

CHAPTER I

INTRODUCTION

High quality is an essential attribute for software systems. For example, high quality software products are often nearly defect-free. The cost of detecting faults at a later phase of the software development life cycle is very high.

The development of software quality models can point out the potential risk areas at an early phase [30]. Examples of software quality modeling techniques include regression and classification techniques. Software quality models can predict which modules can be of high risk. Software quality models have product metrics and process metrics that can be predictors of quality. Khoshgoftaar et al. [20] distinguish the product metrics as measures of product attributes and the process metrics as measures of process attributes in the software development process. Most of the measures are based on counting, whereas this research adopts information theory as a basis for the measures.

Design decisions at various levels of abstractions are information. Information metrics as well as counting metrics have been shown formally to be proper [2]. The aim of this research is to provide practical evidence to show the utility of the metrics [1]. If one collects metrics regularly then tracking the project progress and checking whether the product has reached the desired quality can be predicted.

The essence of software design is identification of components, relationships among the components, and relationships among the elements within the components. Graphs are composed of nodes and edges, which are an attractive abstraction for depicting the relationships among the objects and are widely used in depicting abstractions of the software [1]. Briand, Morasca, and Basili [12] have defined five families of measures through distinct sets of properties in a formal framework based on graphs, namely size, length, complexity, coupling, and cohesion. These measures are applicable to any abstraction of software using graphs.

Allen and Khoshgoftaar [4] have proposed information theory-based metrics for coupling and cohesion of graphs at module level. Allen [3] proposes additional measures of size, length, complexity, and also updates measures for coupling and cohesion. Table 1.1 lists alternative definitions of these measures in information theory terminology and shows how the definitions differ from the counting-based measures. Gottipati [17] uses hyperedges instead of ordinary edges to represent the connection between nodes in a hypergraph system. In this hyperedges are used to represent the connection between nodes in a hypergraph system.

This research uses a case study approach. Parallel Mathematical Library Project (PMLP) is the subject of the case study. PMLP consists of scalable libraries that combine the features of object-oriented design, and sequential and parallel modes. An empirical study was conducted on PMLP to get the desired information theory-based measurements. The information theory-based measurements were then compared with measurements ob-

Table 1.1 Alternative Software Metrics

Family	Information theory-based metric	Counting-based metric
Size	Information in graph	Number of nodes
Length	Information in path	Number of nodes in path
Complexity	Information in relationships	Number of edges
Coupling	Information in intermodule relationships	Number of intermodule edges
Cohesion	Information in the intramodule relationships divided by maximum possible	Number of intramodule edges divided by the total number of edges in a complete graph

tained from counting-based metrics to show that information theory-based measurements have advantages over the traditional counting-based measurements. Analysis of the case study provided insights into the measures based on the information theory. The research showed empirical evidence that information theory-based software metrics have potential as useful predictors of software quality [1].

1.1 Hypothesis

Allen [3] based his study on graphs and subgraphs that may represent the software module. These graphs represent components and their relationships. Allen [3] proposes information theory-based measures at the system and module levels for application to the software. This thesis is an extended study and analysis of the information theory-based measures proposed by Allen [3]. The goal of this research is to provide evidence

that information theory-based metrics have practical advantages over the counting based metrics [17]. The hypothesis of the research is :

> Information theory-based metrics proposed by Allen [3] (size, complexity, coupling, and cohesion) can be useful for large codebases in real-world software development projects, compared to corresponding counting-based metrics [17].

The research takes into consideration several aspects of usefulness, including suitability for statistical software quality modeling, insight into development processes, and conformance of the metrics to intuition about the attributes.

1.2 Research Questions

A case study of the Parallel Mathematical Library Project (PMLP) was conducted. PMLP is a real-world software development project. The following are the research questions that provided evidence for or against the hypothesis in the context of this case study.

1. What are the similarities and differences between the distributions of information theory-based metrics and counting-based metrics?
   (a) Which metrics are more suitable for statistical software quality modeling?
   (b) Do the distributions of measurement values yield insight into the software development process and resulting product attributes?
   (c) Are the information theory and counting metrics highly correlated to the traditional software metrics obtained from the Datrix?
   (d) Are the ranges of distributions of metrics suitable for software quality modeling?
2. Does each information theory-based metric and counting-based metric preserve our intuition about the attribute it purports to measure?
   (a) Does each of the metrics conform to the properties proposed by Briand, Morasca, and Basili [12]?

(b) Is Gottipati's [17] conjecture true regarding conditions when information module complexity is negative?

(c) What is a definition of counting cohesion for a hypergraph system that will satisfy the properties of Briand, Morasca, and Basili [12, 17]?

(d) What are the measures of counting complexity and counting coupling that correspond to the revised definition of the counting cohesion?

High correlation between the independent variables is not good for software quality modeling. For example, counting and information theory metrics should not be independent variables in the same model if they are highly correlated. If the range of the distribution of metric is large or narrow, then it may not be useful for modeling. We consider intuition about the attributes to be defined as conformance to the metrics properties proposed by Briand, Morasca, and Basili [12].

1.3 Relevance

The goal of software engineering is to identify the faults in the software at an early stage, rectify them, and produce software that is defect-free. The cost of detecting and rectifying errors as the development goes through the life cycle is very high. This cost can be lowered by identifying the potential problem areas or modules early in development. Software quality models aid in making improvements to the identified problem areas. They predict the problems so that the risks can be dealt with before they turn into a quality problem. The proposed research empirically investigates new variables for models to improve their robustness and accuracy and, indirectly, software quality.

1.4 Overview

In Chapter II, relevant concepts and views of various researchers that support this research are discussed. Chapter III introduces definitions of both information theory-based and counting-based metrics. The following chapters discuss in detail the tools used, measurement, and statistical analysis performed on the case study. Chapters IX and X answer the research questions identified, form conclusions of this research and summarize contributions to the research community.

CHAPTER II

RELATED WORK

This chapter introduces concepts that are the foundation for this research. This chapter also talks about a common framework within which several metrics were defined.

2.1 Measurement Framework

The literature proposes many measures to capture the structural quality of the design. Such measures aim at providing ways to assess the quality of the software. Briand, El Emam, and Morasca [11] suggest a pragmatic approach to the application of measurement theory rather than sticking rigidly to the mathematical and schematic viewpoint. Measurement theory defines a framework within which the measures are defined. Briand, El Emam, and Morasca [11] review and discuss the current state of measurement theory and its uses, provide a balanced view on the issues that exist, and present a pragmatic approach to application.

Several attributes (size, length, complexity, coupling, and cohesion) are measured by a large number of metrics in the literature. These metrics are often ambiguously defined, making the adequacy of the measures unclear. Briand, Morasca, and Basili [12] propose a mathematical framework that is generic. They use the mathematical framework to de-

fine several important measurement concepts. Morasca and Briand [17, 24] generalize the framework by considering n-ary relationships between elements and by proposing a hierarchical axiomatic framework where each level in the hierarchy is mapped to the level of measurement.

Briand, Morasca, and Basili [12] have proposed five families of measures, namely, size, length, complexity, coupling, and cohesion, in a formal framework of distinctive properties based on graphs composed of nodes and edges. These measures were defined at both system and module level. A subgraph represents a software module. Their example measures are based on counting. These measures can be used as a guide to choose among the alternative techniques or artifacts. All these measures are related to the internal software attributes. Briand, Morasca, and Basili provide properties for a partial set of measurement concepts that are required for the definition of internal software attributes. Briand, Morasca, and Basili [12] focus on the properties that differentiate the different concepts, namely, size, length, complexity, coupling, and cohesion. These measures can address artifacts other than code produced in the software process. The authors point out the importance of the earlier phases of the software process upon which the rest of the development depends. The paper discusses size, length, and complexity in general, and coupling and cohesion in relation to a modular system. In comments on Briand, Morasca, and Basili's [12] measurement theory, Zuse [31], emphasizes the importance of the additivity property and extensive structure to show the empirical properties behind the software measures.

Poels and Dedene [27] comment on several inconsistencies related to additivity properties in the mathematical framework proposed by Briand, Morasca, and Basili [12]. Poels and Dedene [27] suggest ways to remove the ambiguity in the property definitions. The authors point out the importance of identifying and removing inconsistencies so that the axiomatic framework can be better understood. The removal of inconsistencies will allow a better understanding and refinement of the axiomatic framework proposed [12]. In their comments, they explored the additivity property, which is an important property of metrics, and introduced a new union operator for modules. The inconsistencies in the definitions are removed by introducing an ordinal scale of connection strength between modules and by relating the scale to the additivity properties. Their idea is supported by the discrimination between the modules that are disjointed. Briand, Morasca, and Basili [13] acknowledged the comments and improvements suggested by Poels and Dedene [27].

Briand et al. [7] have aimed at empirically exploring the relationships that exist among coupling, cohesion and the probability of faults in the system level classes. Empirical exploration gives a better understanding of relationships between measures. The results of the study showed that many of the measures capture similar dimensions in data sets, thus reflecting the fact that many of the measures are based on similar attributes. Their case study found that the frequency of method invocations and the depth of inheritance were the driving forces for fault-proneness. The proposed thesis will follow a case study approach that aims at understanding information theory measures and showing their advantages over the traditional counting metrics.

2.2 Information Theory-Based Software Metrics

Information theory-based metrics share similar principles and concepts with counting-based metrics, but they are defined from an information theory perspective. That is, the basic measures of size, length, complexity, coupling, and cohesion differ in the way they are calculated. Table 1.1 lists informal definitions of counting-based and information theory-based metrics. Traditional software metrics count the artifact features, which are difficult for the developers to remember [3]. A graph composed of nodes and hyperedges may be an abstraction of the software system, and the subgraph represents a software module [3]. Allen [3] suggests that repetitive patterns are easier to remember than a comparable set of distinctive items because of very low information content. In contrast to counting, this research adopts information theory as the basis for measurement because the design decisions embodied by a graph abstraction of the software are information. Because the measurement algorithms measure graphs, they can be used to measure many software abstractions,laying down a foundation for the empirical research.

Allen [3] extended this line of research by empirically exploring the relationships between nodes and hyperedges in a graph system. Allen discusses the information theory-based measures on graphs at the system level as well as module level for the five families of metrics defined by Briand, Morasca, and Basili [12].

2.3 Validation Framework

Kitchenham, Pfleeger, and Fenton [22] propose a framework for validating software measurement. They define a measurement structure model that identifies the elementary component of measures and the measurement process and then consider five other models involved in measurement. Potential problems in measures are identified. Kitchenham, Pfleeger, and Fenton try to identify these problems by proposing a validation framework that helps the researchers and practitioners get an unambiguous understanding of how to validate a measure, the way in which the work of others can be validated, and when to apply these measures in a given situation. However, Morasca et al. [25] note misinterpretation of Weyuker's properties by Kitchenham, Pfleeger, and Fenton [22]. They also criticize Kitchenham, Pfleeger, and Fenton's [22] argument on properties used to define the measures.

In response to the validation framework proposed by Kitchenham, Pfleeger, and Fenton [22], Briand, Morasca, and Basili [14] defined ways to validate measures for a high-level design. The authors suggest the importance of metrics in the early phases of software development. Availability of metrics allow a better management of the later phases and more effective quality assessment. Several high-level designs are introduced and compared. The measures are derived based on an experimental goal, identification of fault-prone software parts, and several experimental hypotheses arising from development. Then, coupling and cohesion measures that satisfy the mathematical framework are defined. For a measure to be valid, it should satisfy the previously published mathematical framework. After the

completion of the previous steps, the relationship between the fault-proneness and the measures are investigated to provide empirical evidence of the validity of the measure.

Zelkowitz and Wallace [30] stress the importance of using the experimental models for validating the metrics. The authors suggest experimentation as a crucial part of the evaluation of new metrics. Experimentation can help in determining the compliance of new metrics to proposed theories, resulting in software being effective. However, there is need for a concise taxonomy for demonstrating the validity of metrics. The authors categorize experiments into four method groups: scientific methods, engineering methods, empirical methods, and analytical methods. This thesis adopts an empirical study and uses a case study for validating the metrics. Validation models mainly deal with data collection and experimenting with or analyzing the data collected. The information gathered from observation, historic data, and controlled environment are briefly analyzed. The results of the analysis are useful in determining the validity of the new technique. In this research the case study was observed, and information and counting metrics were calculated in a controlled environment.

Schneidewind [29] presents a comprehensive empirical validation methodology that consists of six validation criteria that support the quality functions of assessment, control, and prediction. Empirically validated measures can be a basis for making decisions and taking actions to improve the quality of the software. The supported quality functions are the activities conducted by the software organization for achieving the defined goals.

Schneidewind [29] shows that the nonparametric statistical methods can play an important role in evaluating metrics against the validity criteria.

The proposed research will build on empirically validating the information theory-based metrics in four of the five metrics families (size, complexity, coupling, and cohesion) proposed by Briand, Morasca, and Basili [12] both at system and module [17].

CHAPTER III

DEFINITIONS OF METRICS

This section deals with defining the information theory and counting-based system level and module level metrics. Before defining the metrics its essential to get an understanding of various entities. A system [2] is an abstraction of a software development artifact, defined by set of elements and relationships on them. The abstraction of the system, **S**, in this research is represented as a hypergraph containing nodes and hyperedges. Each node corresponds to an element and each hyperedge corresponds to a relationship among a set of nodes. This relationship is represented using a nodes $\times$ hyperedges table. The nodes $\times$ hyperedges table consists of four columns for software identifier, module identifier, method identifier, and row pattern. Each row pattern is associated with a node and is called a label. The row patterns represent the nodes $\times$ hyperedges relationship in the form of 1's and 0's. "1" in a row pattern indicates the connection of a hyperedge to a node and 0 corresponds to absence of connection (see Table 3.1). We form a system graph S for calculation of metrics by adding an environment node to the system model. From the system graph we estimate the probability mass function p by the number of occurances of each row pattern divided by the the number of nodes in the system plus the environment node $(n + 1)$.

Table 3.1 An Example Nodes × Hyperedges Table

Module	Node	Hyperedges	$p_{L(i)}$
M0	0	0000000000	$1/14$
M1	1	1000000000	$1/14$
M2	2	1100000000	$1/14$
M2	3	0100100000	$1/14$
M2	4	0010100000	$1/14$
M3	5	1010000000	$1/14$
M3	6	1001000000	$1/14$
M3	7	0010010000	$1/14$
M3	8	0011001000	$1/14$
M3	9	0000001000	$1/14$
M4	10	0000010100	$1/14$
M4	11	0100000110	$1/14$
M4	12	0000000101	$1/14$
M4	13	0000000011	$1/14$

In Chapter II, we have seen Briand, Morasca, and Basili [12] define five families of metrics both at system level and module level of the system [17] (size, length, complexity, coupling, and cohesion). Table 3.2 and Table 3.3 [2] summarize the properties of the size of a system and a module. Satisfying the size properties qualifies a metric as a member of the family of size measures. For example, the number of nodes in a system is a member of the family of size measures and we call it "counting system size".

Table 3.4 and Table 3.5 [2] summarize properties of system and module complexity. The measures in our research make use of the same properties of Briand, Morasca, and Basili except for a slightly stronger constraint in Property 4, namely the "no nodes in common", rather than "no edges in common". For example the number of hyperedges in

Table 3.2 Properties of Size of a System

1. Nonnegativity. The size of the system is nonnegative.

2. Null value. The size of the system is null if its set of nodes is empty.

3. Module additivity. Given a system, **S**, having modules, m_1 and m_2, such that every node in **S** is in m_1 or m_2, but not both, the size of this system is equal to the sum of the sizes of the modules m_1 and m_2.

   $Size(\mathbf{S}) = Size(m_1|\mathbf{S}) + Size(m_2|\mathbf{S})$

Table 3.3 Properties of Size of a Module

1. Nonnegativity. The size of a module is nonnegative.

2. Null value. The size of a module is null if its set of nodes is empty.

3. Module additivity. Given a module, m_k, in a system, **S**, having modules within it, m_1 and m_2, such that every node in m_k is in m_1 or m_2, but not both, the size of this module is equal to the sum of the sizes of the modules m_k and m_k.

   $Size(m_k|\mathbf{S}) = Size(m_1|\mathbf{S}) + Size(m_2|\mathbf{S})$

a system is a member of family of complexity measures and we call it counting system complexity.

Table 3.6 and Table 3.7 [2] summarize the sets of properties that define the concept of coupling at both the system level and module level. The properties in our research differ from the Briand, Morasca, and Basili [12] properties in only one aspect where we make no distinction between inbound and outbound coupling.

Table 3.8 and Table 3.9 [2] summarize the properties essential to define the systemlevel and module level cohesion. We make use of the same framework proposed by Briand, Daly, and Wüst [8] to calculate cohesion. The amount of cohesion is interpreted as the proportion of information in the intramodule edges graph, compared to the "most cohesive graph possible".

3.1 Definitions Based on Information Theory

Suppose we have a system, **S**, with n nodes. Let the system graph be S with an additional disconnected node n_0. We call n_0 the "environment" node. Let us model the system graph S as a set of statistically independent samples from a probability distribution on the row patterns of its nodes $\times$ edges table, $p_l, l = 1, ..., n_s$, where l is the index of the pattern of values on a row, and n_s is the number of possible distinct row patterns. Entropy is a fundamental concept of information theory. Entropy in our case is then calculated as the average information per node [2].

$$H(S) = \sum_{l=1}^{n_s} p_l(-\log p_l) \tag{3.1}$$

Table 3.4 Properties of Complexity of a System

1. Nonnegativity. The complexity of a system is nonnegative.

2. Null value. The complexity of the system is null if its set of hyperedges is empty.

3. Symmetry. The complexity of a system does not depend on the convention chosen to represent the direction of hyperedges.

4. Module monotonicity. Given a System, **S**, with any two modules, m_1 and m_2, that have no nodes in common, the complexity of the system is no less than the sum of the complexities of the two modules.

   $\mathit{Complexity}(\mathbf{S}) \geq \mathit{Complexity}(m_1|\mathbf{S}) + \mathit{Complexity}(m_2|\mathbf{S})$

5. Disjoint module additivity. Given a system, **S**, composed of two disjoint modules, m_1 and m_2, the complexity of the system is equal to the sum of the complexities of the two modules.

   $\mathit{Complexity}(\mathbf{S}) = \mathit{Complexity}(m_1|\mathbf{S}) + \mathit{Complexity}(m_2|\mathbf{S})$

Table 3.5 Properties of Complexity of a Module

1. Nonnegativity. The complexity of a module is nonnegative.

2. Null value. The complexity of a module is null if its set of intermodule and intramodule hyperedges is empty.

3. Monotonicity. Adding a hyperedge to a module does not decrease its complexity.

Table 3.6 Properties of Coupling of a System

1. Nonnegativity. Coupling of a modular system is nonnegative.

2. Null value. Coupling of a modular system is null if its set of intermodule hyperedges is empty.

3. Monotonicity. Adding an intermodule hyperedge to a modular system does not decrease its coupling.

4. Merging of modules. If two modules, m_1 and m_2, are merged to form a new module, $m_{1 \cup 2}$, that replaces m_1 and m_2, then the coupling of the modular system with $m_{1 \cup 2}$ is not greater than the coupling of the modular system with m_1 and m_2.

5. Disjoint module additivity. If two modules, m_1 and m_2, which have no intermodule hyperedges between nodes in m_1 and nodes in m_2, are merged to form a new module, $m_{1 \cup 2}$, that replaces m_1 and m_2, then the coupling of the modular system with $m_{1 \cup 2}$ is equal to the coupling of the modular system with m_1 and m_2.

Table 3.7 Properties of Coupling of a Module

1. Nonnegativity. Coupling of a module is nonnegative.

2. Null value. Coupling of a module is null if its set of intermodule hyperedges is empty.

3. Monotonicity. Adding an intermodule hyperedge to a module does not decrease its module coupling.

4. Merging of modules. If two modules, m_1 and m_2, are merged to form a new module, $m_{1\cup 2}$, that replaces m_1 and m_2, then the module coupling of $m_{1\cup 2}$ is not greater than the sum of the module coupling of m_1 and m_2.

5. Disjoint module additivity. If two modules, m_1 and m_2, which have no intermodule hyperedges between nodes in m_1 and nodes in m_2, are merged to form a new module, $m_{1\cup 2}$, that replaces m_1 and m_2, then the module coupling of $m_{1\cup 2}$ is equal to the sum of the module coupling of m_1 and m_2.

Table 3.8 Properties of Cohesion of a System

1. Nonnegativity and Normalization. Cohesion of a modular system belongs to a specified interval, $\mathit{Cohesion}(\mathit{MS}) \in [0, \mathit{Max}]$.

2. Null value. Cohesion of a modular system is null if its set of intramodule hyperedges is empty.

3. Monotonicity. Adding an intramodule hyperedge to a modular system does not decrease its cohesion.

4. Merging of modules. If two unrelated modules, m_1 and m_2, are merged to form a new module, $m_{1\cup 2}$, that replaces m_1 and m_2, then the cohesion of the modular system with $m_{1\cup 2}$ is not greater than the cohesion of the modular system with m_1 and m_2.

Table 3.9 Properties of Cohesion of a Module

1. Nonnegativity and Normalization. Cohesion of a module belongs to a specified interval, $\mathit{Cohesion}(m_k|\mathit{MS}) \in [0, \mathit{Max}]$.

2. Null value. Cohesion of a module is null if its set of intramodule hyperedges is empty.

3. Monotonicity. Adding an intramodule hyperedge to a module does not decrease its cohesion.

4. Merging of modules. If two unrelated modules, m_1 and m_2, are merged to form a new module, $m_{1\cup 2}$, that replaces m_1 and m_2, then the module cohesion of $m_{1\cup 2}$ is not greater than the maximum of the module cohesion of m_1 and m_2.

This forms the basis for defining other information theory-based measures. The following are the definitions of information theory-based metrics [2].

3.1.1 Information Size of a System

The size of a system **S**, is the amount of information in its system graph, S, less the contribution of the environment [2].

$$\mathit{Size}(\mathbf{S}) = (n+1)H(S) - (-\log p_{L(0)}) \tag{3.2}$$

The estimated information contribution of the environment node is $-\log p_{L(0)}$ [2]. From this the size of the system is estimated by the following.

$$\mathit{Size}(\mathbf{S}) = \sum_{i=1}^{n}(-\log p_{L(i)}) \tag{3.3}$$

It should be noted that summation begins with i = 1 instead of i = 0.

3.1.2 Information Size of a Module

The size of a module, m_k, in a system [2], $\mathbf{S}$, is the information in its system graph contributed by the module.

$$\mathit{Size}(m_k|\mathbf{S}) = \sum_{i \in m_k} (-\log p_{L(i)}) \tag{3.4}$$

3.1.3 Information Complexity of a System

Consider a hyperedges-only graph, $\mathbf{S}^{\#}$, where $\mathbf{S}_i^{\#}$ is a node subgraph. The complexity of a system [2] is given by the amount of information in relationships in its hyperedges-only graph, less the contribution of the environment.

$$\mathit{Complexity}(\mathbf{S}) = \sum_{i=1}^{n} \mathit{Size}(\mathbf{S}_i^{\#}) - \mathit{Size}(\mathbf{S}^{\#}) \tag{3.5}$$

From the above one can see that *Size* does not include the contribution of the environment node. Similarily, estimated *Complexity* can be viewed as the number of nodes times the excess entropy of the system graph [2], less the contribution of the environment.

3.1.4 Information Complexity of a Module

The complexity of a module, m_k, in a system, $\mathbf{S}$, is its contribution to the complexity of the system given by

$$\mathit{Complexity}(m_k|\mathbf{S}) = \sum_{i \in m_k} \mathit{Size}(\mathbf{S}_i^{\#}) - \mathit{Size}(m_k|\mathbf{S}^{\#}) \tag{3.6}$$

3.1.5 Information Coupling of a Modular System

The coupling of a modular system [2], *MS*, is the amount of information in the intermodule relationships in its system graph, less the contribution of the environment.

$$Coupling(MS) = Complexity(MS^{*}) \tag{3.7}$$

where MS^{*} is the intermodule hyperedges graph.

3.1.6 Information Coupling of a Module

The coupling of a module [2], m_k, in a modular system, *MS*, is its contribution to the coupling of the system, given by

$$Coupling(m_k|MS) = Complexity(m_k|MS^{*}) \tag{3.8}$$

3.1.7 A Complete Modular System

Given a modular system, *MS*, with n nodes, define the corresponding complete modular system [2], $MS^{(n)}$, as the modular system consisting of all the nodes in *MS* and all the possible edges (i.e, two end points) between those nodes and let all the nodes be in one module. Let $S^{(n)}$ be the corresponding system graph. Similarly, denote a module that is a complete graph by $m_k^{(n_k)}$.

3.1.8 Information Cohesion of a Modular System

Because the properties in Table 3.9 focus on the intramodule edges, we define cohesion as a measurement on an intramodule-edges graph, MS°. We interpret the amount of

cohesion as a proportion comparing the relational information in a graph, to that of the "most cohesive" graph possible.

The cohesion of a modular system [2], *MS*, with n nodes is the proportion of information in a complete system graph due to intramodule relationships.

$$\mathit{Cohesion}(\mathit{MS}) = \frac{\mathit{Complexity}(\mathit{MS}^{\circ})}{\mathit{Complexity}(\mathit{MS}^{(n)})} \tag{3.9}$$

for $n > 1$. By convention, $\mathit{Cohesion}(\mathit{MS}) = 0$ when $n = 1$.

3.1.9 Information Cohesion of a Module

Cohesion of a module m_k, with n_k nodes, in a modular system *MS* is the proportion of information in intramodule relationships of a complete module $m_k^{(n_k)}$, due to the intramodule relationships of m_k.

$$\mathit{Cohesion}(m_k|\mathit{MS}) = \frac{\mathit{Complexity}(m_k|\mathit{MS}^{\circ})}{\mathit{Complexity}(m_k^{(n_k)}|\mathit{MS}^{\circ})} \tag{3.10}$$

3.2 Definitions Based on Counting

The following sections define each counting-based metrics used in this research [1, 17].

3.2.1 Counting Size of a System

The counting size of the system **S**, *CountingSystemSize*, is given as the number of nodes in **S**.

$$\mathit{CountingSystemSize}(\mathbf{S}) = n \tag{3.11}$$

3.2.2 Counting Size of a Module

The counting size of a module in a system **S**, *CountingModuleSize*, is the number of nodes in the module.

$$\textit{CountingModuleSize}(m_k|\mathbf{S}) = n_k \tag{3.12}$$

3.2.3 Counting Complexity of a System

The counting complexity of a system **S**, *CountingSystemComplexity*, is given as the number of hyperedges in the system.

$$\textit{CountingSystemComplexity}(\mathbf{S}) = n_e \tag{3.13}$$

3.2.4 Counting Complexity of a Module

The counting complexity of a module in a system **S**, *CountingModuleComplexity*, is the number of hyperedges incident to the nodes in the module.

$$\textit{CountingModuleComplexity}(m_k|\mathbf{S}) = n_{e_k} \tag{3.14}$$

3.2.5 Counting Coupling of a Modular System

The counting coupling of a modular system *MS*, *CountingSystemCoupling*, is given as the number of intermodule hyperedges in the system.

$$\textit{CountingSystemCoupling}(\mathbf{S}) = n_{inter_e} \tag{3.15}$$

3.2.6 Counting Coupling of a Module

The counting coupling of a module in a modular System *MS*, *CountingModuleCoupling*, is the number of intermodule hyperedges incident to the module.

$$CountingModuleCoupling(m_k|\mathbf{S}) = n_{inter_e_k} \tag{3.16}$$

3.2.7 Counting Cohesion of a Modular System

The counting cohesion of a system **S**, *CountingSystemCohesion*, is given as the ratio of the number of intramodule hyperedges to the total number of edges (i.e. two end points) in a complete graph of the system.

$$CountingSystemCohesion(\mathbf{S}) = \frac{n_{intra_e}}{n_{ec}} \tag{3.17}$$

3.2.8 Counting Cohesion of a Module

The counting cohesion of a modular system *MS*, *CountingModuleCohesion*, is the ratio of number of intramodule hyperedges in the module to the total number of edges (i.e. two end points) in a complete graph of that module.

$$CountingModuleCohesion(m_k|\mathbf{S}) = \frac{n_{intra_e_k}}{n_{eck}} \tag{3.18}$$

3.3 Revised Definition of Counting Cohesion

The earlier definition [1, 17] of counting cohesion for a system as well as for a module violates the properties proposed by Briand, Morasca, and Basili [12], when considering a complete graph as the "most cohesive graph possible". Cohesion by convention should lie within 0 and 1. But Sampath Gotipatti [17], has shown that some of the calculated values for counting-based cohesion measures, at both system and module levels are greater than 1. In order for counting cohesion to conform to the properties of Briand, Morasca, and Basili [12], we have to change the definition of the cohesion measures. We consider sum of the intermodule and intramodule hyperedges in a hypergraph system as the "most cohesive graph" possible. The research make use of the revised definitions of counting cohesion for a system and a module. The revised definitions for the system level and module level cohesion measures are as follows.

3.3.1 Counting Cohesion of a System

The counting cohesion of a system **S**, *CountingSystemCohesion*, is given as the ratio of the number of intramodule hyperedges to the total number of hyperedges in the graph of the system.

$$\mathit{CountingSystemCohesion}(\mathbf{S}) = \frac{n_{intra_e}}{n_{intra_e} + n_{inter_e}} \tag{3.19}$$

3.3.2 Counting Cohesion of a Module

The counting cohesion of a modular system *MS*, *CountingModuleCohesion*, is the ratio of number of intramodule hyperedges in the module to the total number of hyperedges incident to the subgraph of that system corresponding to the module.

$$\mathit{CountingModuleCohesion}(m_k|\mathbf{S}) = \frac{n_{intra_e_k}}{n_{intra_e_k} + n_{inter_e_k}} \tag{3.20}$$

CHAPTER IV

ANALYSIS OF NEGATIVE COMPLEXITY

According to Briand, Morasca, and Basili [12], complexity should be non-negative (see Table 3.4, and Table 3.5). The condition of negative complexity of a module violates this property. Gottipati [17] found that some row patterns have negative module information complexity. In this chapter, we analyze this phenomenon.

4.1 Gottipati's Conjecture

Measurement of certain row patterns results in negative information complexity. Gottipati explored this condition, and found repeated occurences of negative complexity values when row patterns satisfied a set of conditions.

Conjecture 1 (Gottipati [17]) *Given a system,* $\mathbf{S}$*, composed of at least two modules, we call the module of interest* m_1*. A system that fulfills the conditions below has negative module information complexity, Complexity*$(m_1|\mathbf{S}) < 0$.

1. *The row patterns of all the nodes in module* m_1 *are identical.*
2. *A hyperedge connected to a node in module* m_1 *must also be connected to all the other nodes in the system.*
3. *There must be at least one hyperedge connected to module,* m_1.

4. There must be at least one hyperedge not connected to module, m_1.

However, we found that Gottipati's conjecture [17] (Conjecture 1) is not always true. This can be shown using a counter-example that gives a complexity value greater than zero yet fulfills Gottipati's criteria. Table 4.1 shows an example that satisfies all the conditions specified in the Conjecture 1.

Table 4.1 Counter Example

Module Identifier	Node Identifier	RowPattern
M1	N1	10111
M1	N2	10111
M2	N3	11111

Module M1 has
$\textit{Complexity}(M1|\mathbf{S}) = 0.49 > 0$

4.2 Analysis of Negative Complexity: A Revised Conjecture

The counter-example in Table 4.1 shows Gottipati's conjecture [17] is not always true. This was further confirmed by more counter examples. Further analyzing Gottipati's conjecture [17] with various other conditions to redefine the rules, led to the addition of two new rules that would refine the conditions for negative information complexity of a module.

The condition of negative information complexity was analyzed considering row patterns of different sizes, and by varying the number of nodes and modules. Table 4.2 and Table 4.3 show the results of the analysis by varying the number of nodes and modules. From the tables we see that when modules considered for negative information complexity analysis had fewer number of nodes than the other modules in the system, negative information complexity occured. This condition constrained the rules further. Further analysis to validate the new condition was performed. Most of the measures obtained had negative information complexity when the above condition was true. But there were few cases that resulted in positive information complexity values. These cases were analyzed.

Table 4.2 Two Modules with Variable Number of Nodes

No. of nodes in module under consideration	No. of nodes in the other module	Module Information Complexity(negative/positive)
2	1	+
1	2	-
2	2	-
3	1	+
3	2	+
3	3	-
3	4	-
1	4	-
4	4	-

The analysis revealed an interesting fact that when at least one of the modules other than the module under consideration had all its nodes with identical row patterns to the

Table 4.3 Three Modules with Variable Number of Nodes

No. of nodes in module under consideration	No. of nodes in M1	No. of nodes M2	Module Information Complexity
3	1	1	+
3	2	1	+
2	2	2	-
4	3	1	+
3	3	1	-
4	3	2	+
4	5	1	-
5	6	3	-
5	3	6	-
5	6	5	-
5	6	6	-
5	2	6	-
5	3	6	-

nodes in the module under consideration, the module under consideration had positive information complexity. Positive information complexity persisted even after satisfying Gottipati's conjectures [17] and the first condition added by Govindarajan. This led to the addition of a second rule to further constrain the row patterns. The new constraints resulted in negative information complexity values for test row patterns that satisfied the criteria.

Conjecture 2 shows Govindarajan's conjecture for negative complexity of a module. The redefined conditions hold for the cases that were used for the analysis.

Conjecture 2 (Govindarajan) *Given a system,* **S***, composed of at least two modules, we call the module of interest as* m_1*. A system that fulfills the conditions below has negative*

module information complexity, Complexity$(m_1|\mathbf{S}) < 0$.

1. *The row patterns of all the nodes in module* m_1 *are identical.*
2. *A hyperedge connected to a node in module* m_1 *must also be connected to all the other nodes in the system.*
3. *There must be at least one hyperedge connected to module,* m_1.
4. *There must be at least one hyperedge not connected to module,* m_1.
5. *There must not be another module with all its nodes having the same pattern as nodes in module* m_1.
6. *Module* m_1 *must not have more nodes than the other modules.*

Example 1 discusses the counter example considered. In Example 2, an example that satisfies Govindarajan's conjectures is shown.

Example 1: A system with two modules with two nodes in the module under consideration and one node in the other module is shown in Table 4.1. This is a counter example to show that Gottipati's conjecture [17] is not always true.

Example 2: A system with two modules with four nodes in one module and one node in the module under consideration. This row pattern in Table 4.4 satisfies the Govindarajan's conjecture for negative complexity for the smaller module.

It was found that the information module complexity tends towards a positive value for larger systems. However, the conjecture still holds for large systems.

All criteria are described using software engineering conventions in terms of classes, methods and public variables, similar to the case study in Chapter VII, as follows.

Table 4.4 A System with Greater than Two Nodes

Module Id	Node Id	RowPattern
M1	N1	11111
M1	N2	11111
M1	N3	11111
M1	N4	11111
M2	N2	10111

M2 has *Complexity*$(m_2|\mathbf{S})$ = -1.27

1. All methods defined in the class under consideration must make use of same public variables.
2. Any public variable used in a method in the class must also be used by all other methods in the system.
3. The class under consideration should use at least one public variable.
4. The class under consideration should not use at least one public variable defined in a different class.
5. There must not be another class with all its methods using the same set of public variables as any method in the class under consideration.
6. Class under consideration should have the least number of methods compared to all the other classes.

In real-world systems, it is very hard to find systems that satisfy the conditions for negative complexity. For example, the systems in our case study never resulted in negative complexity. In Gottipati's [17] research, working programs obtained from the Physics Department were measured. None of the programs measured had a negative information complexity value. He also measured a part of PMLP version 3.0. In this research, we measured PMLP version 4.0. The measurement of three systems in PMLP yielded positive

values of information complexity. These are representatives of the systems in the real-world. So the measurement of several real-world systems, and programs serve as examples to show that the likelihood of negative information complexity in practice is very low.

In order to get negative information complexity, the row patterns of the modules should satisfy all Govindarajan's conjectures shown in Conjecture 2. It is highly unlikely that all the conditions would be true at the same time. Especially, it is very rare that the first and the second conditions are true. The first condition states, "the row patterns of all the nodes in the module are identical". The first condition could be true for a small system where there are only few hyperedges, i.e. public variables in our case study. However, all the methods in a module having identical patterns, that is, all the methods make use of the same hyperedge or public variable, is rare in a bigger system where the number of hyperedges are likely to be more. The condition 2 states, "a hyperedge connected to a node in module m_1 must also be connected to all the other nodes in the system". This condition is highly unlikely for large systems to have public variables connected to a method inside the class under consideration, to be connected to all the methods in the system.

Negative complexity may occur for a very small system but the likelihood of occurence in a real-world system is almost nil because of programming conventions and the design principles of information hiding and encapsulation. Programmers make extensive use of these two conventions, which minimizes the number of public variables declared. By information hiding and encapsulation, variables are protected from having direct access to other classes. This prevents the criteria one from being true, which states in software

engineering terms, “all methods defined in the class under consideration must make use of same public variables”. It is highly likely that condition two which states, “public variable used in a method in the class must also be used by all other methods in the system”, will also be violated. From the measurements obtained for the set of programs in our case study, we conclude that negative module information complexity is highly unlikely in real-world systems.

CHAPTER V

TOOLS

This chapter discusses some of the custom-built tools and off-the-shelf tools used by this research. The architecture shown in Figure 5.1 uses Unix pipes, `stdin` and `stdout`, and Unix `stderr`. Some tools, namely the Reorganizer, Abstraction Extractor, Duplicates Filter, Nodes $\times$ Hyperedges Generator, and Measurement, were custom-built for this research. All except the Duplicates Filter were developed using Java. CPPX, SAS, and Microsoft Excel are the off-the-shelf tools used in this research.

Table 5.1 Tools

Tool Name	Off-the-Shelf/Custom	Programming Language	Developer Name
CPPX	Off-the-Shelf	-	Univ. of Wateloo
Statistical Analysis System	Off-the-Shelf	-	SAS Institute
Microsoft Excel	Off-the-Shelf	-	Microsoft
Reorganizer	Custom	Java	Rajiv Govindarajan
Abstraction Extractor	Custom	Java	Rajiv Govindarajan
Duplicate Filter	Custom	Perl	Rajiv Govindarajan
Nodes x Edges Generator	Custom	Java	Rajiv Govindarajan
Measurement	Custom	Java	Sampath Gottipati

C or C++ code is preprocessed and the resultant output in `*.ii` format is parsed using the command `CPPX -datrix [filename]`. The output of the parser is an

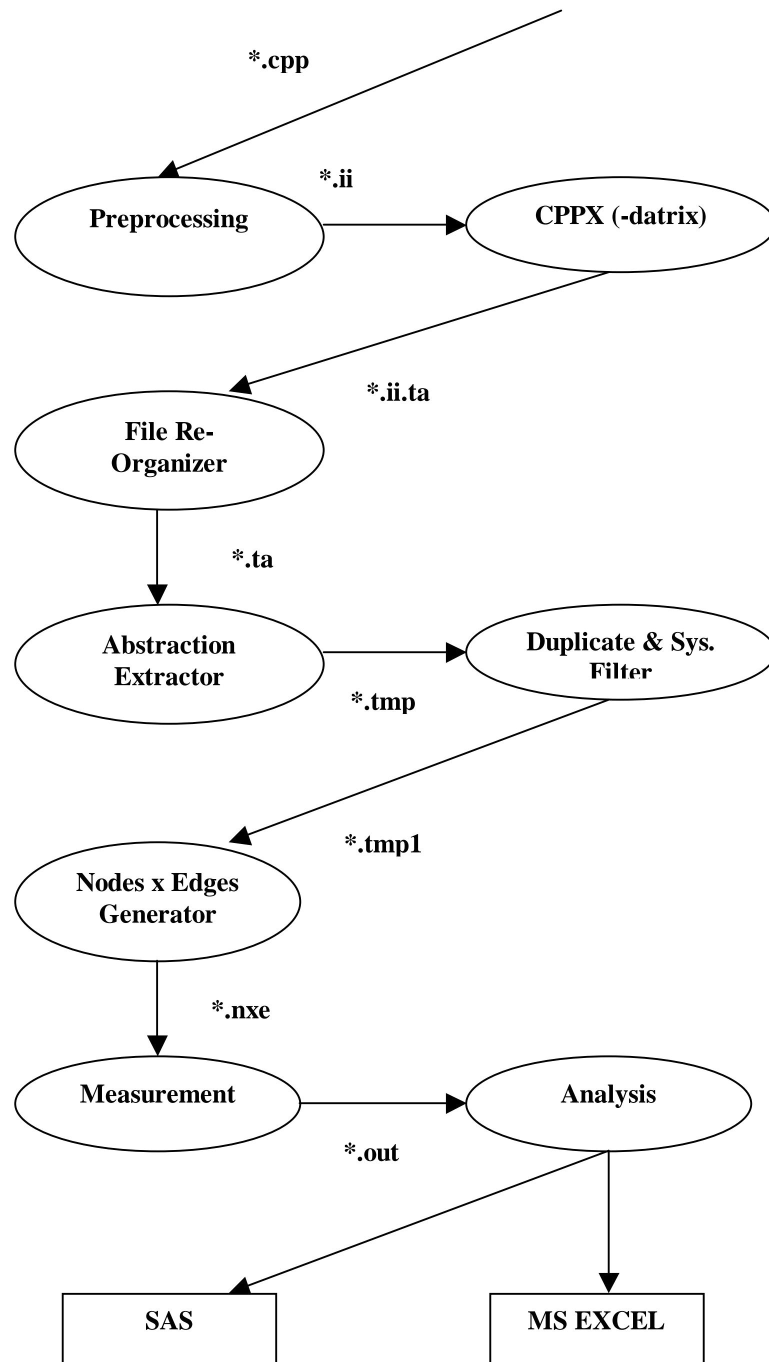

Figure 5.1 Tools Architecture

abstract semantic graph (ASG). The abstract semantic graph obtained is not organized in a convenient form. The output generated by the CPPX is fed as an input to the Reorganizer. Reorganizer reorders the ASG generated by CPPX making it easy for the Abstraction Extractor to extract the class-method-public variable relationships. Abstraction Extractor generates the relationships in the ASG but does not filter out some duplicates and system related files generated during the preprocessing of the C++ code. Duplicate-System Filter identifies the system files generated and the duplicates, removes them from the extracted relationships. The output generated is the input to the Nodes $\times$ Hyperedges generator. The main objective of the Nodes $\times$ Hyperedges generator is to parse through the relationship extracted and produce a nodes $\times$ hyperedges table in a file. The nodes $\times$ hyperedges file is required to have the fields software identity (`swid`), module identity, node identity, and row pattern. The node $\times$ hyperedges table is used to calculate the information theory-based measures of size, complexity, cohesion, and coupling [20]. The output is a comma delimited file. In this research, the measurement process involves various steps as shown in the Figure 5.1. The preprocessing step is handled by the `g++` compilers `-E` option whereas the parsing, creation of nodes $\times$ hyperedges table, and measurement of the metrics are handled by CPPX, the abstractor tool and the measurement tool.

5.1 Datrix

Datrix is a software metrics tool developed by Bell Canada in collaboration with others [6]. In this research Datrix parses the preprocessed C++ source files to produce an

abstract semantic graph. Datrix can also be used to generate traditional software metrics. Datrix tends to throw syntax errors when it encounters C++ template declaration or definition or a template call. This factor led to the search for other tools like CPPX for handling the templates. The inability of Datrix in handling the templates has made the comparison of traditional software metrics generated by Datrix with the information theory-based measures, and the counting-based measures impossible.

5.2 CPPX

CPPX also known as C/C++ Fact Extractor (http://swag.uwaterloo.ca/ dean/cppx) is a program analyzer that converts C++ into an intermediate language for semantic graphs. CPPX provides syntactic and semantic analysis. The abstract semantic graph is generated in a Datrix data model format. The abstract semantic graph produced by the CPPX contains tuple attributes same as Datrix generated abstract semantic graph. The input to the CPPX is any C++ which the `gcc` compiler will accept. The output from the CPPX is translated into a `*.ta` similar to that of a Datrix using `-datrix` switch.

5.3 Reorganizer

In the ASG generated by CPPX, the node type, information on the instances of the node type, and the edge type are not conveniently ordered. Therefore, in order to increase efficiency, the `*.ta` is reorganized in a consistent way that facilitates easy manipulation and less computation. Reorganizer takes a `*.ta` generated by the CPPX as its input. The

node types and corresponding information on the instances of the node types are grouped together. Then the edge types are grouped in a consistent way. The output of this tool is the input to the Abstraction Extractor. The output is generated in `*.ta` format with its node types and edge types similar to the output of Datrix.

5.4 Abstraction Extractor

The purpose of the Abstraction Extractor is to extract relationships among classes, methods, and public variables. These relationships are represented in the form of row patterns in a nodes $\times$ hyperedges table. In our case, classes are modules, methods are the nodes, and public variables are the hyperedges. The reorganized abstract semantic graph generated by Reorganizer serves as an input to this tool. The Abstraction Extractor has two important parts namely the relationship extraction part, and the nodes $\times$ hyperedges table generator part. The relationship extractor identifies the class, method, and public variable relationship. The output generated by this part is used as an input to Duplicates-System filter.

5.5 Duplicates-System Filter

Duplicates-System Filter program takes its input and searches for references to system classes and methods, and repeated or unwanted relationships in the abstraction. A class or a method or a public variable is identified by the directory they belong to. If a class or a method or a public variable is out of the scope of the system under study, it is removed

from the abstraction. The filtered output is then fed to the nodes $\times$ hyperedges table generator. Duplicates-System filter is coded in Perl.

5.6 Nodes $\times$ Hyperedges Generator

The objective of this tool is to take the filtered output and generate a nodes $\times$ hyperedges file. The output of the nodes $\times$ hyperedges table generator is a comma-separated file and has four columns namely system identifier, module name, node name, and row pattern. Row patterns are in the form of 1's and 0 's representing connection to hyperedge or not.

5.7 Measurement

This tool takes row patterns in the nodes $\times$ hyperedges file as the input, and calculates both information theory-based and counting-based metrics at system as well as module levels implementing the formulas in Chapter III. This tool was developed by Sampath Gottipati [17] using Java and was modified by Shea L. Fox.

5.8 Statistical Analysis System (SAS)

Statistical Analysis System (SAS) is a statistical package that we have chosen for the purpose of running different statistical procedures on raw data, performing analysis, and deriving useful results. SAS was developed by the SAS Institute Inc. Statistical analysis of measurements was done using the SAS.

5.9 Microsoft Excel

This tool developed by Microsoft Corporation is a well known tool used for various purposes like accounting, statistical analysis, generation of charts, etc. We use mainly the scatter plot and histogram feature of Excel. The scatter plots were generated for the information theory module measures to the corresponding counting module measures of a system. The use of the scatter plot helps in analyzing the degree to which the measures are correlated. Histograms were generated for comparing and analyzing the system level metrics obtained.

CHAPTER VI

METHODOLOGY

6.1 Methodology

The following section addresses the methodology followed by the case study. This thesis falls under the empirical category as classified by Zelkowitz and Wallace [30]. A case study approach, a type of observational method of data collection, was followed. The case study approach was used because real-world data illustrated the usefulness of the metrics. The case study focused on a large, historical, real-world dataset.

The analysis included the following steps

1. Choose the case study from the repository [20, 21]. Source code file sets needed for the analysis are selected from the repository. In this research we consider a group of files that perform related functions as a "System".

2. Preprocess the source code.

3. Derive abstract semantic graphs (ASG) of the source using CPPX.

4. Extract abstractions from the ASG [20, 21]. For each abstraction, a nodes $\times$ hyperedges table is generated using the Reorganizer, Abstraction Extractor, Duplicates-System Filter, and Nodes $\times$ Hyperedges Generator.

5. For each nodes $\times$ hyperedges table, the information theory-based attributes and corresponding counting-based attributes of the system, and of the modules, are measured.

6. Analyze the distributions and correlations among the measured attributes using SAS and by generating graphs.

7. Examine each information theory-based obtained measures closely to verify the degree to which the results satisfy intuition about the attributes.

8. Summarize the lessons learned comparing the usefulness of the information theory based measures to the counting based measures.

Based on the lessons learned, we answer the research questions and evaluate the stated hypothesis of the research.

6.2 Abstraction Issues

Abstraction Extractor deals with extracting the relationship between a public variable, a method, and a class from a C++ source file. This process involves several issues, which must be considered. This section deals with the explanation of some of the abstraction issues identified during the course of this research. These issues deal with identifying the scope of the system, module, and a node. There are several ways of looking at the scope of a module and the scope of a node. This is related to what constitutes a system.

1. When a file is considered as a system then classes are the modules and methods are the nodes.

2. When group of related files are considered as a single system, methods are the nodes and class or a single C++ source file becomes a module.

3. If the files are independent of each other, a nodes $\times$ hyperedges table is calculated for each file. We call this group of independent “trees” formed as a “forest”.

The abstraction used in this research is the second abstraction. File groups containing related files were considered a system, class as a module, and method as a node. Researchers can make use of these different abstractions in their research.

Handling of Templates: While dealing with a C++ source file, handling of templates of C++ was an issue. The issue was mainly due to the ambiguity that existed regarding the place of templates in the abstraction. This issue was resolved by viewing a template as similar to a class and deriving relationships for that "class".

Handling of Unclassed Method: Another abstraction issue is when a file has many methods but no class. This introduces the problem what constitutes a module. Two candidates considered for module were the namespace to which the file belongs to, and name of the file itself. In this case we consider the file as module similar to a class, to which the methods are related.

6.3 Measurement Issues

This section deals with the explanation of two major measurement issues identified.

Handling of Header File Relationship: The extraction of relationships that are present in the header files was an issue, as preprocessing of the header file alone is not possible. In order to overcome this, `*.cpp` files containing these headers are preprocessed, which in turn includes the relationships that may exist in the header file.The test programs in the case study are preprocessed to measure the different systems identified.

Detecting Use of Public Variables: CPPX represents the occurence of entities in the program by name, line number of the entity, name of the file to which the entity is as-

sociated, and visibility (private or public or protected). The scope of the public variable defined inside a method is obtained by comparing the line number of the method with that of the line number of the public variable. Here we assume that public variable is in the scope of a method if the public variable's line number is greater than that of the starting line of the method and lesser than the end line of the method, and the scope of the files are the same.

CHAPTER VII

CASE STUDY

Parallel Mathematical Library Project (PMLP) was the subject of the case study. Parallel Mathematical Library Project was a joint effort by Intel, Lawrence Livermore National Laboratory, Russian Federal Nuclear Agency (VNIIEF), and the High Performance Computing Laboratory at the Mississippi State University. It is a parallel mathematical library suite for sparse matrices. PMLP includes sequential sparse basic linear algebra, parallel sparse matrix vector products, and sequential and parallel iterative solvers with Jacobi and incomplete LU preconditioners.

PMLP consists of scalable libraries that combine the features of object-oriented design, sequential and parallel modes, etc. An empirical study was conducted on PMLP to get the desired measurements. PMLP was developed in C++ using object-oriented techniques, such as template classes, generic programming, parametrized types, run-time polymorphism, compile-time polymorphism, and iterators. The major part of PMLP was coded as header files. Other files make use of it to perform various mathematical functions. Code using PMLP libraries runs cleanly with a GNU C++ (`g++`) 3.2 version compiler.

Systems to be considered at a higher level were identified. From the analysis of PMLP, we identified several directories in PMLP containing related sets of files that are candidate

systems. Table 7.1 shows the directories in PMLP that can be considered a system. The subdirectories under these highest level directories can be considered subsystems that can help in deriving new intuitions about the system.

Table 7.1 PMLP Directories for a System

	Systems Measured
No.	Directories
1.	solvers together with graph
2.	sp_blas
3.	psp_blas
4.	graph
5.	persistent

Three out of five systems were studied and measured as a part of this research. `Graph`, `SP_BLAS`, and `Solvers` are the systems studied. `psp_blas` and `persistent` were not studied because of the non-availability of programs used to test these systems.

The definitions of a system and a module at different levels of granularity is listed below:

- At the system level, the highest level directory containing libraries that perform set of related functions is considered a system.
- At the module level, as discussed in Chapter VI,
  - Classes are considered modules

- Methods are nodes
- Public variables form the hyperedges represented as columns in a node $\times$ hyperedges table

In the following sections, we discuss the directory structure of each system and the parts of the system that are measured. The following sections also present the measured values of systems under consideration. The root directory of the PMLP is `pmlp`. Preprocessed files generated from the test programs are inputs to the CPPX tool. CPPX generates an abstract semantic graph which is then reorganized. The reorganized abstract semantic graph is used to generate class-method-public variable relationship in the form of a nodes $\times$ hyperedges table. Information theory-based and counting-based measures are calculated for the systems using the row patterns in the nodes $\times$ hyperedges table. Table 7.2 and Table 7.3 shows the system level information theory-based metrics and counting-based metrics for all the systems studied. Looking at the tables containing the system level and module level metrics, we see that cohesion values are either zero or a negligible value. This condition was analyzed. The conjectured reasons for the zero cohesion is discussed in the next chapter.

Table 7.2 System Measurements of Information Theory-Based Metrics

	Information Theory Measurements			
	Size (bits)	Complexity (bits)	Coupling (bits)	Cohesion
Graph	16.8	37.1	13.9	0.12
Solvers	2417.8	313776.7	313776.7	0
SP_BLAS	6174.4	1504101.9	1504101.9	0

Table 7.3 System Measurements of Counting-Based Metrics

	Counting Based Measurements			
	Size (nodes)	Complexity (hyperedges)	Coupling (hyperedges)	Cohesion
Graph	6	112	79	0.29
Solvers	300	258	255	0.02
SP_BLAS	681	246	246	0

7.1 Solvers

This system of iterative and direct solvers has `pmlp/src/solvers` as its root directory. `pmlp/src/solvers/include` contains the header files for `Solvers` functionality. `Solvers` includes sequential iterative and direct solvers, and parallel iterative and direct solvers. The source files of the `Solvers` could be found in the `pmlp/src/solvers/src` directory path. Test programs are found in the `test` directory of the `Solvers` root. `test.cpp` file is preprocessed and an abstract semantic graph is generated using the CPPX. The class-method-public variable relationship is extracted for this system making use of a test program. `Solvers` makes use of the `Graph` system to perform some of its functions.

In order to measure `Solvers`, a test program inside the `pmlp/src/solvers/test/direct_perl/functions` directory is preprocessed making use of the `-E` switch in the `g++` compiler. This switch is added to the `Makefile` and the output is a `*.ii` file. This `*.ii` generated is then used as an input to CPPX which when run with

-datrix switch produces a Datrix like abstract semantic graph (ASG). The ASG file obtained is then reorganized, abstraction extracted, and measured. The measured information theory metrics, and counting metrics, at both the system level and module level, are comma-separated in the output file. Fifty two modules were identified within the scope of Solvers. Both system level as well as module level metrics were calculated for the modules. Table 7.4 shows a sample nodes × hyperedges table generated for the Solvers system. Table 7.6 and Table 7.5 shows the module level information theory-based metrics and counting-based metrics for the Solvers system. It was observed from the table that values of complexity are often equal to coupling. This is discussed in detail in next chapter (Section 8.4.1).

7.2 Graph

Graph system consists of group of methods implemented to perform different reordering algorithms. Graph has pmlp/src/graph as its root directory. pmlp/src /graph/include directory consists of header files that are included to make use of the methods in the Graph system. The relationships in the Graph system can be obtained by making use of the test programs in the Solvers. Preprocessed file generated for the Graph system is used to generate the abstract semantic graph. The abstract semantic graph generated by the CPPX is used to derive the class-method-public variable relationship that exist within the scope of the Graph system. Class-method-public vari-

Table 7.4 Sample Nodes × Hyperedges Table for Solvers

SystemId	ModuleId	NodeId	Row Patterns
Solvers	arrays_in_gmres.hxx	ArraysInGMRES	00001000000000000000000000000000000000000000000...
Solvers	ConvergenceCriterion<MT>	Convergence_test_initial	11010101100111000000000001011...
Solvers	ConvergenceCriterion<MT>	New_coarse	0001010110001100000000000101101100111001100 0...
Solvers	dendy_black_box.h	New_coarse_iter_solver	0000010110001100000000000100100000000100 ...
Solvers	dendy_black_box.h	Solve	0000010000000000000000000100000000000010000...
Solvers	DomainDecomposition<MT>	Init_domain_decomposition	000000000010001100001001100000000000000 ...
Solvers	gauss_seidel_dns.h	New_coarse	0000100000000000000000000000000000000000000000...
Solvers	handles.hxx	ISolverHandle	00000101100011000000000001001000000001001...
Solvers	handles.hxx	'operator='	0000010000000000000000000100000000000100110000 0...
Solvers	ilu.h	PreconditionerILU	000011100110111110011001110010000000111110...
Solvers	ISolver<MT>	Solve	0001010110001100000000000101101100111001100 0...
Solvers	jac.h	New_coarse	00001010011000111001100110000000000000110000111001...
Solvers	ldlt.h	Get_diag	000010100110001110011001100000000000001100001110...
Solvers	lu.h	New_coarse_direct_solver	00000100000000100000000000100100000000100110000000...
Solvers	multigrid_parameters.h	MultigridParameters	00000111011011110001100111001000000001111 1 ...
Solvers	PreconditionerSOR<MT>	Clone_preconditioner	00000000000000000000000000000000000000 ...
Solvers	PreconditionerSSOR<MT>	Get_upp	00000010010000000100100000000000000000110000010001...
Solvers	schwartz_additive...	New_coarse	000010100100000111001100000000000000001...
Solvers	solver.hxx	Apply	000101111110111110000100111011011001111111001111...
Solvers	solver.hxx	Set_tolerance	0000000000100011000010011000000000000000000001010 ...
Solvers	solver_trace.h	New	0000000000000000000000000000000000000000000000...
...	...	...	...

Table 7.5 Module Level Information Measurements for Solvers

ModuleId	Size (bits)	Complexity (bits)	Coupling (bits)	Cohesion
aip.h	74.1	9455.1	9455.1	0
arrays_in_gmres.hxx	8.2	670.6	670.6	0
block.h	16.5	704.9	704.9	0
choleski.h	23.7	2374.1	2374.1	0
convergence.hxx	75.1	2725.2	2725.2	0
ConvergenceCriterion<MT>	131.7	15025.7	15025.7	0
dendy_black_box.h	64.9	4716.8	4716.8	0
direct_solver.hxx	24.7	2182.8	2182.8	0
DirectSolver<MT>	8.2	1211.6	1211.6	0
domain_decomposition_class.h	8.2	378.2	378.2	0
DomainDecomposition<MT>	80.3	9022.3	9022.3	0
gauss_seidel.h	16.5	2527.9	2527.9	0
gauss_seidel_dns.h	32.9	3170.1	3170.1	0
gauss_seidel_sym.h	41.2	5957.8	5957.8	0
gauss_seidel_sym_dns.h	49.4	6176.6	6176.6	0
GraphStorFormat	49.4	10079.4	10079.4	0
handles.hxx	146.2	20475.0	20475.0	0
ic.h	97.8	9065.6	9065.6	0
ic_sqrt.h	16.5	2245.7	2245.7	0
ic0.h	24.7	3581.0	3581.0	0
ict.h	23.1	3769.6	3769.6	0
ildu.h	48.4	4646.0	4646.0	0
ilu.h	105.0	22103.0	22103.0	0
ilu_pivot.h	41.2	1764.9	1764.9	0
ilu0.h	56.6	9946.5	9946.5	0
ilut.h	72.5	9721.3	9721.3	0
isolver.hxx	7.2	1078.6	1078.6	0
isolver_classes.hxx	56.9	7484.3	7484.3	0
ISolver<MT>	153.4	19912.6	19912.6	0
jac.h	41.2	6454.5	6454.5	0
ldlt.h	41.2	5905.9	5905.9	0
lu.h	106.0	20938.6	20938.6	0
mict.h	32.9	6014.6	6014.6	0
milut.h	56.1	9459.1	9459.1	0
multigrid_parameters.h	24.7	3177.1	3177.1	0
preconditioner_identity.hxx	24.7	2729.2	2729.2	0
Preconditioner<MT>	8.2	370.8	370.8	0
PreconditionerSchwartzAdditive<MT>	8.2	1051.7	1051.7	0
PreconditionerSOR<MT>	73.1	5778.4	5778.4	0
PreconditionerSSOR<MT>	88.6	9478.9	9478.9	0
qr.h	40.2	6091.1	6091.1	0
saad.h	8.2	1421.7	1421.7	0
schwartz_additive.h	8.2	269.4	269.4	0
schwartz_additive_coarse_grid.h	31.9	4778.4	4778.4	0
schwartz_multiplicative.h	16.5	1770.6	1770.6	0
solver.hxx	107.0	19448.9	19448.9	0
solver_trace.h	80.3	6728.3	6728.3	0
Solver<MT>	8.2	1302.5	1302.5	0
SolverTrace<VT>	8.2	1528.6	1528.6	0
sor.h	8.2	297.3	297.3	0
sp_cgs.h	32.9	6427.9	6427.9	0
ssor.h	8.2	179.6	179.6	0

Table 7.6 Module Level Counting Measurements for Solvers

ModuleId	Size (nodes)	Complexity (hyperedges)	Coupling (hyperedges)	Cohesion
aip.h	9	196	196	0
arrays_in_gmres.hxx	1	13	13	0
block.h	2	14	14	0
choleski.h	3	132	132	0
convergence.hxx	11	83	83	0
ConvergenceCriterion<MT>	16	105	105	0
dendy_black_box.h	8	60	60	0
direct_solver.hxx	3	51	51	0
DirectSolver<MT>	1	74	74	0
domain_decomposition_class.h	1	5	5	0
DomainDecomposition<MT>	10	186	186	0
gauss_seidel.h	2	164	164	0
gauss_seidel_dns.h	4	98	98	0
gauss_seidel_sym.h	5	151	151	0
gauss_seidel_sym_dns.h	6	132	132	0
GraphStorFormat	6	171	171	0
handles.hxx	18	152	152	0
ic.h	12	220	220	0
ic_sqrt.h	2	123	123	0
ic0.h	3	187	187	0
ict.h	3	194	194	0
ildu.h	6	129	129	0
ilu.h	13	195	195	0
ilu_pivot.h	5	65	65	0
ilu0.h	7	190	190	0
ilut.h	9	194	194	0
isolver.hxx	1	49	49	0
isolver_classes.hxx	8	58	58	0
ISolver<MT>	19	135	135	0
jac.h	5	173	173	0
ldlt.h	5	155	155	0
lu.h	13	258	253	0.00195
mict.h	4	199	199	0
milut.h	7	194	194	0
multigrid_parameters.h	3	160	160	0
preconditioner_identity.hxx	3	125	125	0
Preconditioner<MT>	1	10	10	0
PreconditionerSchwartzAdditive<MT>	1	37	37	0
PreconditionerSOR<MT>	9	59	59	0
PreconditionerSSOR<MT>	11	74	74	0
qr.h	5	162	162	0
saad.h	1	74	74	0
schwartz_additive.h	1	4	4	0
schwartz_additive_coarse_grid.h	4	122	122	0
schwartz_multiplicative.h	2	75	75	0
solver.hxx	13	186	186	0
solver_trace.h	10	171	171	0
Solver<MT>	1	88	88	0
SolverTrace<VT>	1	107	107	0
sor.h	1	8	8	0
sp_cgs.h	4	149	149	0
ssor.h	1	1	1	0

able relationship is generated as a nodes × hyperedges table, which is then measured to obtain information theory-based, and counting-based measures.

Graph system is used by Solvers system for performing some set of functions. Test programs of the Solvers were used to measure the Graph system. Steps applied for the measurement, as mentioned earlier were repeated. Measured Graph system had two modules with class-method-public variable relationship. The modules were measured for system level and module level metrics. Table 7.7 shows the sample nodes × hyperedges table generated for the Graph system. Table 7.8 and Table 7.9 shows the module level information theory-based metrics and counting-based metrics for the Graph system.

7.3 SP_BLAS

Sequential Sparse Basic Linear Algebra is one of the systems considered for the study. The source of SP_BLAS is inside the pmlp/src/sp_blas/ directory. Simple test programs are found inside the pmlp/src/sp_blas/test directory. testgen.cpp is a test program generator, which when compiled generates test programs for testing different functions of the SP_BLAS. mat_elem, mat_mat, mat_vec,vec_elem, and vec_vec are some of the directories generated by the testgen.cpp. These directories have source to perform functions like matrix-element functions, matrix-matrix functions, matrix-vector, vector-element, and vector-vector functions. These directories are tested as a part of SP_BLAS.

Table 7.7 Sample Nodes × Hyperedges Table for Graph

SystemId	ModuleId	NodeId	Row Patterns
GRAPH	GraphStorFormat	GraphStorFormat	1110111111011000011111111011110100001011110111111 ...
GRAPH	GraphStorFormat	'operator='	101000001000110000000011101011000100001010010010100 ...
GRAPH	sp_cgs.h	Copy	0000100011000000000000000000000000000001100000001 ...
GRAPH	sp_cgs.h	GetEdgeWeigth	1110011100011000011111111011110100001010010111110 ...
GRAPH	sp_cgs.h	Print_sf	000100000010011110000000001000010111101000001000000 ...

Table 7.8 Module Level Information Measurements for Graph

ModuleId	Size (bits)	Complexity (bits)	Coupling (bits)	Cohesion
sp_cgs.h	8.4	9.9	5.8	0.05
GraphStorFormat	8.4	27.2	8.2	0.44

Table 7.9 Module Level Counting Measurements for Graph

moduleId	Size (nodes)	Complexity (hyperedges)	Coupling (hyperedges)	Cohesion
sp_cgs.h	3	108	77	0.032
GraphStorFormat	3	83	77	0.268

The test programs of the `SP_BLAS` were preprocessed and the resultant file was used as an input to the CPPX. The steps for the measurement of system level and module level metrics were repeated. There were 95 modules in the `SP_BLAS` system. Row patterns obtained for the 95 modules were measured. Table 7.10 shows the sample nodes $\times$ hyperedges table generated for the `SP_BLAS` system. Table 7.11 and Table 7.12 shows the module level information theory-based metrics and counting-based metrics for the `SP_BLAS` system.

Table 7.10 Sample Nodes × Hyperedges Table for SP_BLAS

SystemId	ModuleId	NodeId	Row Patterns
SP_BLAS	complex<float>	complex	1111111001100011111010110011101100111111100110101...
SP_BLAS	'complex<long >	complex	0100010000100010000000110010100100000101001001000010 ...
SP_BLAS	DenseRows<PREC>	Goto_first	0000000000100000000000000001000010000000001000 ...
SP_BLAS	DenseRows<PREC>	Is_in_col_begin	00000000000000001000000000101010000l ...
SP_BLAS	DNS<PREC>	Set_flag_sorted	1110010101110111110010110111111110011111101 ...
SP_BLAS	FormatOtherRow	FormatOtherRow	000000000000000000000010000000000000000000000001...
SP_BLAS	MatNotHermSymskewGen	'operator='	1110000000000010000000000000000000000000000 ...
SP_BLAS	Matrix<PR_SF_MT>	Remove_share	00100001000000000000000000000000000000000 ...
SP_BLAS	MatrixPool<double>	Is_not_empty	00000000100000000000101001100000000000000...
SP_BLAS	MatSymBase	MatSymBase	000000000000000000000000000000000000000000000000000000 ...
SP_BLAS	RefCount	Decr_ref_count	0000000000000000000000000000000000000000000000010000100 ...
SP_BLAS	RefCount	Lock	0000000000000000000000000010000000000000000000001000010000 ...
SP_BLAS	SfCsrUser	SfCsrUser	1110010110010110000101001100010010000000010001010000000 ...
SP_BLAS	sp_coo.h	Resize	0100010000000000000000000000000000000000000000000000000 ...
SP_BLAS	sp_ell.h	Goto_first_in_row	00000000010000010100100000001000000000100 ...
SP_BLAS	sp_list_err	sp_list_err	111001000000000100000000110000000000000001000010010...
SP_BLAS	sp_mtherm.h	Get_element	1110010100010110000000011010001001000010010100100100 ...
SP_BLAS	sp_mthermskew.h	Insert_element	11100101000000100000000000000000000000000010 ...
SP_BLAS	sp_mtsymskew.h	MatrixSymmetricSkew	11100101100101100001011111000100100010 ...
SP_BLAS	sp_skyupp.h	Goto_next_in_row	000000000000000000000000000000000000000000000...
SP_BLAS	sp_sp_vec.h	'operator+='	000000000000000000000000000000000000000010000 ...
SP_BLAS	VectorBlocked<double>	'operator()'	11100101000101100000000110100010010000 ...
SP_BLAS	VectorRep<PR>	'operator()'	1110010110010110000101000100010010000000001010 ...
SP_BLAS	VectorRep<PR>	VectorRep	1110010110010110000101001100010010000000001010100 ...
...	...	...	...

Table 7.11 Module Level Information Measurements for SP_BLAS

ModuleId	Size (bits)	Complexity (bits)	Coupling (bits)	Cohesion
AuxiliarySF	17.8	2302.3	2302.3	0
complex_base<double>	71.1	20461.5	20461.5	0
complex<double>	25.2	3194.6	3194.6	0
complex<float>	43.1	18911.2	18911.2	0
'complex<long	47.1	16677.0	16677.0	0
CSR<double>	166.4	39393.0	39393.0	0
DenseRows<PREC>	460.5	94358.3	94358.3	0
DIA<PREC>	102.0	20299.8	20299.8	0
DNS<PREC>	435.1	106570.4	106570.4	0
'ExtendedPrecision<pmlp::complex<long	16.8	9341.4	9341.4	0
File_in_Binary	52.5	3851.4	3851.4	0
File_out_Binary	65.9	10392.3	10392.3	0
FormatCol	9.4	531.3	531.3	0
FormatConvertCols	9.4	296.8	296.8	0
FormatConvertDense	18.8	9244.3	9244.3	0
FormatConvertRows	18.8	5451.0	5451.0	0
FormatCOO	9.4	5643.8	5643.8	0
FormatCSC	9.4	2342.5	2342.5	0
FormatCSR	18.8	7992.2	7992.2	0
FormatDenseRows	9.4	647.1	647.1	0
FormatDIA	16.8	2129.7	2129.7	0
FormatDNS	9.4	702.3	702.3	0
FormatNotCol	18.8	6515.6	6515.6	0
FormatNotCOO	16.8	5143.5	5143.5	0
FormatNotRow	17.8	8714.3	8714.3	0
FormatOtherGen	18.8	2747.8	2747.8	0
FormatOtherNotRow	18.8	8304.3	8304.3	0
FormatOtherRow	18.8	3268.8	3268.8	0
FormatSorted	16.8	2528.6	2528.6	0
FormatUnsorted	9.4	2002.7	2002.7	0
Head	18.8	3953.7	3953.7	0
HeadContents	9.4	2433.9	2433.9	0
header_matrix	15.7	2266.4	2266.4	0
header_vector	18.8	442.2	442.2	0
IteratorSubmatrixBlocked	9.4	744.3	744.3	0
MapIndex	16.8	2394.0	2394.0	0
MatBand	8.4	742.2	742.2	0
MatGen	8.4	1337.1	1337.1	0
MatHasLowtriang	9.4	629.0	629.0	0
MatHermSym	8.4	954.7	954.7	0
MatHermSymskew	9.4	2046.1	2046.1	0
MatNotHermSymskewGen	17.8	2205.0	2205.0	0
MatNotTrianglowGen	16.8	2485.8	2485.8	0
'Matrix<double_pmlp::CSR<double>	18.8	1828.4	1828.4	0
Matrix<PR_SF_MT>	173.3	46310.9	46310.9	0
'MatrixBanded<pmlp::CSR<double>	9.4	589.1	589.1	0
'MatrixBase1<double_pmlp::CSR<double>	101.5	21866.0	21866.0	0
'MatrixBase2<typename	25.7	3960.7	3960.7	0

Table 7.11 Module Level Information Measurements for SP_BLAS (continued)

'MatrixGeneral<pmlp::CSR<double>	9.4	1861.7	1861.7	0
'MatrixGeneralBase<double_pmlp::CSR	136.0	25766.6	25766.6	0
MatrixPool<double>	171.7	54335.9	54335.9	0
'MatrixSymmetricBase<typename	9.4	1568.8	1568.8	0
'MatrixTriangularBase<typename	9.4	2016.8	2016.8	0
MatSym	9.4	686.0	686.0	0
MatSymBase	18.8	1502.3	1502.3	0
MatSymSkew	18.8	2096.9	2096.9	0
MatTriangBase	18.8	1853.2	1853.2	0
MatTriangLow	9.4	625.5	625.5	0
MatTriangUpp	9.4	621.8	621.8	0
MatWithAnyDiag	9.4	637.5	637.5	0
MatWithoutDiag	15.7	1419.3	1419.3	0
RefCount	110.0	24855.7	24855.7	0
SfCsrUser	16.8	7524.2	7524.2	0
SKY_LOWER<PREC>	163.3	44051.8	44051.8	0
sp_complex.h	9.4	1971.8	1971.8	0
sp_coo.h	174.3	38929.3	38929.3	0
sp_csc.h	119.4	32182.4	32182.4	0
sp_dns.h	7.4	630.1	630.1	0
sp_ell.h	346.0	56964.5	56964.5	0
sp_list_err	7.4	2765.6	2765.6	0
sp_mtherm.h	205.1	75055.5	75055.5	0
sp_mthermskew.h	229.8	66215.3	66215.3	0
sp_mtsymmetric.h	64.9	13672.3	13672.3	0
sp_mtsymskew.h	167.4	68233.9	68233.9	0
sp_mttriangular.h	7.4	1394.1	1394.1	0
sp_poolvec.h	63.9	24677.2	24677.2	0
sp_refelem.h	64.9	10571.0	10571.0	0
sp_sf_user.h	96.2	21131.5	21131.5	0
sp_skylow.h	9.4	1936.9	1936.9	0
sp_skyupp.h	470.8	129445.7	129445.7	0
sp_sp_vec.h	241.8	29253.1	29253.1	0
sp_subvec.h	166.4	38334.7	38334.7	0
sp_vec_blocked.h	9.4	1341.7	1341.7	0
sp_vec_user.h	26.7	4549.3	4549.3	0
sp_vector_norm_class.h	28.2	6174.2	6174.2	0
StorFormat<double>	47.1	11515.9	11515.9	0
StorFormat<PR>	75.7	16822.4	16822.4	0
Vector<double>	37.7	5319.1	5319.1	0
'VectorBase<double_pmlp::VectorBlocked<double>	16.8	858.7	858.7	0
'VectorBase<PR_pmlp::Vector<PREC>	57.7	19229.4	19229.4	0
'VectorBase<PR_pmlp::VectorBlocked<PR>	9.4	2172.1	2172.1	0
VectorBlocked<double>	175.3	56829.8	56829.8	0
VectorNorm<VEC>	47.1	14361.6	14361.6	0
VectorPool<double>	18.8	7937.1	7937.1	0
VectorRep<PR>	170.9	55052.4	55052.4	0

Table 7.12 Module Level Counting Measurements for SP_BLAS

ModuleId	Size (nodes)	Complexity (hyperedges)	Coupling (hyperedges)	Cohesion
AuxiliarySF	2	42	42	0
complex_base<double>	8	113	113	0
complex<double>	3	25	25	0
complex<float>	5	161	161	0
'complex<long	5	99	99	0
CSR<double>	18	239	239	0
DenseRows<PREC>	50	199	199	0
DIA<PREC>	11	137	137	0
DNS<PREC>	48	188	188	0
'ExtendedPrecision<pmlp::complex<long	2	110	110	0
File_in_Binary	6	40	40	0
File_out_Binary	7	94	94	0
FormatCol	1	1	1	0
FormatConvertCols	1	1	1	0
FormatConvertDense	2	121	121	0
FormatConvertRows	2	112	112	0
FormatCOO	1	196	196	0
FormatCSC	1	44	44	0
FormatCSR	2	102	102	0
FormatDenseRows	1	6	6	0
FormatDIA	2	33	33	0
FormatDNS	1	5	5	0
FormatNotCol	2	196	196	0
FormatNotCOO	2	70	70	0
FormatNotRow	2	119	119	0
FormatOtherGen	2	40	40	0
FormatOtherNotRow	2	210	210	0
FormatOtherRow	2	42	42	0
FormatSorted	2	50	50	0
FormatUnsorted	1	42	42	0
Head	2	98	98	0
HeadContents	1	62	62	0
header_matrix	2	29	29	0
header_vector	2	4	4	0
IteratorSubmatrixBlocked	1	18	18	0
MapIndex	2	12	12	0
MatBand	1	3	3	0
MatGen	1	12	12	0
MatHasLowtriang	1	1	1	0
MatHermSym	1	6	6	0
MatHermSymskew	1	25	25	0
MatNotHermSymskewGen	2	10	10	0
MatNotTrianglowGen	2	12	12	0
'Matrix<double_pmlp::CSR<double>	2	40	40	0
Matrix<PR_SF_MT>	19	178	178	0
'MatrixBanded<pmlp::CSR<double>	1	1	1	0
'MatrixBase1<double_pmlp::CSR<double>	11	93	93	0
'MatrixBase2<typename	3	63	63	0

Table 7.12 Module Level Counting Measurements for SP_BLAS (continued)

ModuleId	Size (nodes)	Complexity (hyperedges)	Coupling (hyperedges)	Cohesion
'MatrixGeneral<pmlp::CSR<double>	1	32	32	0
'MatrixGeneralBase<double_pmlp::CSR	15	92	92	0
MatrixPool<double>	19	221	221	0
'MatrixSymmetricBase<typename	1	36	36	0
'MatrixTriangularBase<typename	1	44	44	0
MatSym	1	2	2	0
MatSymBase	2	4	4	0
MatSymSkew	2	15	15	0
MatTriangBase	2	10	10	0
MatTriangLow	1	1	1	0
MatTriangUpp	1	1	1	0
MatWithAnyDiag	1	3	3	0
MatWithoutDiag	2	9	9	0
RefCount	12	97	97	0
SfCsrUser	2	90	90	0
SKY_LOWER<PREC>	18	169	169	0
sp_complex.h	1	34	34	0
sp_coo.h	19	120	120	0
sp_csc.h	13	183	183	0
sp_dns.h	1	2	2	0
sp_ell.h	41	199	199	0
sp_list_err	1	47	47	0
sp_mtherm.h	22	132	132	0
sp_mthermskew.h	25	211	211	0
sp_mtsymmetric.h	7	108	108	0
sp_mtsymskew.h	18	153	153	0
sp_mttriangular.h	1	31	31	0
sp_poolvec.h	7	131	131	0
sp_refelem.h	7	127	127	0
sp_sf_user.h	11	57	57	0
sp_skylow.h	1	29	29	0
sp_skyupp.h	52	228	228	0
sp_sp_vec.h	26	204	204	0
sp_subvec.h	18	109	109	0
sp_vec_blocked.h	1	22	22	0
sp_vec_user.h	3	49	49	0
sp_vector_norm_class.h	3	75	75	0
StorFormat<double>	5	203	203	0
StorFormat<PR>	9	70	70	0
Vector<double>	4	23	23	0
'VectorBase<double_pmlp::VectorBlocked<double>	2	6	6	0
'VectorBase<PR_pmlp::Vector<PREC>	7	128	128	0
'VectorBase<PR_pmlp::VectorBlocked<PR>	1	49	49	0
VectorBlocked<double>	19	117	117	0
VectorNorm<VEC>	5	75	75	0
VectorPool<double>	2	97	97	0
VectorRep<PR>	19	119	119	0

CHAPTER VIII

STATISTICAL ANALYSIS

This chapter presents statistical analysis performed in this research. Summary statistics and correlation coefficients were calculated on measured information theory-based measures and the corresponding counting based measures. The distribution of the measures were analyzed. Insight into the software development methods and the lessons learned were summarized.

8.1 Preliminary Data Analysis

Preliminary data analysis performed on the collected data led to some interesting results. Table 8.1, Table 8.2, and Table 8.3 show summary statistics for the three systems. Metric names are abbreviated into variable names. From Table 8.1, Table 8.2, and Table 8.3 we see that the range of the distributions is not extremely large nor constant for module measurements, except cohesion. This observation is essential to compare the suitability of the metrics for software quality modeling.

Some systems had zero cohesion. `SP_BLAS` is one of the systems in PMLP that was measured (see Table 7.2 and Table 7.3). Further analyzing the measured values, it is seen that the cohesion values of both information theory, and counting based measures for the

Table 8.1 Summary Statistics for Graph

Variable	Mean	Std Dev	Variance	Minimum	Median	Maximum	Range
moduleinfosize	8.4	0.0	0.0	8.4	8.4	8.4	0.0
moduleinfocomplexity	18.5	12.2	149.4	9.9	18.5	27.2	17.3
moduleinfocoupling	6.9	1.7	2.9	5.8	6.9	8.2	2.4
moduleinfocohesion	0.2	0.3	0.07	0.04	0.2	0.4	0.4
modulecountingsize	3	0.0	0.0	3	3	3	0.0
modulecountingcomplexity	95.5	17.7	312.5	83	95.5	108	25
modulecountingcoupling	77.0	0	0	77	77	77	0
modulecountingcohesion	0.2	0.2	0.03	0.03	0.2	0.3	0.2

Number of modules $n = 2$

Table 8.2 Summary Statistics for Solvers

Variable	Mean	Std Dev	Variance	Minimum	Median	Maximum	Range
moduleinfosize	46.5	38.3	1465.4	7.2	36.6	153.4	146.2
moduleinfocomplexity	6034.2	5854.7	34277842.5	179.7	4681.4	22103	21923.37
moduleinfocoupling	6034.2	5854.7	34277842.5	179.7	4681.4	22103	21923.4
moduleinfocohesion	0.0	0.0	0.0	0.0	0.0	0.0	0.0
modulecountingsize	5.8	4.8	22.6	1	4.5	19	18
modulecountingcomplexity	117.6	67.1	4502.2	1	127	258	257
modulecountingcoupling	117.6	67.1	4502.2	1	127	258	257
modulecountingcohesion	0.0	0.0	0.0	0.0	0.0	0.002	0.002

Number of modules $n = 52$

Table 8.3 Summary Statistics for SP_BLAS

Variable	Mean	Std Dev	Variance	Minimum	Median	Maximum	Range
moduleinfosize	65	97.1	9435.5	7.4	18.8	470.7	463.3
moduleinfocomplexity	15832.6	24719.1	611033730	296.8	3960.7	129445.7	129148.9
moduleinfocoupling	15832.6	24719.1	611033730	296.7	3960.7	129445.7	129148.9
moduleinfocohesion	0.0	0.0	0.0	0.0	0.0	0.0	0.0
modulecountingsize	7.2	10.7	114.9	1	2	52	51
modulecountingcomplexity	78.6	68.7	4721.7	1	57	239	238
modulecountingcoupling	78.6	68.7	4721.7	1	57	239	238
modulecountingcohesion	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Number of modules $n = 95$

module and the system are zeros (see Table 7.11 and Table 7.12). This is analyzed in detail in Section 8.4.1.

8.2 Correlation Analysis

Correlation analysis analyzes the degree to which the two metrics are correlated in the context of this case study. The correlation coefficient of each of the individual information theory-based metrics to the corresponding counting-based metrics was calculated at the module level. We prepared scatter plots using MS Excel, and performed correlation calculations using SAS. Figure 8.1, Figure 8.2, Figure 8.3, Figure 8.4, Figure 8.5, and Figure 8.6 show the scatter plots for `Solvers` and `SP_BLAS`. There are only two modules in the `Graph` system so preparing a scatter plot is not useful. The figures show the scatter plots for the measures of size, complexity, and coupling. The cohesion values in the systems are zeros, so the correlation coefficient of the constant values is not meaningful. The system measures are represented as histograms. The number of modules measured in the two systems considered, `Solvers` and `SP_BLAS`, are 53 and 95 respectively.

In Figure 8.1 and Figure 8.4, size measures in both systems look highly correlated. This suggests that using either counting or information size would yield almost similar results in a software quality model.

Table 8.4 and Table 8.5 shows correlation values for information theory-based metrics to the corresponding counting-based metrics for the `Solvers` and `SP_BLAS`. From the tables we can see that infomation size is highly correlated with counting size.

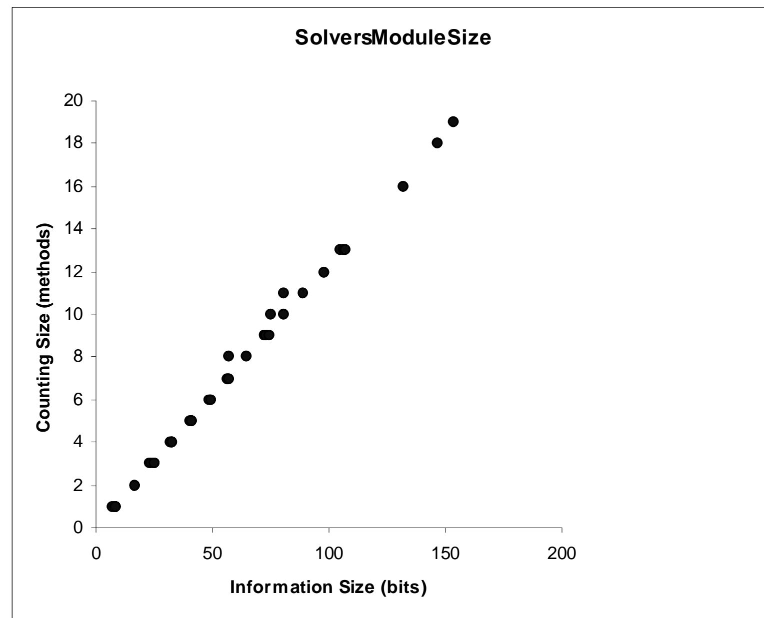

Figure 8.1 Information Module Size *vs.* Counting Module Size of Solvers

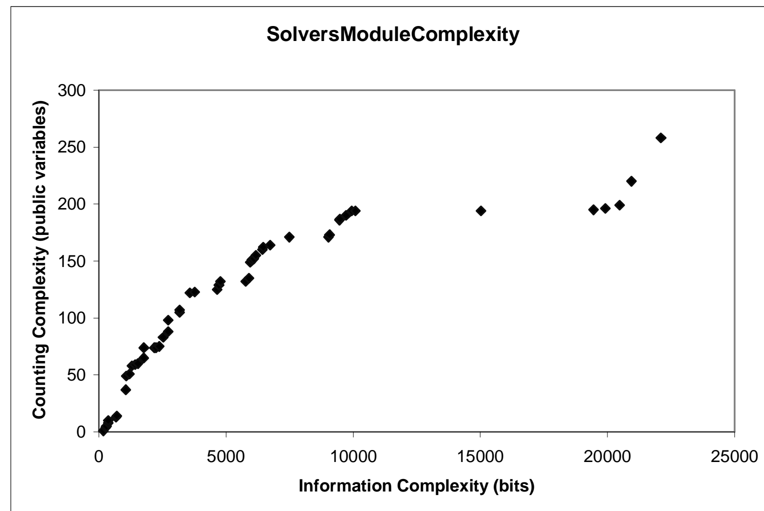

Figure 8.2 Information Module Complexity *vs.* Counting Module Complexity of Solvers

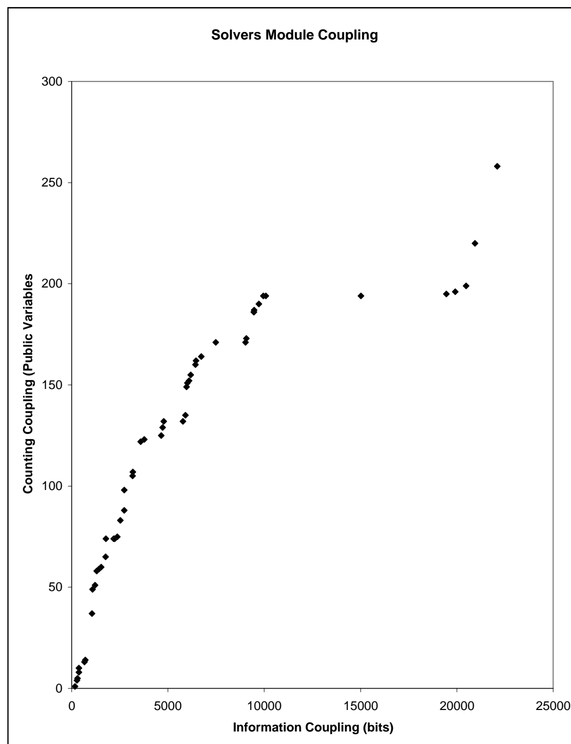

Figure 8.3 Information Module Coupling *vs.* Counting Module Coupling of Solvers

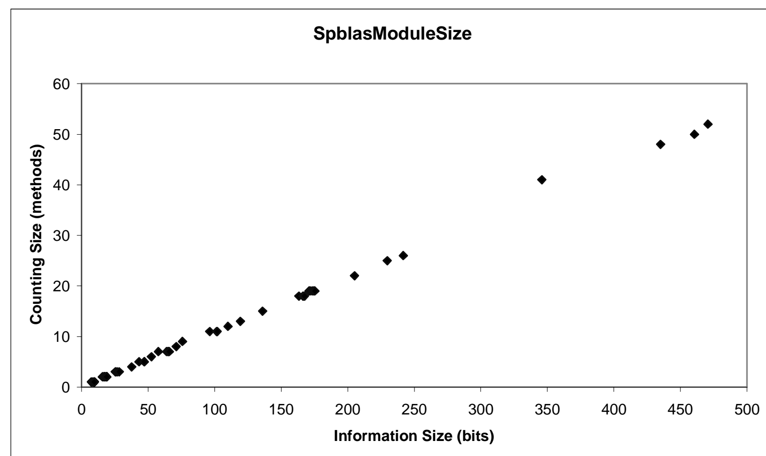

Figure 8.4 Information Module Size *vs.* Counting Module Size of SP_BLAS

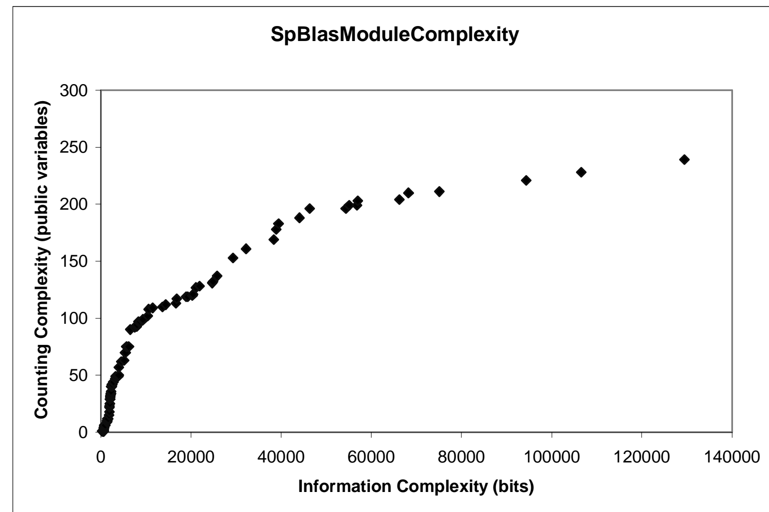

Figure 8.5 Information Module Complexity *vs.* Counting Module Complexity of SP_BLAS

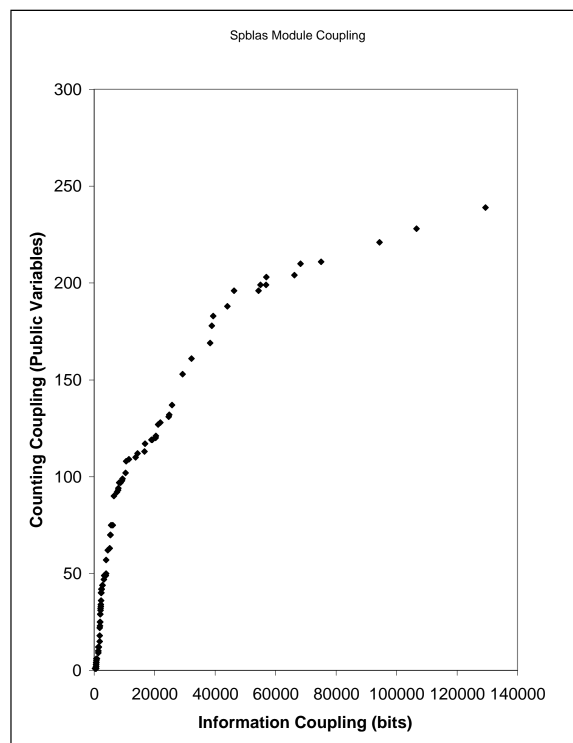

Figure 8.6 Information Module Coupling *vs.* Counting Module Coupling of SP_BLAS

Table 8.4 Correlation among Solvers Module Measurements

	Infosize	Infocomplexity	Infocoupling	Countingsize	Countingcomplexity	Countingcoupling	Countingcohesion
Infosize	1	0.901	0.783	0.998	0.488	0.488	0.219
Infocomplexity	0.901	1	0.793	0.891	0.618	0.618	0.359
Infocoupling	0.783	0.793	1	0.771	0.519	0.519	0.359
Countingsize	0.998	0.89	0.771	1	0.477	0.837	0.215
Countingcomplexity	0.488	0.617	0.519	0.477	1	0.649	0.295
Countingcoupling	0.488	0.617	0.519	0.477	0.649	1	0.398
Countingcohesion	0.219	0.359	0.359	0.215	0.295	0.398	1

Cohesion = 0, for all modules

$n = 52$

Table 8.5 Correlation among SP_BLAS Module Measurements

	Infosize	Infocomplexity	Infocoupling	Countingsize	Countingcomplexity	Countingcoupling
Infosize	1	0.955	0.694	0.999	0.680	0.680
Infocomplexity	0.955	1	0.725	0.951	0.706	0.706
Infocoupling	0.694	0.725	1	0.692	0.521	0.521
Countingsize	0.999	0.952	0.692	1	0.676	0.952
Countingcomplexity	0.679	0.706	0.521	0.676	1	0.709
Countingcoupling	0.680	0.706	0.521	0.676	0.709	1

Cohesion = 0, for all modules

$n = 95$

Table 8.6 and Table 8.7 shows the ratio of module information size to the corresponding counting size. From the tables we observe that module information size is often some multiple of the counting size. Ratios were analyzed to see if the measurements have identical patterns. Row patterns were verified to check the uniqueness of the row patterns. The following analysis was performed on the measured values to establish facts to show that row patterns are indeed unique.

The value of $n_{L(i)}$ gives the number of nodes with same row pattern. If $n_{L(i)} = 1$ the row patterns of nodes are unique. Using the Equation (3.12) we calculate the $n_{L(i)}$ from the known values of information size, $Size(m_k|\mathbf{S})$, and total number of nodes in the system, n. Consider a module m_k with one node, i. We know that the probability mass function $p_{L(i)}$ is estimated by

$$p_{L(i)} = \frac{n_i}{n+1} \tag{8.1}$$

where n_i is the number of nodes with the same pattern as node i. Expand Equation (3.12) by substituting for $p_{L(i)}$.

$$Size(m_k|\mathbf{S}) = -\sum_{i \in m_k} \log\left(\frac{n_{L(i)}}{n+1}\right) \tag{8.2}$$

For `Solvers`, we know n is 300, and $Size(m_k|\mathbf{S})$ is 8.234 for all the modules with one node (Table 8.6). Calculating $n_{L(i)}$, we get a simplified equation.

$$Size(m_1|\mathbf{S}) = \log(n+1) - \log n_{L(i)} \tag{8.3}$$

$$8.234 = \log 301 - \log n_{L(i)} \tag{8.4}$$

Table 8.6 Analysis of Information Size and Counting Size at Module Level for Solvers

moduleId	moduleinfosize	modulecountingsize	Ratio	SizeFraction
isolver.hxx	7.234	1	7.234	0.880
schwartz_additive.h	8.234	1	8.234	1.000
Solver<MT>	8.234	1	8.234	1.000
arrays_in_gmres.hxx	8.234	1	8.234	1.000
DirectSolver<MT>	8.234	1	8.234	1.000
ssor.h	8.234	1	8.234	1.000
SolverTrace<VT>	8.234	1	8.234	1.000
Preconditioner<MT>	8.234	1	8.234	1.000
saad.h	8.234	1	8.234	1.000
PreconditionerSchwartzAdditive<MT>	8.234	1	8.234	1.000
domain_decomposition_class.h	8.234	1	8.234	1.000
sor.h	8.234	1	8.234	1.000
ic_sqrt.h	16.467	2	8.234	1.999
block.h	16.467	2	8.234	1.999
gauss_seidel.h	16.467	2	8.234	1.999
schwartz_multiplicative.h	16.467	2	8.234	1.999
ict.h	23.116	3	7.705	2.800
choleski.h	23.701	3	7.900	2.870
preconditioner_identity.hxx	24.701	3	8.234	2.999
multigrid_parameters.h	24.701	3	8.234	2.999
direct_solver.hxx	24.701	3	8.234	2.999
ic0.h	24.701	3	8.234	2.999
schwartz_additive_coarse_grid.h	31.934	4	7.984	3.880
mict.h	32.934	4	8.234	3.999
sp_cgs.h	32.934	4	8.234	3.999
gauss_seidel_dns.h	32.934	4	8.234	3.999
qr.h	40.168	5	8.034	4.880
ilu_pivot.h	41.168	5	8.234	4.999
ldlt.h	41.168	5	8.234	4.999
gauss_seidel_sym.h	41.168	5	8.234	4.999
jac.h	41.168	5	8.234	4.999
ildu.h	48.402	6	8.067	5.880
GraphStorFormat	49.402	6	8.234	5.999
gauss_seidel_sym_dns.h	49.402	6	8.234	5.999
milut.h	56.050	7	8.007	6.800
ilu0.h	56.635	7	8.091	6.880
isolver_classes.hxx	56.869	8	7.109	6.900
dendy_black_box.h	64.869	8	8.109	7.880
ilut.h	72.518	9	8.058	8.800
PreconditionerSOR<MT>	73.103	9	8.123	8.880
aip.h	74.103	9	8.234	8.999
convergence.hxx	75.060	11	6.824	9.120
solver_trace.h	80.336	10	8.034	9.760
DomainDecomposition<MT>	80.336	10	8.034	9.760
PreconditionerSSOR<MT>	88.570	11	8.052	10.760
ic.h	97.803	12	8.150	11.880
ilu.h	105.037	13	8.080	12.800
lu.h	106.037	13	8.157	12.900
solver.hxx	107.037	13	8.234	12.999
ConvergenceCriterion<MT>	131.738	16	8.234	15.999
handles.hxx	146.205	18	8.123	17.756
ISolver<MT>	153.439	19	8.076	18.634

Table 8.7 Analysis of Information Size and Counting Size at Module Level for Graph

moduleId	moduleinfosize	modulecountingsize	Ratio	SizeFraction
sp_cgs.h	8.422	3	2.807	1
GraphStorFormat	8.422	3	2.807	1

$$\begin{aligned} 8.234 &= 8.234 - \log n_{L(i)}, \text{ where } \log 301 = 8.234 && (8.5) \\ \log n_{L(i)} &= 0 && (8.6) \\ n_{L(i)} &= 2^0 && (8.7) \\ n_{L(i)} &= 1 && (8.8) \end{aligned}$$

The value of $n_{L(i)}$ is one, because it must be an integer.

Theorem 1

Suppose **S** *has* n *nodes, and module* m_k *has* n_k *nodes. Suppose the row patterns of each row is unique in the* m_k*. Then*

$$Size(m_k|\mathbf{S}) = n_k \left(-\log \frac{1}{n+1}\right) \qquad (8.9)$$

Proof: Given a system **S**. Let n be the number of nodes in **S**. Index the nodes $i = 1, \ldots, n$.

By Equation (3.4), we know that information theory size for a module is

$$\mathit{Size}(m_k|\mathbf{S}) = \sum_{i \in m_k} (-\log p_{L(i)}) \qquad (8.10)$$

We know that the row patterns of all the nodes in m_k are unique.

$$p_{L(i)} = \frac{1}{n+1} \text{for all } i \in m_k \qquad (8.11)$$

Substituting this in Equation (8.10) we can further simplify.

$$Size(m_k|\mathbf{S}) = n_k \left(-\log \frac{1}{n+1}\right) \quad (8.12)$$

■

From the analysis we see that, in general, when the row patterns of the nodes are unique ($n_1 = ... = n_i = 1$), information size of a module is a multiple of the information size of a module with one node and the corresponding counting size. This accounts for the high correlation to counting module size. Results of this analysis prove that all row patterns of many modules were unique for the nodes in `Solvers`.

The complexity of both the systems tends to follow a curve. Figure 8.2 shows scatter plot representing information versus counting complexity values measured for `Solvers`. Complexity shows variation in the values. Analysis of the scatter plot suggest that the complexity measures have moderately high correlation values.

Figure 8.7, Figure 8.8, and Figure 8.9 shows the comparison between counting and information system measures for the three systems. Graph of cohesion is not included. Comparison of system level information and counting cohesion is not meaningful because the system level cohesion values for the systems were zero or some negligible.

8.3 Comparison of Metrics for Suitability in Software Quality Modeling

In this section we discuss the suitability of the metrics for software modeling by comparing information theory-based metrics and the counting-based metrics. The preliminary data collection and correlation analysis above provide us with the required information

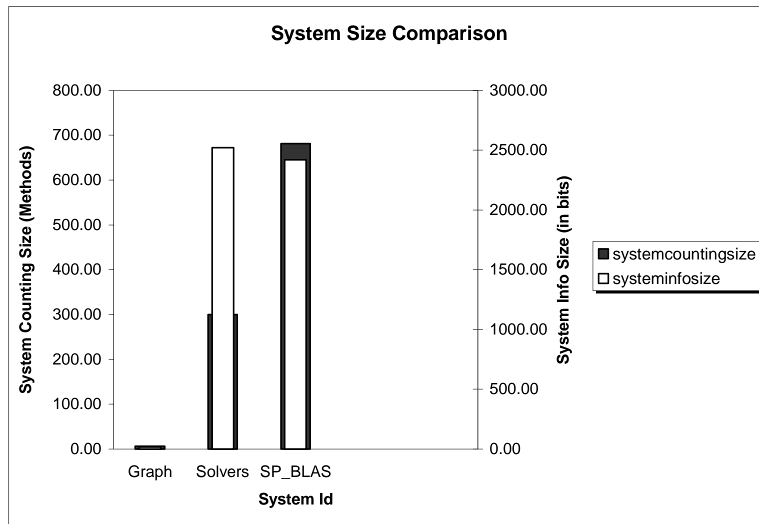

Figure 8.7 Comparison of Information *vs.* Counting System Size

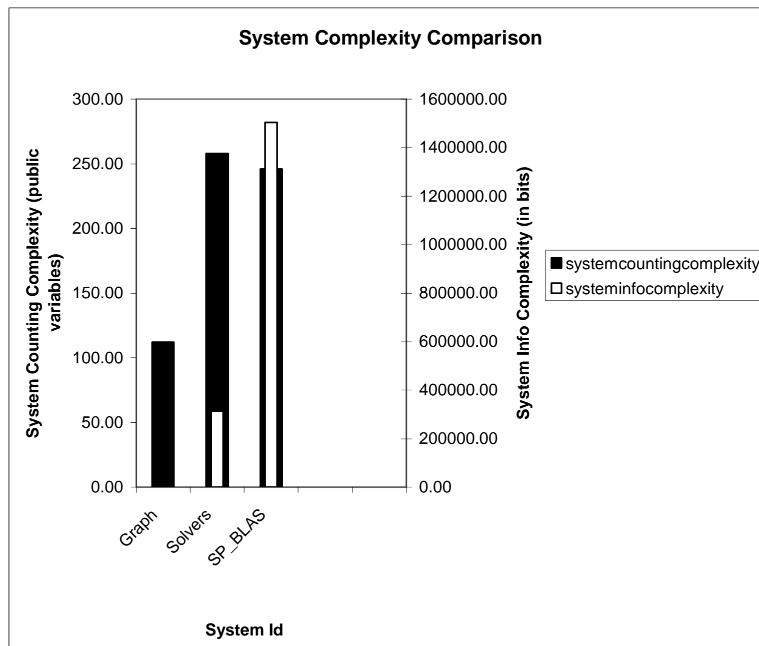

Figure 8.8 Comparison of Information *vs.* Counting System Complexity

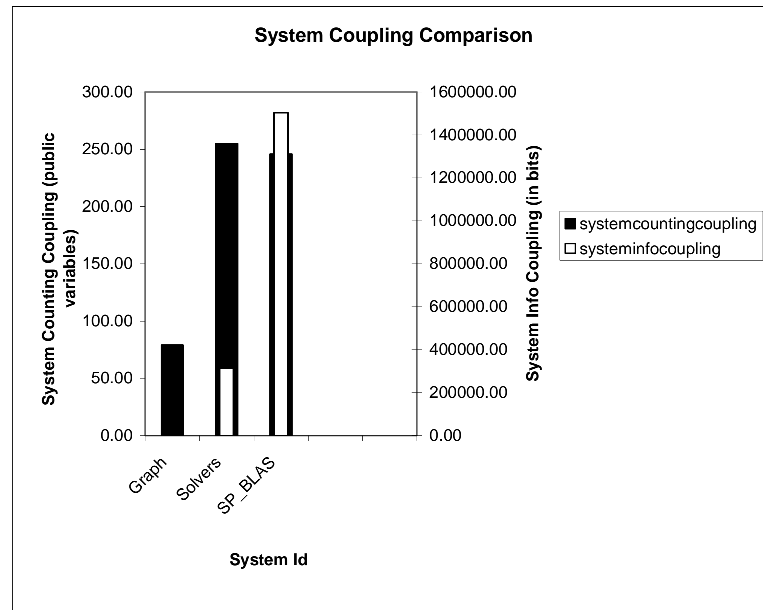

Figure 8.9 Comparison of Information *vs.* Counting System Coupling

necessary for analyzing the suitability of the metrics. Two important factors that affect the suitability of the metrics for software modeling are listed below.

- High correlation between the metrics used as independent variables, is not good for statistical modeling.
- If the range of distribution of a metric is very large or very narrow then it is not useful as a variable for modeling [20].
- A metric that is constant over all modules is not useful as an independent variable in modeling.

Careful examination of values led to some conclusions about the suitability of the measures for software quality modeling.

- Module information size has a very high correlation value compared to its corresponding counting size. So, statistical modeling of both size metrics as independent variables is not helpful with respect to our case study used. Counting-based size measure could be used instead of information theory-based size because the former is easier to collect.

- Module information complexity and module information coupling have moderate correlation with respect to the counting complexity and counting coupling. Correlation is not too high, so complexity and coupling are suitable for statistical software quality modeling.
- Range of the distribution is not extremely large nor constant for module measures, except cohesion. Cohesion has a zero or a negligible range, so statistical modeling using cohesion is not feasible in this case study. `Solvers` and `SP_BLAS` measured zero cohesion values for all the modules.

8.4 Insight into Software Development Processes

This section discusses various findings of the research and the resultant insights gained from these findings. Some of the findings are discussed in the coming sections.

8.4.1 Analysis of Zero Cohesion

The study of zero cohesion in PMLP led to some valuble insights about the software. Zero cohesion is undesirable for the software systems. A higher value of cohesion and a lower coupling value is what is desired for an ideal software from a software engineering point of view. Higher coupling and zero cohesion suggest a poor design in the software. A high coupling value indicates that the module has a high dependency on other modules. Zero cohesion contradicts the purpose of PMLP. PMLP is designed to be highly reliable software incorporating compile-time polymorphism and other object-oriented concepts applied in a manner that reduces the interdependencies among the modules. So theoretically PMLP should have a higher cohesion and a lower coupling value.

From the measurement of `Solvers` system and some directories of `SP_BLAS`, we observed that these systems have very low cohesion and a high coupling value. This does not meet the usual criteria for well designed software, that is, high cohesion and a low coupling value. However, the abstraction we are looking at gives a different perspective.

From the object oriented programming point of view we know that variables are declared public in order to allow access to the variable from any class. A class is considered a "module", and intermodule hyperedges are the public variables accessed by at least two different classes. Intramodule hyperedges are public variables that are accessed only by methods in the same class. These variables would better be declared as private. Therefore, we expect competent programmers to always have zero cohesion for the abstraction considered in this research.

The measured values for `Solvers` and `SP_BLAS` had significant coupling and zero cohesion. This is what is expected. `Graph` had two modules with low cohesion value which may be due to few variables defined public in order to make future enhancements.

It can be seen from Table 7.6, Table 7.5, Table 7.8, and Table 7.9 that the values of complexity and coupling are often the same. This is because almost all of the hyperedges related to a module are intermodule hyperedges. So, the value of complexity is often the same as the corresponding coupling value.

8.4.2 Analysis of High Correlation between Size Metrics

We found that information size and counting size are highly correlated (see Table 8.4). High correlation between metrics is not good for modeling when both information size and counting size are used as independent variables in the same model. Information module size is more difficult to collect than the counting module size. If one has to choose, counting module size is preferred for modeling over information module size. The following steps would give us other insights about the design of a system.

1. Identify modules whose information size to counting size ratio is not equal to the information size of a module with one node (See Table 8.6, Table 8.7).
2. Identify modules or classes that have rows or methods with non-unique patterns.
3. Identify the non-unique row patterns of the methods in the modules.
4. Identify methods that have same row patterns that is, methods that make use of the same public variables.
5. Refactor methods and modules identified in the system to improve the design. Our analysis found few methods in the case study that would be candidates for redesign. The following are example design options.

    - Public variables can be changed to private variables by combining the methods that have same row patterns.
    - Public variables may be a communication mechanism between methods, which when redesigned to use a different mechanism, such as call by reference or message passing.
    - Encapsulating public variables in accessor methods so that they are private.

CHAPTER IX

ANALYSIS OF RESEARCH QUESTIONS

The research takes into consideration several aspects of usefulness, including suitability for statistical software quality modeling, insight into development processes, and conformance of the metrics to intuitions about the attributes. The following section attempts to answer the identified research questions with the help of the analysis performed on the case study.

9.1 Research Questions

This section answers the research questions identified as a part of the analysis. There are two important questions to be answered. The first question addresses the similarities and differences that may exist between the two metric groups. The second question attempts to address our intuition about the attributes.

Question 1: What are the similarities and differences between the distributions of information theory-based metrics and counting-based metrics?

Similarities and differences are characterized by answering the following questions.

Question 1.a:Which metrics are more suitable for statistical software quality modeling?

Metrics that are not highly correlated may be suitable for the statistical software quality modeling. From the analysis we found that modeling using both size measures at a module level is not very helpful as they are highly correlated, more than $0.90 \approx 1$. High correlation confirms Gottipati's findings that high correlation between size measures means the two metrics may measure similar attributes [17]. Therefore, statistical quality modeling of both information and counting size is not advisable in the same model. In other words, one can use either information metrics or counting metrics for modeling of size but not both.

However, the complexity and coupling measures of both information theory-based and counting-based measures at module level are moderately correlated. Therefore, statistical software quality modeling of complexity and coupling is suitable with respect to the case study analyzed.

As far as cohesion is concerned, statistical modeling is not possible for this case study, as cohesion had a zero or negligible value.

Question 1.b: Do the distributions of measurement values yield insight into the software development process and resulting product attributes?

The distribution of measurement values yielded valuable insights into the software development processes. It was found during the analysis of PMLP that the systems measured

had a zero or negligible cohesion value. Zero cohesion is undesirable for the system, so this condition was analyzed. Analysis of zero cohesion led to some insight into the reason for this oddity. For the abstraction considered, namely the class-method-public variable relationship, high coupling and low cohesion are what is expected. In contrast, for this abstraction high cohesion values means variables are declared as public and are not accessed anywhere outside the scope of the class in which they are defined. This constitutes a poor design. A high coupling value obtained for systems shows that PMLP has a good design. This condition is discussed in detail in Chapter VIII.

Analysis of negative information module complexity identified by Gottipati [17] gave more insights into the development process. It was found that Gottipati's conjecture [17] is not always true. The analysis led to the addition of two more rules that further constrained the conditions for negative information module complexity to occur (Govindarajan's conjecture). From the analysis we were able to conclude that negative information module complexity is highly unlikely in real-world systems.

Question 1.c: Are the information theory and counting metrics highly correlated to the traditional software metrics obtained from the Datrix?

It was found during the analysis that Datrix is not suitable for the measurement of templates in PMLP. Therefore, Datrix was replaced by CPPX. CPPX cannot measure traditional software metrics, and so the correlation analysis of information theory-based metrics and counting-based metrics to the traditional software metrics was not performed in

this case study. However, correlation analysis of some of the counting metrics could indirectly correlate the two metric groups to some of the traditional software metrics produced by Datrix. For example, Datrix measures the number of final methods in the class `ClaMetFinNbr` and the number of methods in the class `ClaMetNbr`. Similarily, Datrix measures the number of public methods in the class `ClaMetPubNbr` [6]. `ClaMetNbr` is equivalent to the number of nodes measured in counting metrics and counting module size.

$$\textit{ClaMetNbr} = \textit{CountingModuleSize}(m_k|\mathbf{S}) = n_k \tag{9.1}$$

In the abstraction used in the research, nodes are the methods, and classes are the modules. From the results of the analysis we can say that the information theory module size is highly correlated to this traditional metric measured by Datrix.

Datrix also measures the number of public variables `ClaAttPubNbr` defined in a module (class) [6].

$$\textit{ClaAttPubNbr} = \textit{CountingModuleComplexity}(m_k|\mathbf{S}) = n_{e_k} \tag{9.2}$$

Question 1.d: Are the ranges of distributions of metrics suitable for software quality modeling?

Khoshgoftaar et al. [20] have reported that if ranges of distributions of the metrics are not constant or are extremely large (on the order of 10^{32} or more), statistical software quality modeling is not practical. From the summary statistics measured for the `Graph`, `SP_BLAS` and `Solvers` systems in PMLP (Table 8.1, Table 8.3, and Table 8.2), it is

seen that the ranges of distribution are neither constant nor extremely large. Therefore, statistical modeling of the module information measures and module counting measures is possible in this respect.

Question 2: Does each information theory-based metric and the counting-based metric preserve our intuition about the attribute it purports to measure?

Our main intuition about the attributes measured are their conformance to the properties proposed by Briand, Morasca, and Basili [12]. The explanations to the following questions address Question 2.

Question 2.a: Does each of the metrics conform to the properties proposed by Briand, Morasca, and Basili [12]?

Each of the metrics measured in this research conforms to the properties proposed by Briand, Morasca, and Basili [12], except for module information complexity. Module information complexity can be a negative value for certain row patterns.

Question 2.b: Is Gottipati's conjecture [17] true regarding conditions when information module complexity is negative?

Gottipati's conjecture [17] is not always true regarding conditions when information module complexity is negative. The analysis led to the addition of two new rules that constrained the conditions for the occurence of negative information module complexity. However, it was found that the likelihood of occurence in real-world systems is almost nil.

None of the case studies measured a negative module information complexity. A detailed analysis of the condition can be found in Chapter IV.

Question 2.c: What is a definition of counting cohesion for a hypergraph system that will satisfy the properties of Briand, Morasca, and Basili [12, 17]?

Gottipati's analysis [17] suggested the existence of a counting cohesion value greater than one. This condition violates the properties proposed by Briand, Morasca, and Basili [12, 17]. After careful analysis of this condition, the formulas of counting cohesion of system and of a module were revised. This redefinition of counting cohesion at both system level and module level led to the conformance of counting cohesion to the properties. The revised definition of counting cohesion for a hypergraph at system and module levels is discussed in Chapter III.

Question 2.d: What are the measures of counting complexity and counting coupling that correspond to the revised definition of the counting cohesion?

The formulas for counting complexity and counting coupling were not revised.

9.2 Threats to Validity

This section discusses the possible unexpected sources of biases and limitations of the study. There are two key things that will be discussed in this section. First, threats to internal validity will be discussed. Then threats to the external validity of the study will be discussed.

9.2.1 Threats to Internal Validity

Internal validity relates the extent to which the design and analysis may have been compromised by the existence of biases in the study [23]. Assumptions were made during the development of tools regarding the interpretation of the abstract semantic graph generated by CPPX. The following are possible threats to internal validity.

- Templates are considered a class in this study.
- In a file containing unclassed methods, the name of the file is used as a module, and all the unclassed methods in the file are assumed to be in the scope of this module.
- Relationships in the header files are analyzed by measuring test programs that include the headers of interest.
- CPPX represents the occurence of entities in the program by name, line number of the entity, name of the file to which the entity is associated, and visibility (private, public, or protected). The scope of the public variable used inside a method is obtained by comparing the line number of the method with the line number of the public variables used.

9.2.2 Threats to External Validity

External validity relates the extent to which the hypotheses capture the objectives of this research and the extent to which any conclusions can be generalized [23]. Conclusions of this study are limited to a single case study, namely PMLP. The generalizations made in this study are based on this one case study.

- PMLP consists of scalable libraries that combine the features of object-oriented design, and sequential and parallel modes.
- PMLP was developed mostly in C++ using object-oriented techniques, such as template classes, generic programming, parametrized types, run-time polymorphism, compile time polymorphism, and iterators.
- The major part of PMLP was coded as header files.

- This study analyzed C++ source code and did not address non-C++ components of PMLP.
- The relationship between public variables and methods is the only abstraction measured in this study.
- A system is defined as a collection of files performing tasks that are related to each other.
- A module is an occurence of a class in a system, or the definition of a template.
- A node is a method defined inside the scope of a module.

CHAPTER X

CONCLUSIONS

10.1 Evaluation of the Hypothesis

The hypothesis of the research is

> Information theory-based metrics proposed by Allen [3] (size, complexity, coupling, and cohesion) can be useful for large codebases in real-world software development projects, compared to corresponding counting-based metrics [17].

This research was conducted using PMLP as a case study. The analysis of research questions in Chapter IX provides some evidence for the hypothesis.

Question 1: What are the similarities and differences between the distributions of information theory-based metrics and counting-based metrics?

This question is addressed by explanations to the following questions.

Question 1.a:Which metrics are more suitable for statistical software quality modeling?

Information theory-based and counting based module measures for complexity and coupling are suitable for statistical software quality modeling. Statistical modeling of either information theory-based size or counting size is possible because of a high correlation between them. That is, a statistical quality model should not contain them both.

Question 1.b: Do the distributions of measurement values yield insight into the software development process and resulting product attributes?

The distributions of measurement values yield insight into the software development process and the resulting product attributes such as the analysis of zero cohesion and reasons for very high coupling values. Analysis also revealed insights into the negative information module complexity issue identified by Gottipati [17].

Question 1.c: Are the information theory and counting metrics highly correlated to the traditional software metrics obtained from Datrix?

Datrix is not suitable for the measurement of templates in PMLP. Therefore, Datrix was replaced by CPPX. CPPX cannot measure traditional software metrics, and so the correlation analysis of information theory-based metrics and counting-based metrics to the traditional software metrics was not performed in this case study. However, the counting size measure is equivalent to `ClaMetNbr` measured by Datrix and the counting module complexity is equivalent to `ClaAttPubNbr`.

Question 1.d: Are the ranges of distributions of metrics suitable for software quality modeling?

Metric distributions of modules are not extremely large. Hence information theory-based and counting-based module measures for size, complexity, and coupling are suitable for statistical software quality modeling. As the measured values of cohesion had zero or

some negligible value, statistical modeling of cohesion would not be possible using this case study.

Question 2: Does each information theory-based metric and the counting-based metric preserve our intuition about the attribute it purports to measure?

Each of the information theory-based metrics and the counting-based metrics preserves our intuition about the attributes. However, negative information module complexity could occur in certain circumstances. Negative information module complexity is highly unlikely to occur in real-world systems.

Question 2.a: Does each of the metrics conform to the properties proposed by Briand, Morasca, and Basili [12]?

Each of the metrics, except module information complexity, conforms to the properties proposed by Briand, Morasca, and Basili [12].

Question 2.b: Is Gottipati's conjecture [17] true regarding conditions when information module complexity is negative?

Gottipati's conjectures are not always true. We propose a refined conjecture. The likelihood of occurence of negative information module complexity is almost nil in real-world systems. Analysis of negative information module complexity is discussed in detail in Chapter IV.

Question 2.c: What is a definition of counting cohesion for a hypergraph system that will satisfy the properties of Briand, Morasca, and Basili [12, 17]?

The revised definition of counting cohesion for a hypergraph system, the number of intramodule hyperedges to the total number of hyperedges in the graph of the system, satisfies the properties of Briand, Morasca, and Basili [12, 17]. The revised definitions are discussed in Chapter III.

10.2 Contribution

Contributions of this research are the following.

- The case study found information theory-based metrics of the relationships between public variables and methods that use them to be useful for a large codebase of a scientific mathematics library, compared to corresponding counting-based metrics. Information theory-based metrics provides with fine grained distinctions among the modules compared to the counting-based metrics.
- The case study found that counting and information size metrics are often highly correlated because the patterns of relationships are often unique for each method. Thus, a quality model can use either information or counting size as an independent variable.
- The case study found that each of the metrics measured conforms to the properties proposed by Briand, Morasca, and Basili [12], except module information complexity, which may be negative for certain hypergraph configurations. However the likelihood of negative module information complexity is almost nil in real-world systems.

The findings of this research are helpful to the research community as well as software engineers. Software engineers working on ensuring a reliable software can use the methodology discussed to measure their software products. Metrics are indicators of software quality, so software engineers can use these metrics to assess quality. Software engineers

can extend the methodology used to other programming languages with minor changes to the abstraction that will suit the language selected.

10.3 Future Work

This work can be extended to different levels of abstractions and using new abstractions. More case studies should be used to further evaluate the metrics and generalize facts about the metrics. Future work may mathematically prove our conjecture regarding negative information module complexity. Call graphs are an alternative to method-public variable relationships. The results of the work on call graphs in the future would provide us with more insights into the software development process. Statistical software quality modeling of the metrics could be done in case studies that collect past quality data.

REFERENCES

[1] E. B. Allen, *Empirical Validation of Information Theory-Based Software Metrics*, Proposal for National Science Foundation, Mississippi State University, Sept. 2001.

[2] E. B. Allen, *Information Theory-Based Measures of Graph Abstractions of Software*, Tech. Rep. MSU-010629, Mississippi State University, June 2001.

[3] E. B. Allen, "Measuring Graph Abstractions of Software: An Information theory Approach," *Proceedings: Eighth IEEE Symposium on Software Metrics*. June 2002, pp. 182–193, IEEE Computer Society.

[4] E. B. Allen and T. M. Khoshgoftaar, "Measuring Coupling and Cohesion: An Information theory Approach," *Proceedings of the Sixth International Software Metrics Symposium*, Boca Raton, Florida, Nov. 1999, pp. 119–127, IEEE Computer Society.

[5] E. B. Allen, T. M. Khoshgoftaar, and Y. Chen, "Measuring Coupling and Cohesion of Software Modules: An Information-Theory Approach," *Proceedings: Seventh International Software Metrics Symposium*, London, Apr. 2001, IEEE Computer Society, pp. 124–134.

[6] *DATRIX Metric Reference Manual Version 4.1*, Bell Canada, May 2000.

[7] L. C. Briand, J. Daly, V. Porter, and J. Wüst, "A Comprehensive Empirical Validation of Design Measures for Object-Oriented Systems," *Proceedings: Fifth International Software Metrics Symposium*, Bethesda, Maryland, Nov. 1998, pp. 246–257, IEEE Computer Society.

[8] L. C. Briand, J. W. Daly, and J. Wüst, "A Unified Framework for Cohesion Measurement in Object Oriented Systems," *Proceedings of the Fourth International Software Metrics Symposium*, Albuquerque, New Mexico, Nov. 1997, pp. 43–53, IEEE Computer Society.

[9] L. C. Briand, J. W. Daly, and J. Wüst, "A Unified Framework for Cohesion Measurement in Object Oriented Systems," *Empirical Software Journal: An International Journal*, vol. 3, no. 1, 1998, pp. 65–117.

[10] L. C. Briand, J. W. Daly, and J. Wüst, "A Unified Framework for Coupling Measurement in Object Oriented Systems," *IEEE Transactions on Software Engineering*, vol. 25, no. 1, Jan. 1999, pp. 91–121.

[11] L. C. Briand, K. El Emam, and S. Morasca, "On the Application of Measurement Theory in Software Engineering," *Empirical Software Journal: An International Journal*, vol. 1, no. 1, 1996, pp. 61–88.

[12] L. C. Briand, S. Morasca, and V. R. Basili, "Property-based Software Engineering Measurement," *IEEE Transactions on Software Engineering*, vol. 22, no. 1, Jan. 1996, pp. 68–85.

[13] L. C. Briand, S. Morasca, and V. R. Basili, "Response to: 'Comments on Property-Based Software Engineering Measurement': Refining Additivity Properties.," *IEEE Transactions on Software Engineering*, vol. 23, no. 3, Mar. 1997, pp. 196–197.

[14] L. C. Briand, S. Morasca, and V. R. Basili, "Defining and Validating Measures for Object-based High Level Design," *IEEE Transactions on Software Engineering*, vol. 25, no. 5, Sept. 1999, pp. 722–743.

[15] M. Ernst, G.H.Badros, and D. Notkin, "An Empirical Analysis of C Preprocessor Use," *IEEE Transactions on Software Engineering*, vol. 28, no. 12, Dec. 2002, pp. 1146–1170.

[16] S. L. Fox, *Evaluation of Open-Source Software for Empirical Software Engineering Research*, Tech. Rep. MSU031213, Mississippi State University, December 2003.

[17] S. Gottipati, *Empirical Validation of Usefulness of the Information Theory-Based Software Metrics.*, master's thesis, Mississippi State University, May 2003.

[18] R. Hochman, *Software Reliability Engineering: An Evolutionary Neural Network Approach*, master's thesis, Florida Atlantic University, Boca Raton, Florida, Dec. 1997.

[19] T. M. Khoshgoftaar, E. B. Allen, R. Halstead, G. Trio, and R. Flass, "Process Measures for Predicting Software Quality," *Proceedings of High-Assurance Systems Engineering Workshop*, Washington, DC, Aug. 1997, pp. 155–160, IEEE Computer Society.

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

[21] T. M. Khoshgoftaar, E. B. Allen, K. S. Kalaichelvan, and N. Goel, "The Impact of Software Evolution and the Reuse on Software Quality," *Empirical Sotware Engineering*, vol. 1, 1996, pp. 31–44.

[22] B. Kitchenham, S. L. Pfleeger, and N. Fenton, "Towards a Framework for Software Measurement Validation," *IEEE Transactions on Software Engineering*, vol. 21, no. 12, Dec. 1995, pp. 929–944.

[23] B. A. Kitchenham, S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, K. El Emam, and J. Rosenberg, "Preliminary Guidelines for Empirical Research in Software Engineering," *IEEE Transactions on Software Engineering*, vol. 28, no. 8, Aug. 2002, pp. 721–734.

[24] S. Morasca and L. C. Briand, "Towards a Theoretical Framework for Measuring Software Attributes," *Proceedings of the Fourth International Symposium on Software Metrics*, Albuquerque, New Mexico, Nov. 1997, pp. 119–126, IEEE Computer Society.

[25] S. Morasca, L. C. Briand, V. R. Basili, E. J. Weyuker, M. V. Zelkowitz, B. Kitchenham, S. L. Pfleeger, and N. Fenton, "Comments on "Towards a Framework for Software Measurement Validation"," *IEEE Transactions on Software Engineering*, vol. 23, no. 3, Mar. 1997, pp. 187–189.

[26] G. C. Murphy, D. Notkin, W. G. Grisworld, and E. S. Lan, "An Empirical Study of Static Call Graph Extractors," *ACM Transactions on Software Engineering and Methodology*, vol. 7, no. 2, Apr. 1998, pp. 158–191.

[27] G. Poels and G. Dedene, "Comments on 'Property-Based Software Engineering Measurement: Refining Additivity Properties'," *IEEE Transactions on Software Engineering*, vol. 23, no. 3, Mar. 1997, pp. 190–197.

[28] R. S. Pressman, *Software Engineering: A Practitioner's Approach*, third edition, McGraw-Hill Inc., 1992.

[29] N. F. Schneidewind, "Methodology for Validating Software Metrics.," *IEEE Transactions on Software Engineering*, vol. 18, no. 5, May 1992, pp. 410–422.

[30] M. V. Zelkowitz and D. R. Wallace, "Experimental Models for Validating Technology," *IEEE Computer*, vol. 31, no. 5, May 1998, pp. 23–31.

[31] H. Zuse, "Reply to: "Property-Based Software Engineering Measurement"," *IEEE Transactions on Software Engineering*, vol. 23, no. 8, Aug. 1997, p. 533.