Empirical Validation of the Usefulness of Information Theory-Based Software Metrics

Sampath Gottipati

EMPIRICAL VALIDATION OF THE USEFULNESS OF INFORMATION THEORY-BASED SOFTWARE METRICS

By

Sampath Gottipati

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 2003

EMPIRICAL VALIDATION OF THE USEFULNESS OF INFORMATION

THEORY-BASED SOFTWARE METRICS

By

Sampath Gottipati

Approved:

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

Rayford B. Vaughn
Associate Professor of Computer Science and Engineering
(Committee Member)

David A. Dampier
Assistant 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 College of Engineering

Name: Sampath Gottipati

Date of Degree: May 10, 2003

Institution: Mississippi State University

Major Field: Computer Science

Major Professor: Dr. Edward B. Allen

Title of Study: EMPIRICAL VALIDATION OF THE USEFULNESS OF INFORMATION THEORY-BASED SOFTWARE METRICS

Pages in Study: 96

Candidate for Degree of Master of Science

Software designs consist of software components and their relationships. Graphs are abstraction of software designs. Graphs composed of nodes and hyperedges are attractive for depicting software designs. Measurement of abstractions quantify relationships that exist among components. Most conventional metrics are based on counting. In contrast, this work adopts information theory because design decisions are information.

The goal of this research is to show that information theory-based metrics proposed by Allen, namely size, complexity, coupling, and cohesion, can be useful in real-world software development projects, compared to the counting-based metrics. The thesis includes three case studies with the use of global variables as the abstraction. It is observed that one can use the counting metrics for the size and coupling measures and the information metrics for the complexity and cohesion measures.

DEDICATION

I dedicate this thesis to my parents and sisters.

TABLE OF CONTENTS

LIST OF TABLES

TABLE Page

TABLE Page

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Criteria such as construct validity, internal validity and external validity can be considered to evaluate the quality of any empirical study [26]. “Threats to internal validity are unaccounted influences that may affect case study results” [2]. In practice, faults are caused by a wide variety of conditions. The number of faults in each module may be due a number of things that were not measured. Using a variety of independent variables in each model is a strategy to avoid threats to internal validity. “Threats to external validity are conditions that limit generalization of results” [2]. As experiments to demonstrate the usefulness of software metrics are not feasible, we use a case study approach.

The hypothesis of this research is:

> Information theory-based software metrics proposed by Allen [3], namely, size, complexity, coupling, and cohesion, can be useful in real-world software development projects, compared to counting-based metrics.

This research is motivated by the hope that information metrics are more useful than counting metrics. The following research questions, whose answers provide evidence for the hypothesis, are answered in Chapter VI using three case studies:

1. What are the similarities and differences between the distribution of information theory-based metrics and counting-based metrics?
2. Do the distributions of measurement values yield insight into the software development process and resulting product attributes?

3. Does each information theory-based measure preserve our intuition about its attribute?
4. Does the measurement instrument (tool) precisely specify how to capture measurement data?
5. Does the measurement protocol (procedure) assure consistent, repeatable measurements that are independent of the measurer and the measurement environment?

Very often the software community discusses designs in terms of size, length, complexity, coupling, cohesion, etc. The designs are attractively depicted by graphs that are widely used in the software industry. Briand, Morasca, Basili [11] proposed definitions for the attributes (size, length, complexity, coupling, and cohesion) based on graphs and later extended their framework from graphs to relations in general [20]. Most of their metrics are based on counting. In contrast, this work adopts information theory as a foundation because design decisions embodied by a graph abstraction of software are information [3].

The field of software metrics embraces collection, analysis and modeling of measurements of software [13]. It refers to a broad range of measures for software engineering. Fenton and Pfleeger [13] say that a software metric is a quantitative measure of the degree to which a system possesses a given attribute. If we are able to regularly collect software metrics, then we have a way of tracking project process, measuring complexity, knowing if we have reached a desired state of quality, etc.

Software metrics are recognized in broad categories: processes, products, resources, and quality. "Process" refers to any software-related activities that normally have a time factor [14]. A process can be any part of the software development cycle, from requirements to retirement. Products can be defined as any artifacts, deliverables, or documents that arise out of the processes [14]. Products include specification and design documents at various levels of detail. Resources are the items that are input to processes [14]. Attributes can be classified as internal attributes and external attributes. Internal attributes are those that can be measured in terms of the entity itself. For example, size, modularity, reuse, and functionality are internal attributes. External attributes are those that can be measured with respect to how the entities relate to their environment, e.g., usability and maintainability. Without a measurable definition of software product quality, no quantitative approach to software quality can be complete. Moreover, we need quality measures if we are to improve our product. Software metrics tell us about the quality of a software product. Quality metrics are a subset of metrics measuring external attributes.

We can classify the main classes of metrics into subclasses. Process metrics can be subdivided into maturity metrics, management metrics, and life cycle metrics. For product metrics the division is size metrics, architecture metrics, structure metrics, and complexity metrics. Resource metrics can be divided into personnel metrics, software metrics, and hardware metrics. Measurements require us to identify attributes possessed by clearly defined entities [14]. Direct measurement of an attribute must be preceded by intuitive understanding of that attribute, which leads to the identification of relationships between entities.

Allen and Khoshgoftaar [6] proposed information theory-based measures of coupling and cohesion of graphs at the system level. Allen, Khoshgoftaar, and Chen [7] later proposed information theory-based measures of coupling and cohesion of graphs at the module level. The paper by Allen [3] proposes related additional measures of size, length, complexity, as well as revised measures of coupling and coupling at the system and module levels. The research shows that the information theory-based metrics proposed by Allen [3] can be useful in real-world software development projects, compared to counting-based metrics. Table 1.1 summarizes the metrics to be compared in each family [2].

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 intramodule relationships divided by maximum possible	Number of intramodule edges divided by maximum possible

The development process details determine the set of abstractions that are likely to be related to faults. The thesis outlines the tasks to be accomplished and analyzes the steps to evaluate module-level metrics and system-level metrics. Information theory-based metrics are compared with counting-based metrics of size, length, complexity, coupling and cohesion. The remainder of the thesis summarizes the related work, definition of metrics, methodology, tools, and results.

CHAPTER II

RELATED WORK

Briand, Morasca and Basili [11] proposed a mathematical framework to define several important measurement concepts (size, length, complexity, coupling, and cohesion). In their paper they refer to a paper by Parnas, who recommends decreasing coupling between modules and increasing cohesion within modules. Coupling and cohesion can be used as guides for choosing among alternative techniques or artifacts. The goal of Briand, Morasca and Basili's paper is to provide properties for a partial set of concepts that are relevant in measurement of internal software attributes, which are most commonly found in software engineering literature. The investigation of measures may also address artifacts other than code that are produced in the software process. Early phases of the software development process produce artifacts, upon which the rest of the development depends. Concepts that are relevant with respect to code are also relevant to other artifacts. In their paper, Briand, Morasca and Basili [11] investigate size, length, and complexity related to systems in general, and coupling and cohesion related to modular systems. One can speak about coupling and cohesion of a whole system only if it is structured into modules. The properties of each attribute except the length are paraphrased in Chapter III, and the concept of modularity was also employed. In the case studies we considered a class to be a module.

Morasca and Briand [20] provide an axiomatic approach for the definition of measures of software attributes in two ways: (i) they generalize the framework by considering n-ary relationships between system and module elements, and (ii) they propose a hierarchical axiomatic framework where hierarchical levels map to levels of measurement. The axiomatic approaches can be combined with the theory of measurement scales so that, depending on the level of empirical understanding of the attribute, one can select an appropriate level of measurement and a suitable axiomatic framework [20]. They also discuss a variety of abstractions, but we have used the use of global variables as the abstraction. They have used ordinary edges as relations to show the relationship between elements, while we have used hyperedges to show the relationship.

Poels and Dedene [23] contribute to a formal and rigorous approach to property-based software-engineering measurement because a number of inconsistencies related to additivity properties might hinder its acceptance and further elaboration. Poels and Dedene [23] show how to remove ambiguity by introducing the concept of connection strength between systems and modules. In the case study of artificial examples in Section 6.1, the additivity property did not show any ambiguity.

It is difficult to determine how measures relate to one another and for which application [9]. Briand, Daly, and Wüst [9] discuss a unified framework based on object-oriented cohesion measures for (i) comparing measures and their potential use, (ii) integrating existing measures that examine the same concepts in different ways, and (iii) facilitating more rigorous decision making regarding the definition of new measures and the selection of existing measures for a specific goal of measurement. They also explain that some proposed metrics do not satisfy the properties of coupling and cohesion defined by Briand, Morasca and Basili [11]. Our metrics satisfy the properties and are paraphrased in Chapter III.

Coupling measurement in object-oriented systems requires a comprehensive framework that can be used to facilitate comparison of existing measures, evaluation and empirical validation of existing measures, and to support definitions of new measures [10]. Briand, Daly, and Wüst [10] provide a standard terminology and formalism for expressing measures, a structural synthesis, a review of the existing framework and measures for coupling in object-oriented systems. The properties of coupling are shown in Chapter III.

Briand, Daly, Porter, and Wüst [8] discuss the fact that many of the coupling, cohesion, and inheritance measures studied in the literature appear to capture structural dimensions in the data. They empirically explore the relationships between existing object-oriented coupling, cohesion, and inheritance measures and the probability of fault detection in system classes during testing [8]. They found that frequency of method invocation and depth of inheritance hierarchically seem to be the driving factors of fault-proneness. Since data was not available for us to do a similar case study, the factor of fault-proneness will be investigated by future work.

"A graph composed of nodes and edges may be an abstraction of a software system and a subgraph may represent a module" [3]. In contrast to software measures based on counting, Allen has focused his research by adopting information theory because the design decisions embodied by a graph abstraction of software are elements of information. Allen and Khoshgoftaar [5] proposed an information theory-based measure of cohesion on graphs for application to software design. Cohesion summarizes the degree of interdependence or connectivity within subsystems [5]. Allen and Khoshgoftaar [6] later proposed information theory-based measures of coupling and cohesion of a modular system. These measures have the properties of system-level coupling and cohesion defined by Briand, Morasca and Basili [6]. Allen and Khoshgoftaar [6] also proposed coupling based on an intramodule abstraction, calculated in the same way as intermodule coupling, and then defined cohesion in terms of intramodule coupling, normalized to between zero and one [6]. Allen, Khoshgoftaar, and Chen [7] further proposed information theory-based measures of coupling and cohesion of a module, which have the properties of module-level coupling and cohesion defined by Briand, Morasca, and Basili.

Allen [3] extended this line of research and discusses information theory-based measures on graphs at the system level and module level for each family of metrics defined by Briand, Morasca, and Basili. The primary objective of his research is to provide empirical evidence that innovative software metrics based on information theory are indeed useful as predictors of software quality. The definition of each metric is shown in Chapter III. This study incrementally builds on the work done by Allen [3].

Kitchenham, Pfleeger, and Fenton [18] propose a framework for theoretically validating software measurement by defining a measurement structure model, measurement process, and five other models involved in measurement. The framework can help to understand how to validate a measure, how to assess the validation work of others, and when to apply a measure. They point out that measurement validation is required for pragmatic as well as theoretical reasons based on discussion of function points [18]. This paper provides criteria for answering the research question addressed in Chapter VI.

Schneidewind [24] illustrates a comprehensive empirical metrics-validation methodology having six validity criteria, which support the quality functions of assessment, control, and prediction. Such empirically validated metrics can be a basis for making decisions and taking actions to improve quality of software. He also shows that nonparametric statistical methods play an important role in evaluating whether metrics satisfy the validity criteria [24]. This paper is related to measuring the factor of fault-proneness, which will be a study of future work.

As an example of an empirical validation study, Briand, Morasca, and Basili [12] introduce and compare various high-level design measures for object-based software systems based on experimental goals, identifying fault-prone measures and several experimental hypotheses. They state that these measures allow for early detection of problems, better software quality monitoring, and more accurate planning of resource utilization [12]. Briand, Morasca, and Basili also show that models of good statistical significance can be built based on high-level design information for systems designed on abstract data types [12]. The fault-prone measure will be investigated by future work.

The research reported by this thesis builds on the above by empirically validating information theory-based metrics for size, complexity, coupling, and cohesion, defined by Briand, Morasca and Basili [11] both at system level and module level. Measures of length are deferred to future research.

CHAPTER III

DEFINITION OF METRICS

Table 3.1 and Table 3.2 taken from [3] provide the definition of symbols and notation used in the later part of this chapter. A system is an abstraction of a software development artifact, defined by a set of elements and a relation on them [20]. We restrict this abstraction to a hypergraph consisting of nodes and hyperedges. Each node corresponds to an element, and each hyperedge corresponds to a relationship among a subset of nodes. The word “label” in Table 3.2 refers to the set of incident edges for a node. An environment node is a disconnected node that represents the enviroment. We form a system graph S for calculation of the metrics by adding the environment node to the system model **S**. The probability mass function p for each node is estimated by the number of occurances of the row pattern divided by the number of nodes plus the environment node ($n+1$). The binary row pattern is generated by identifying whether a node is associated to each hyperedge, and encoding a “1” or a “0” accordingly.

Table 3.1 Symbols

Symbol	Name	Definition
S	System	Abstraction of software (nodes and hyperedges)
$S^{\#}$	Hyperedges-only graph	Hyperedges in **S** and end points
S_i	Node subgraph	Nodes in $S^{\#}$ and hyperedges incident to node i
MS	Modular system	**S** partitioned into modules
m_k	module k	Nodes in a module and their incident hyperedges
$MS^{\star}$	Intermodule hyperedges graph	Nodes in *MS* and intermodule hyperedges
MS°	Intramodule hyperedges graph	Nodes in *MS* and intramodule hyperedges
S	System graph	**S** plus environment node, represented by nodes $\times$ hyperedges table
$MS^{(n)}$	Complete graph	Complete graph with n nodes in one module

3.1 Properties of Measures of Hypergraphs

Table 3.3 and Table 3.4 summarize the properties of any measure of the size of a system and the size of a module that Briand, Morasca,and Basili [11] proposed. These properties define the concepts of the size of a system and the size of a module.

Table 3.5 and Table 3.6 summarize the properties of the measure of complexity of a system and complexity of a module that Briand, Morasca, and Basili [11] proposed. These properties define the concepts of the complexity of a system and complexity of a module. There is a change in the system property 4, module monotonicity, in Table 3.5. It is related

Table 3.2 Notation

Symbol	Definition
n	The number of nodes in system, **S**.
n_M	The number of modules in *MS*.
n_k	Number of nodes in the module, m_k.
i, j	Indexes for row in S, $i = 0,\ .., n$ and similarly j.
k	Index for a module in **S**, $k = 1,\ ..., n_M$.
	Index for a pattern of values on a row.
(i)	A function that determines the label of a row.
$_i(j)$	A function that determines the label of a row j in S_i
p	Probability mass function.
$\log$	Logarithm, base 2.
	Entropy of a probability distribution.
n_e	Number of hyperedges in system, **S**.
n_{e_k}	Number of hyperedges incident to nodes in module m_k
$n_{intr\ \_e}$	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_{ec}	Number of hyperedges in a complete graph of a system, **S**.
n_{ecm}	Number of hyperedges in a complete graph of module, m_k.

Table 3.3 Properties of the 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.

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

Table 3.4 Properties of the Size of a Module

1. Nonnegativity. The size of a module is nonnegative.
2. Null value. The size of the module is null if its set of nodes is empty.
3. Monotonicity. Adding a node to a module does not decrease its size.

to nodes rather than edges as defined by Briand, Morasca, and Basili [11]. This change makes Property 4 unnecessary because Property 5 is a stronger version.

Table 3.7 and Table 3.8 summarize the properties of the measure of coupling of a modular system and coupling of a module, respectively, that Briand, Morasca, and Basili [11] proposed. These properties define the concepts of coupling of a system and coupling of a module.

Table 3.9 and Table 3.10 summarize the properties of the measure of cohesion of a modular system and cohesion of a module, respectively, that Briand, Morasca, and Basili [11] proposed. These properties define the concepts of cohesion of a system and cohesion of a module.

Table 3.5 Properties of the 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.

   Complexity(**S**)$\geq$ *Complexity*$(m_1|\mathbf{S})$ + *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.

   Complexity(**S**) = *Complexity*$(\mathrm{m}_1|\mathbf{S})$ + *Complexity*$(m_2|\mathbf{S})$

Table 3.6 Properties of the Complexity of a Module

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

2. Null value. The complexity of the 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 3.7 Properties of Coupling of a Modular 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, $m_{1\ 2}$, that replaces m_1 and m_2, then the coupling of the modular system with $m_{1\ 2}$ 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, $m_{1\ 2}$, that replaces m_1 and m_2, then the coupling of the modular system with $m_{1\ 2}$ is equal to the coupling of the modular system with m_1 and m_2.

Table 3.8 Properties of Module Coupling

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, $m_{1\ \ 2}$, that replaces m_1 and m_2, then the module coupling of $m_{1\ \ 2}$ 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, $m_{1\ \ 2}$, that replaces m_1 and m_2, then the module coupling of $m_{1\ \ 2}$ is equal to the sum of the module coupling of m_1 and m_2.

Table 3.9 Properties of Cohesion of a Modular System

1. Nonnegativity and Normalization. Cohesion of a modular system belongs to a specified interval, *Cohesion*( ) $\in$ 0, *Max* . S
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, $m_{1\ \ 2}$, that replaces m_1 and m_2, then the cohesion of the modular system with $m_{1\ \ 2}$ is not greater than the cohesion of the modular system with m_1 and m_2.

Table 3.10 Properties of Module Cohesion

1. Nonnegativity and Normalization. Cohesion of a module belongs to a specified interval, *Cohesion*$(m_k|\quad) \in\ 0, $*Max* . S

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, $m_{1\ \ 2}$, that replaces m_1 and m_2, then the module cohesion of $m_{1\ \ 2}$ is not greater than the maximum of the module cohesion of m_1 and m_2.

3.2 Information Theory-Based Metrics Definitions

Shannon's paper [25] lays the foundation of information theory. For a discrete random variable, x, distributed according to a probability mass function, p, entropy is defined as

$$(x) = \sum_{=1}^{n_x} p\ (-\log p\ ) \tag{3.1}$$

where is an index over the domain of x, and n is the cardinality of the domain of x. (x) is interpreted as the average information per sample from the distribution of x [2]. The logarithms are to the base two, thus the unit of measure is a bit. In this application, entropy of the distribution of the row patterns is the average information per node.

According to van Emden [27]

> Excess-entropy is the difference between the sum of the entropies taken separately and the entropy of the predicates together. Excess-entropy would be

zero in the case where there is no interaction at all between predicates and the system of all predicates is trivially simple. When excess-entropy is greater than zero, there is interaction between the components, which can be regarded as evidence of complexity.

For m random variables, $x_1,.\ \ ,x\ \ $, excess-entropy is defined as

$$C(x_1,.\ \ ,x\ \ ) = \sum_{i=1} \ \ (x_i) - \ \ (x_1, ..., x\ \ ) \tag{3.2}$$

where $\ \ (x_1,.\ \ ,x\ \ ) = -\sum p(x_1,.\ \ ,x\ \ ) \log p(x_1,.\ \ ,x\ \ )$ summed over all combinations of values of x_i [1].

The following are the definitions of information theory-based metrics taken from [3].

3.2.1 Size of a System

The size of the system **S** is given by the amount of information in its system graph S, less the contribution of the environment node.

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

Note that by convention the environment node corresponds to $i = 0$. Allen [3] derives this formula from Equation (3.1) and therefore this is an information theory-based metric.

3.2.2 Size of a Module

Size of module m_k, in a System **S**, is its contribution to the system's size, given as

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

3.2.3 Complexity of a System

Complexity of a system is the amount of information in relationships in its edges-only graph, less the contribution of the environment. Complexity is based on the concept of excess entropy [4].

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

3.2.4 Complexity of a Module

Complexity of a module m_k, in a system **S**, is its contribution to the complexity of the system, given by

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

3.2.5 Coupling of a Modular System

Coupling of a modular system *MS* is the amount of information in intermodule relationships in its system graph, less the contribution of the environment.

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

3.2.6 Coupling of a Module

Coupling of a module m_k, in a modular system *MS*, is its contribution to the coupling of the system, given by

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

3.2.7 Cohesion of a Modular System

Cohesion of a modular system *MS*, with n nodes, is the proportion of information in a complete system graph due to intramodule relationships.

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

3.2.8 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 module relationships of m_k.

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

3.3 Counting-Based Metrics Definitions

The following are the definitions of counting metrics taken from [2]

3.3.1 Counting Size of a System

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

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

3.3.2 Counting Size of a Module

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

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

3.3.3 Counting Complexity of a System

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

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

3.3.4 Counting Complexity of a Module

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

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

3.3.5 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.

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

3.3.6 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.

$$\textit{CountingModuleCoupling}(m_k|\mathbf{S}) = n_{inter\_e\_k} \tag{3.16}$$

3.3.7 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 hyperedges in a complete graph of the system.

$$\textit{CountingSystemCohesion}(\mathbf{S}) = \frac{n_{intr\ \_e}}{n_{ec}} \tag{3.17}$$

3.3.8 Counting Cohesion of a Module

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

$$\textit{CountingModuleCohesion}(m_k|\mathbf{S}) = \frac{n_{intr\ \_e\_k}}{n_{ecm}} \tag{3.18}$$

3.4 Complexity of a Complete Graph

The information theory-based system complexity of a complete graph with ordinary edges has a closed form.

Lemma 1 (Complexity of a complete graph)

$$\mathit{Complexity}(\mathit{MS}^{(n)}) = n\,(n-1)\,\left(-\log\frac{1}{n+1}\right)$$

Proof:

For a complete graph, $\mathit{MS}^{(n)} = \mathit{MS}^{(n)\#}$. Since the $\mathit{Size}(S_i)$ of each node subgraph, S_i, is the same for the entire complete graph, $\mathit{Size}(\mathit{MS}^{(n)\#})$, the summation of Equation (3.5) becomes n times the size of the complete graph. The final term of Equation (3.5) is also the size of the complete graph. Therefore, the complexity of a complete graph is given as

$$\mathit{Complexity}(\mathit{MS}^{(n)}) \quad = \quad n\,\mathit{Size}(\mathit{MS}^{(n)}) - \mathit{Size}(\mathit{MS}^{(n)}) \tag{3.19}$$

By algebra,

$$\mathit{Complexity}(\mathit{MS}^{(n)}) \quad = \quad (n-1)\,\mathit{Size}(\mathit{MS}^{(n)}) \tag{3.20}$$

Since each node has a unique row pattern, the probability mass function p of each node is the same. Therefore, substituting Equation (3.3) into Equation (3.20) gives

$$\mathit{Complexity}(\mathit{MS}^{(n)}) \quad = \quad n\,(n-1)\,(-\log p\,) \tag{3.21}$$

Note that p is one divided by the number of nodes plus one (environment node), because every row pattern is unique. Thus,

$$\textit{Complexity}(\textit{MS}^{(n)}) \quad = \quad n\,(n-1)\left(-\log\frac{1}{n+1}\right) \tag{3.22}$$

■

3.5 Module Complexity of an Intramodule Complete Graph

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

Lemma 2 (Module complexity of an Intramodule Complete Graph)

The information in intramodule relationships of a complete module, $m_k^{(n_k)}$, is given as

$$\textit{Complexity}(m_k^{(n_k)}|\textit{MS}^o) = n_k\,(n-1)\left(-\log\frac{1}{n+1}\right)$$

Proof:

From Equation (3.6),

$$\textit{Complexity}(m_k^{(n_k)}|\textit{MS}^o) \quad = \quad \sum_{i \quad k} \textit{Size}(\textit{MS}_i^{o\#}) - \textit{Size}(m_k^{(n_k)}|\textit{MS}^{o\#}) \tag{3.23}$$

From Equation (3.3),

$$\sum_{i \quad k} \textit{Size}(\textit{MS}_i^{o\#}) \quad = \quad \sum_{i \quad k}\sum_{i=1}^{n}(-\log p_{L(i)}) \tag{3.24}$$

By algebra,

$$\sum_i \sum_k \mathit{Size}(MS_i^{o\#}) \;=\; n\, n_k \left(-\log \frac{1}{n+1}\right) \tag{3.25}$$

From Equation (3.4),

$$\mathit{Size}(m_k^{(n_k)}|MS^{o\#}) \;=\; \sum_i \sum_k (-\log p_{L(i)}) \tag{3.26}$$

Since the pattern of each row is unique for each i, the probability mass function $p_{L(i)}$ is one divided by the number of nodes plus the environment node. Therefore, size of the module is minus the logarithm of the probability mass function times the number of nodes, n_k, in the module m_k.

$$\mathit{Size}(m_k^{(n_k)}|MS^{o\#}) \;=\; n_k \left(-\log \frac{1}{n+1}\right) \tag{3.27}$$

Substituting Equation (3.25) and Equation (3.27) into Equation (3.23)

$$\mathit{Complexity}(m_k^{(n_k)}|MS^{o}) \;=\; (n-1)\, n_k \left(-\log \frac{1}{n+1}\right) \tag{3.28}$$

■

3.6 Metric Calculations

The following two examples illustrate the method of calculating the complexity, coupling and cohesion metrics.

3.6.1 Example: Ordinary Edges

Figure 3.1 and Table 3.11 represent a nodes $\times$ edges graph. In order to find the complexity, the nodes $\times$ edges graph is first translated to an edges-only graph. As from Table 3.11 node 14 is the only node not connected to any other nodes, so it is removed when constructing $S^{\#}$. Once the edges-only graph is obtained, the probability of occurrence of each pattern is found and is tabulated as shown. Notice that the estimated probability of node 0 is one divided by the number of nodes.

Table 3.11 Example Nodes $\times$ Edges Table

Module	Node	Edges	$p_{L(i)}$
M0	0	0000000000000000	$/15$
M1	1	1110000000000000	$1/15$
M2	2	1001100000000000	$1/15$
M2	3	0000110000000000	$1/15$
M2	4	0000011000000000	$1/15$
M3	5	0100001110000000	$1/15$
M3	6	0010000001000000	$1/15$
M3	7	0000000100100000	$1/15$
M3	8	0000000011010000	$1/15$
M3	9	0000000000010000	$1/15$
M4	10	0000000000101100	$1/15$
M4	11	0001000000001010	$1/15$
M4	12	0000000000000101	$1/15$
M4	13	0000000000000011	$1/15$
M4	14	0000000000000000	$/15$

Figure 3.2 and Table 3.12 represent an edges-only graph. From Table 3.12 for each node except for node 0 identify which other nodes the corresponding node is linked to. For example for node 1 identify the columns in the edges pattern that consists of 1 and list those columns as the new pattern. Once the pattern is identified, find the probability of each pattern and tabulate the probability columns to construct S_i.

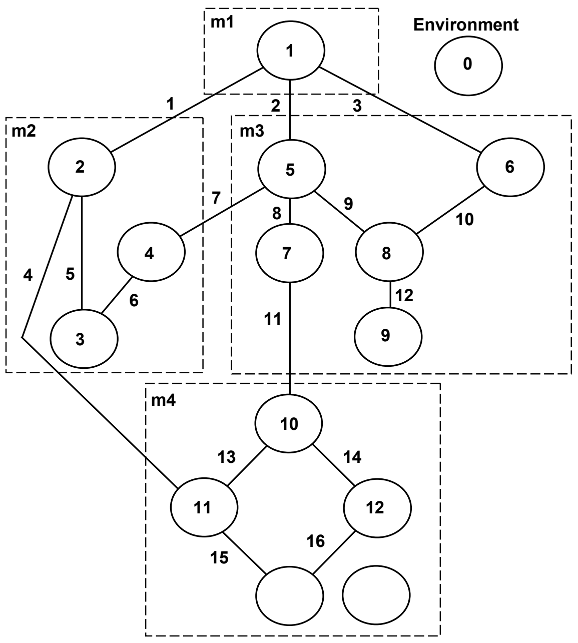


Figure 3.1 Example Nodes × Edges

Table 3.12 Example Edges-Only Table

Module	Node	Edges	$p_{L(i)}$
M0	0	0000000000000000	$1/14$
M1	1	1110000000000000	$1/14$
M2	2	1001000000000000	$1/14$
M2	3	0000100000000000	$1/14$
M2	4	0000011000000000	$1/14$
M3	5	0100001110000000	$1/14$
M3	6	0010000001000000	$1/14$
M3	7	0000000100100000	$1/14$
M3	8	0000000011010000	$1/14$
M3	9	0000000000010000	$1/14$
M4	10	0000000000101100	$1/14$
M4	11	0001000000001010	$1/14$
M4	12	0000000000000101	$1/14$
M4	13	0000000000000011	$1/14$

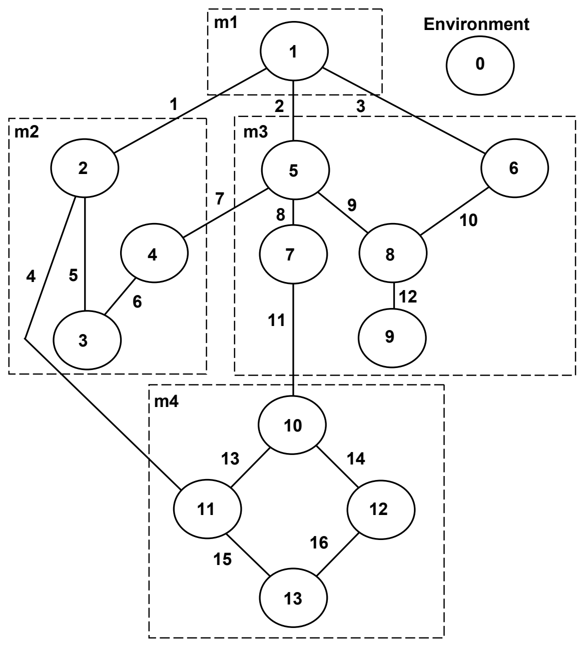


Figure 3.2 Example Edges-Only Graph

Table 3.13 represents the graph for node 1, and Table 3.14 represents the graph of node 2. Likewise, tables are generated for every other node. Calculating the size for each graph obtained and summing the results yields the sum in Equation (3.5).

In order to find coupling and cohesion we generate an intermodule edges-only graph and an intramodule edges-only graph from the graph represented by Table 3.11. Both the graphs can be obtained simultaneously as follows. Starting from the first column of the edges pattern, identify the first "1" along the column and note its module. Then traverse down the column to find if there exists another "1" in a different module. If a "1" exits then list the column in the intermodule-edges table. Otherwise list it in the intramodule-edges table. Likewise, traverse each and every column of the pattern and concatenate the result

Table 3.13 Node 1 Subgraph

Module	Node	Edges	$p_{L_1(j)}$
M0	0	000	$10/14$
M1	1	111	$1/14$
M2	2	100	$1/14$
M2	3	000	$10/14$
M2	4	000	$10/14$
M3	5	010	$1/14$
M3	6	001	$1/14$
M3	7	000	$10/14$
M3	8	000	$10/14$
M3	9	000	$10/14$
M4	10	000	$10/14$
M4	11	000	$10/14$
M4	12	000	$10/14$
M4	13	000	$10/14$

Table 3.14 Node 2 Subgraph

Module	Node	Edges	$p_{L_2(j)}$
M0	0	00	$11/14$
M1	1	10	$1/14$
M2	2	11	$1/14$
M2	3	00	$11/14$
M2	4	00	$11/14$
M3	5	00	$11/14$
M3	6	00	$11/14$
M3	7	00	$11/14$
M3	8	00	$11/14$
M3	9	00	$11/14$
M4	10	00	$11/14$
M4	11	01	$1/14$
M4	12	00	$11/14$
M4	13	00	$11/14$

with the previous list to identify the final table. Figure 3.3 and Table 3.15 represent an intermodule edges-only graph, and Figure 3.4 and Table 3.16 represent an intramodule-edges graph.

Table 3.15 Example Intermodule-Edges Graph

Module	Node	Edges	$p_{L(i)}$
M0	0	000000	$1/9$
M1	1	111000	$1/9$
M2	2	100100	$1/9$
M2	4	000010	$1/9$
M3	5	010010	$1/9$
M3	6	001000	$1/9$
M3	7	000001	$/9$
M4	10	000001	$/9$
M4	11	000100	$1/9$

Calculating the complexity of an intermodule-edges graph yields coupling, and the complexity of an intramodule-edges graph divided by the complexity of a complete graph (every node is connected to every other node) yields cohesion.

Table 3.17 represents the information theory-based system-level metrics and the counting-based system-level metrics, and Table 3.18 represents the information theory-based module-level metrics and counting-based module-level metrics. From Table 3.18, for ordinary edges it can be said that the information of size, complexity, coupling and cohesion of each module have the same variation as that of the counting size, complexity, coupling and cohesion, respectively. Therefore, in this example, using either information measure-

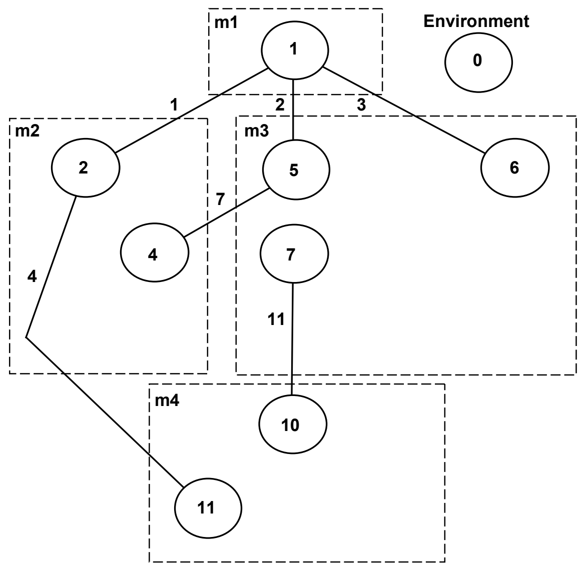


Figure 3.3 Example Intermodule-Edges Graph

Table 3.16 Example Intramodule-Edges Graph

Module	Node	Edges	$p_{L(i)}$
M0	0	0000000000	$1/12$
M2	3	1000000000	$1/12$
M2	4	0100000000	$1/12$
M3	5	0011000000	$1/12$
M3	6	0000100000	$1/12$
M3	7	0010000000	$1/12$
M3	8	0001110000	$1/12$
M3	9	0000010000	$1/12$
M4	10	0000001100	$1/12$
M4	11	0000001010	$1/12$
M4	12	0000000101	$1/12$
M4	13	0000000011	$1/12$

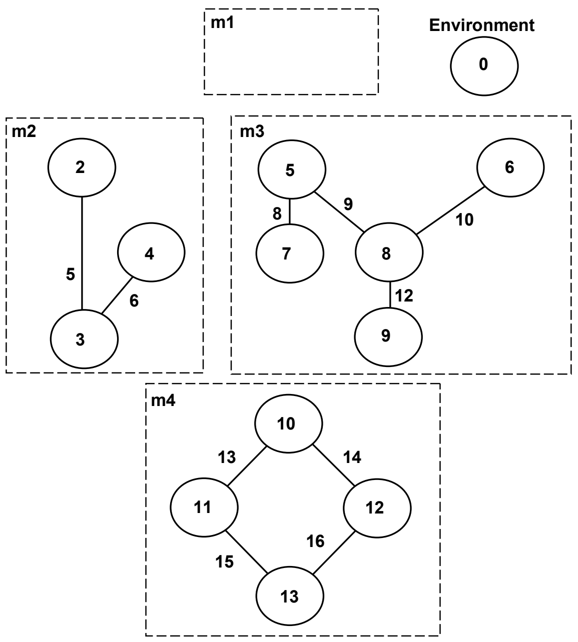


Figure 3.4 Example Intramodule-Edges Graph

ment or counting measurement for calculation of size, complexity, coupling and cohesion does not make a difference.

3.6.2 Example: Hyperedges

Figure 3.5 and Table 3.19 represent a nodes $\times$ hyperedges graph. In order to find the complexity, the nodes $\times$ hyperedges graph is first translated to a hyperedges-only graph. As from Table 3.19, node 14 is the only node not connected to any other nodes, so it is removed. Once the hyperedges-only graph is obtained the probability of occurance of each pattern is found and is tabulated as shown. It can be noticed that the probability of node 0 is one divided by the number of nodes.

Table 3.17 Example Ordinary Edges System-level Metric Values

	Information	Counting
Size	53.7 bits	14 nodes
Complexity	170.1 bits	16 edges
Coupling	50.2 bits	6 edges
Cohesion	0.14	0.11

Table 3.18 Example Ordinary Edges Module-level Metric Values

	Information Theory-based Metrics			
Module	size (bits)	complexity (bits)	coupling (bits)	cohesion
M1	3.9	15.8	12.9	0.00
M2	11.7	38.0	12.6	0.16
M3	19.5	62.6	17.0	0.19
M4	18.5	53.8	7.7	0.21
	Counting-based Metrics			
Module	size (nodes)	complexity (edges)	coupling (edges)	cohesion
M1	1	3	3	0.00
M2	3	5	3	0.67
M3	5	8	4	0.40
M4	5	6	2	0.67

Table 3.19 Example Nodes × Hyperedges Table

Module	Node	Edges	$p_{L(i)}$
M0	0	0000000000	$/15$
M1	1	1000000000	$1/15$
M2	2	1100000000	$1/15$
M2	3	0100100000	$1/15$
M2	4	0010100000	$1/15$
M3	5	1010000000	$1/15$
M3	6	1001000000	$1/15$
M3	7	0010010000	$1/15$
M3	8	0011001000	$1/15$
M3	9	0000001000	$1/15$
M4	10	0000010100	$1/15$
M4	11	0100000110	$1/15$
M4	12	0000000101	$1/15$
M4	13	0000000011	$1/15$
M4	14	0000000000	$/15$

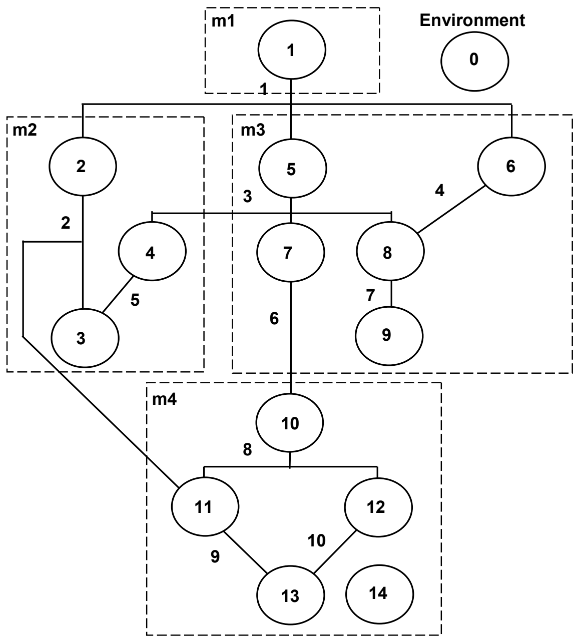


Figure 3.5 Example Node× Hyperedges Graph

Figure 3.6 and Table 3.20 represent a hyperedges-only graph. From Table 3.20 for each node except for node 0, identify which other nodes are linked to it. For example, for node 1, identify the columns in the hyperedges pattern that consists of “1” and list those columns as the new pattern. Once the pattern is identified find the probability of each pattern and tabulate the probability column.

Table 3.20 Example Hyperedges-Only Table

Module	Node	Edges	$p_{L(i)}$
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	$1/14$

Table 3.21 represents the graph for node 1, and Table 3.22 represents the graph of node 2. Likewise, the tables are generated for every other node. Calculating the size for each graph obtained and summing the results yields the complexity of the graph.

In order to find coupling and cohesion, we generate an intermodule hyperedges-only graph and an intramodule hyperedges-only graph from the graph represented by Table 3.19.

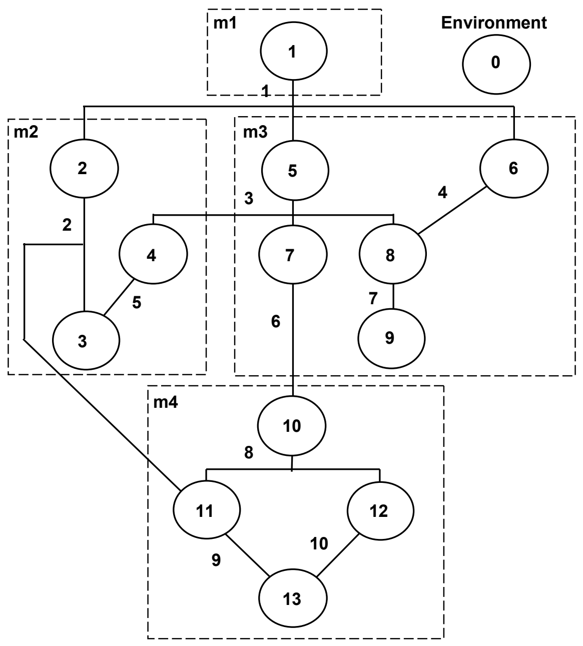


Figure 3.6 Example Hyperedges-Only Graph

Table 3.21 Node 1 Subgraph

Module	Node	Edges	$p_{L_i(j)}$
M0	0	0	$10/14$
M1	1	1	$4/14$
M2	2	1	$4/14$
M2	3	0	$10/14$
M2	4	0	$10/14$
M3	5	1	$4/14$
M3	6	1	$4/14$
M3	7	0	$10/14$
M3	8	0	$10/14$
M3	9	0	$10/14$
M4	10	0	$10/14$
M4	11	0	$10/14$
M4	12	0	$10/14$
M4	13	0	$10/14$

Table 3.22 Node 2 Subgraph

Module	Node	Edges	$p_{L_2(j)}$
M0	0	00	$10/14$
M1	1	10	$4/14$
M2	2	11	$4/14$
M2	3	01	$10/14$
M2	4	00	$10/14$
M3	5	10	$4/14$
M3	6	10	$4/14$
M3	7	00	$10/14$
M3	8	00	$10/14$
M3	9	00	$10/14$
M4	10	00	$10/14$
M4	11	01	$10/14$
M4	12	00	$10/14$
M4	13	00	$10/14$

Both graphs can be obtained simultaneously as follows. Starting from the first column of the hyperedges pattern, identify the first "1" along the column and note its module. Then, traverse down the column to find if there exists another "1" in a different module. If a "1" exits then list the column in the intermodule-edges table. Otherwise, list it in the intramodule-edges table. Likewise, traverse each and every column of the pattern and concatenate the result with the previous list to identify the final table. Figure 3.7 and Table 3.23 represent an intermodule hyperedges-only graph, and Figure 3.8 and Table 3.24 represent an intramodule-hyperedges graph.

Calculating the complexity of an intermodule-edges graph yields coupling, and complexity of an intramodule-edges graph divided by the complexity of a complete graph (every node is connected to every other node) yields cohesion.

Table 3.23 Example Intermodule-Hyperedges Graph

Module	Node	Edges	$p_{L(i)}$
M0	0	0000	$1/11$
M1	1	1000	$/11$
M2	2	1100	$1/11$
M2	3	0100	$/11$
M2	4	0010	$/11$
M3	5	1010	$1/11$
M3	6	1000	$/11$
M3	7	0011	$1/11$
M3	8	0010	$/11$
M4	10	0001	$1/11$
M4	11	0100	$/11$

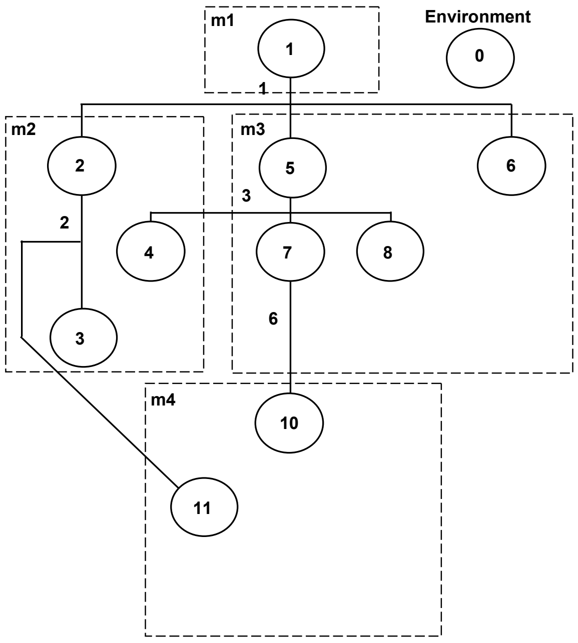


Figure 3.7 Example Intermodule-Hyperedges Graph

Table 3.24 Example Intramodule-Hyperedges Graph

Module	Node	Edges	$p_{L(i)}$
M0	0	0000000	$1/12$
M2	3	1000000	$/12$
M2	4	1000000	$/12$
M3	5	0100000	$/12$
M3	6	0010000	$1/12$
M3	7	0100000	$/12$
M3	8	0111000	$1/12$
M3	9	0001000	$1/12$
M4	10	0000100	$1/12$
M4	11	0000110	$1/12$
M4	12	0000101	$1/12$
M4	13	0000011	$1/12$

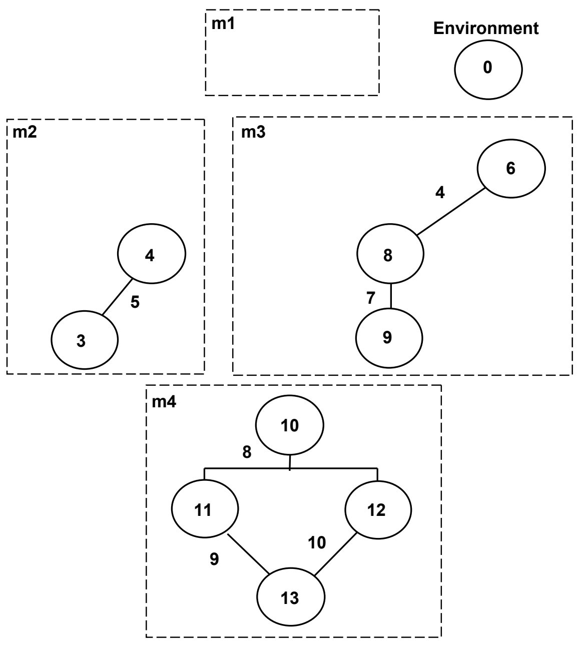


Figure 3.8 Example Intramodule-Hyperedges Graph

Table 3.25 represents the information theory-based system-level metrics and the counting-based system-level metrics, and Table 3.26 represents the information theory-based module-level metrics and counting-based module-level metrics. From Table 3.26, for hyperedges it can be said that the information of size, complexity, coupling and cohesion of each module has the same variation as the counting size, complexity, coupling and cohesion respectively. Therefore, in this example using either information measurement or counting measurement for calculation of size, complexity, coupling, and cohesion does not make a difference.

Table 3.25 Example Hyperedges System-level Metric Values

	Information	Counting
Size	53.7 bits	14 nodes
Complexity	189.1 bits	10 edges
Coupling	89.9 bits	4 edges
Cohesion	0.10	0.07

Table 3.26 Example Hyperdges Module-level Metric Values

	Information Theory-based Metrics			
Module	size (bits)	complexity (bits)	coupling (bits)	cohesion
M1	3.9	7.8	7.3	0.00
M2	11.7	47.3	28.8	0.11
M3	19.5	74.7	43.7	0.13
M4	18.5	59.4	10.2	0.29
	Counting-based Metrics			
Module	size (nodes)	complexity (hyperedges)	coupling (hyperedges)	cohesion
M1	1	1	1	0.00
M2	3	4	4	1.00
M3	5	5	6	0.67
M4	5	5	2	0.5

CHAPTER IV

METHODOLOGY

A case study approach is taken to illustrate the usefulness of the metrics in a real-world setting. Case studies provide weight of evidence, rather than scientific proof of propositions. A graph can represent an abstraction of a software system [11]. The objective is to measure graphs directly used by designers that are likely to be related to software quality, such as artifacts produced by design tools, or graphs derived from relationships in code. The research project is a case study consisting of three tasks: (1) developing research tools, (2) collecting data from various sources, and (3) analyzing data for useful relationships. The case study examined software systems of limited size to give an indication of metric usefulness and to resolve practical issues. The objective of the case study was to evaluate module-level metrics and system-level metrics.

The analysis included the following steps:

1. Obtain sets of source code files to be analyzed.
2. Generate an abstract semantic graph for each file using the Datrix tool.
3. For each abstraction, generate a nodes $\times$ hyperedges table using the Abstractor tool. This tool currently analyzes use of public variables by methods.
4. For each node $\times$ hyperedges table, calculate the information theory-based attributes and corresponding counting-based attributes of the system and of each module using the Measurement tool.
5. Analyze the distributions and correlations among measured attributes using SAS and Excel by generating graphs.

CHAPTER V

TOOLS

5.1 Architecture

Figure 5.1 represents a pipes-and-filters architecture. A raw source code file is given to a compiler preprocessor to obtain preprocessed source code (*.ii). The preprocessed source code is parsed with the Datrix parser to generate an abstract semantic graph (ASG), which is an output to a file (*.asg). Using the ASG, an abstraction extraction is performed, such as use of global variables, call graph, or control flow graph. At this time we focus on the use of global variables as the abstraction extracted. In this approach, we identify the classes or methods and their associated global variables. With the matchings a node $\times$ hyperedge table is generated. The measurement is then applied to the node $\times$ hyperedge table to obtain various software measurements for both system level and module level. The measurements are stored in a tabular file and are analyzed statistically using SAS and Excel.

Table 5.1 represents a node $\times$ hyperedge table. The node $\times$ hyperedge table file consists of four fields. The first field specifies the software-identifier, the second field specifies the module-identifier, the third field specifies the node-identifier, and the fourth field specifies the hyperedges (row pattern), which are represented by a binary pattern. The binary pattern for each node shows to which other nodes the current node is connected. The number of hyperedges in each row are equal, and only those hyperedges that are connected to a particular node are represented by “1”, the rest are represented by “0”.

Table 5.1 Nodes $\times$ Hyperedges

Software-Id	Module-Id	Node-Id	RowPattern
sw1	m1	n1	1001
sw1	m1	n2	1000
sw1	m2	n3	1011
sw1	m3	n4	0110
sw1	m3	n5	0101

5.2 Design of Measurement Package

The box shown as “Measurement” in Figure 5.1 represents the measurement package. The measurement package is a tool that calculates the information metrics and counting metrics defined in Chapter III.

5.2.1 Class Diagram of Measurement Program

Figure 5.2 represents a class diagram for the implementation of the calculation of the defined information theory-based metrics and the counting-based metrics. The interfaces *Collection* and *Map* and classes *Abstract Set*, *Hash Set*, *Array List*, *Abstract Map*, and *Hash Map* are off-the-shelf components. Class *ModularSystem* implements the calcula-

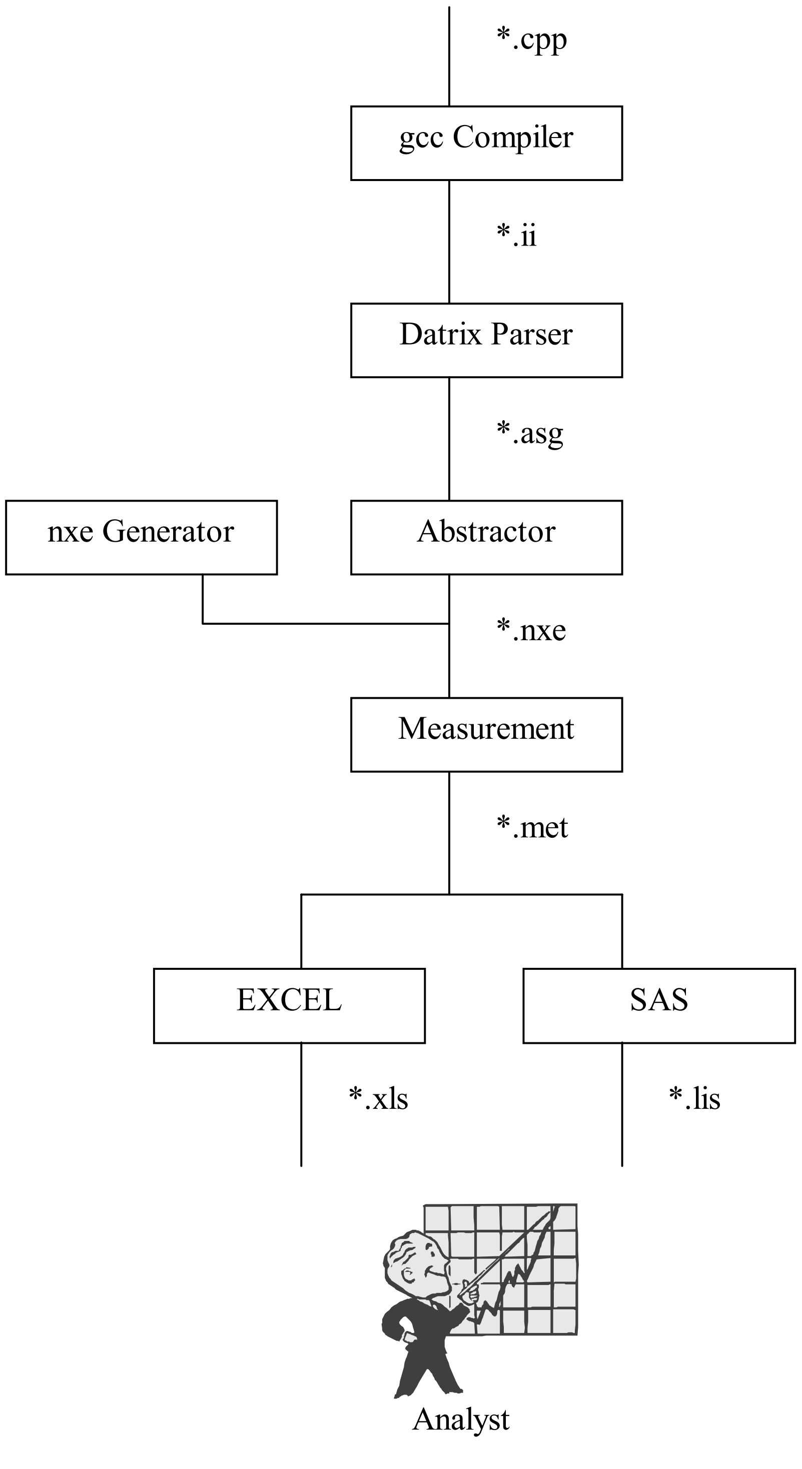


Figure 5.1 Tool Kit Architecture

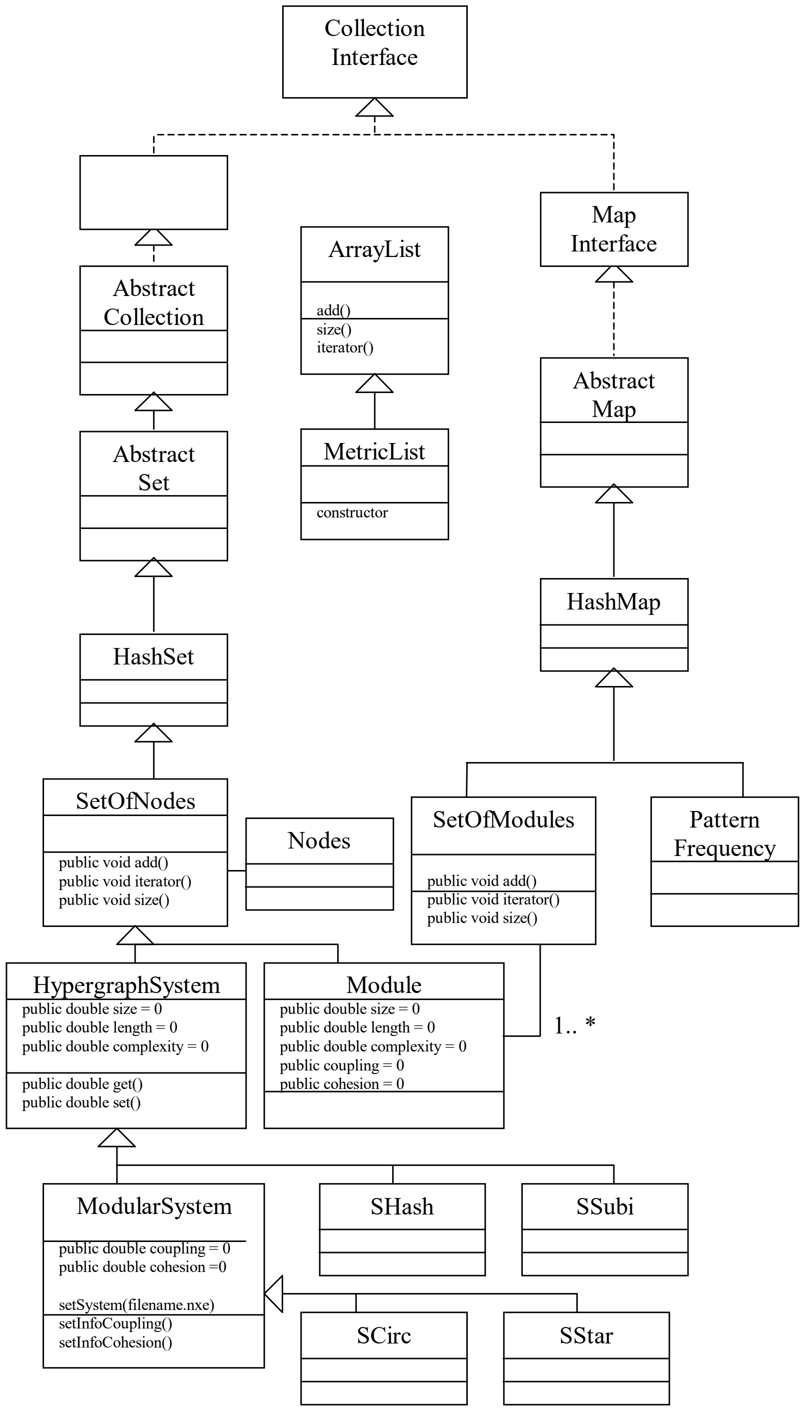


Figure 5.2 Class Diagram

tion of *Coupling* and *Cohesion* of information theory-based metrics and counting-based metrics. It also sets the *SetOfNodes* and *SetOfModules* for the given node $\times$ hyperedges file. Class *HypergraphSystem* implements the calculation of size and complexity of information theory-based metrics and the counting-based metrics. The class *SetOfNodes* creates the node objects, and the class *SetOfModules* creates the module objects. Class *PatternFrequency* finds the number of times each pattern occurs in a given node $\times$ hyperedges graph. The class *SHash* implements the hyperedges-only graph, *SSubi* implements the node subgraph, *SCirc* implements the intramodule hyperedges graph, and the class *SStar* implements the intermodule hyperedges graph.

5.2.2 Call Graph of Measurement Program

Figures 5.3 through 5.10 show the call graphs of the methods. The *Main* call graph creates an instance of the *SetOfModules* class, identifies the list of metrics to be calculated, calculates all the infomation theory-based metrics and counting-based metrics, and finally generates the output metric file. The *setInfoMetrics* call graph and *setCountMetrics* call graph call the methods for calculating the size, complexity, coupling, and cohesion of information theory-based metrics and counting-based metrics, respectively. The *getMetrics* call graph gets all the calculated metrics required by the user, provided in the list of metrics input file. The *setInfoSize*, *setInfoComplexity*, *setInfoCoupling*, and *setInfoCohesion* call graphs show the implementation for calculation of size, complexity, coupling, and cohesion of information theory-based metrics, respectively.

```
Main
   ├── HypergraphSystem
   │          ├── SetOfModules()
   │          └── setSetOfModules()
   ├── MetricsList(fileName)
   │          ├── toUppercase()
   │          ├── add(String)
   │          └── ArrayList()
   ├── S.setSystem(.nxe filename)
   │          ├── Node
   │          ├── getSetOfModules()
   │          ├── findModule(String, Module)
   │          ├── Module
   │          ├── SOM.add(Module)
   │          ├── M.add(Node)
   │          └── S.add(Node)
   ├── S.setInfoMetrics(HypergraphSystem)
   ├── S.setCountMetrics(HypergraphSystem)
   └── S.getMetrics(HypergraphSystem)
```

Figure 5.3 Main Call Graph

```
setInfoMetrics(HypergraphSystem)
   ├── S.setInfoSize()
   ├── S.setInfoComplexity()
   ├── S.setInfoCoupling()
   └── S.setInfoCohesion()
```

Figure 5.4 setInfoMetrics Call Graph

```
setCountMetrics(HypergraphSystem)
   ├── S.setCountSize()
   ├── S.setCountComplexity()
   ├── S.setCountCoupling()
   └── S.setCountCohesion()
```

Figure 5.5 setCountMetrics Call Graph

```
getMetrics(HypergraphSystem)
    ├── ML.iterator()
    ├── S.getSetOfModules()
    ├── SOM.iterator()
    ├── S.getSystemInfoSize()
    ├── S.getSystemInfoComplexity()
    ├── S.getSystemInfoCoupling()
    ├── S.getSystemInfoCohesion()
    ├── M.getModuleInfoSize()
    ├── M.getModuleInfoComplexity()
    ├── M.getModuleInfoCoupling()
    ├── M.getModuleInfoCohesion()
    ├── S.getSystemCountSize()
    ├── S.getSystemCountComplexity()
    ├── S.getSystemCountCoupling()
    ├── S.getSystemCountCohesion()
    ├── M.getModuleCountSize()
    ├── M.getModuleCountComplexity()
    ├── M.getModuleCountCoupling()
    └── M.getModuleCountCohesion()
```

Figure 5.6 getMetrics Call Graph

```
SetInfoSize
    ├─ S.PatternFrequency()
    │          ├─ S.iterator()
    │          ├─ N.getRowPattern()
    ├─ S.getSetOfModules()
    ├─ SOM.iterator()
    ├─ M.setModuleInfoSize(PatternFrequency)
    │          ├─ S.size()
    │          ├─ M.iterator()
    │          ├─ N.getRowPattern()
    │          ├─ S.findPatternFrequency(String)
    ├─ M.getModuleInfoSize()
    ├─ S.setSystemInfoSize()
```

Figure 5.7 setInfoSize Call Graph

```
setInfoComplexity
    ├─ S.getSetOfModules()
    ├─ SOM.iterator()
    ├─ S.SHash()
    │          ├─ S.setInfoSize()
    │          ├─ S.getInfoSize()
    ├─ S.SSubi()
    │          ├─ S.setInfoSize()
    │          ├─ S.getInfoSize()
    ├─ S.setSystemInfoComplexity()
```

Figure 5.8 setInfoComplexity Call Graph

```
setInfoCoupling
    ├─ S.getSetOfModules()
    ├─ SOM.iterator()
    ├─ S.SStar()
    │          ├─ S.setInfoComplexity()
    │          ├─ S.getInfoComplexity()
    ├─ S.setSystemInfoCoupling()
```

Figure 5.9 setInfoCoupling Call Graph

```
setInfoCohesion
    ├── S.getSetOfModules()
    ├── SOM.iterator()
    ├── S.SCirc()
    │           ├── S.setInfoComplexity()
    │           ├── S.getInfoComplexity()
    ├── S.Sn()
    └── S.setSystemInfoCoupling()
```

Figure 5.10 setInfoCohesion Call Graph

5.3 Off-the-Shelf Components

Table 5.2 gives an overview of the off-the-shelf components used in this research.

Table 5.2 Off-the-Shelf Components

Off-the-shelf component	Description
gcc	The GNU Compiler Collection is used to compile a given C++ program. The output is the preprocessed file in the format of *.ii Command: gcc -E <filename.cpp> <outputfile>(*.ii) The command -E is used to eliminate syntax errors and general warnings.
Datrix	A software code assessment tool provided by Bell Canada Command: dxparscpp -asg <output.asg> *.ii dxparscpp is a datrix parser for C++ files that builds an Abstract Symantic Graph (asg) and outputs it in a *TA-like* format.
Abstractor	This is a tool that reads the asg file and generates a node $\times$ edges table (*.nxe).
SAS	A statistics package for data manipulation, statistical analysis, report writing, and generating plots.
nxeGenerator	The nxeGenerator generates a set of nodes $\times$ hyperedges tables depending on the user inputs.

CHAPTER VI

CASE STUDIES

This chapter provides exploratory case studies of (1) a set of artificially generated graphs, (2) a data manipulation program for a physics research project, and (3) selected source files from a mathematical library.

6.1 Nodes$\times$Hyperedges Generator Examples

6.1.1 Data Collection

The tool is a nodes $\times$ hyperedges generator (NxeGenerator). Based on the user input, various nodes $\times$ hyperedges (*.nxe) files are generated. These files are then measured using the measurement tool and analyzed. Figure 6.1 depicts a series of small graphs where a node is added and then a hyperedge is added. Figure 6.2 depicts three series of graphs where hyperedges are added that have the same connections as existing hyperedges. In these small graphs, every node is considered a module.

6.1.2 Measurement

Table 6.1 presents the system-level metrics for Figure 6.1 and Figure 6.2. Coupling measurements are the same as complexity because every hyperedge is an intermodule

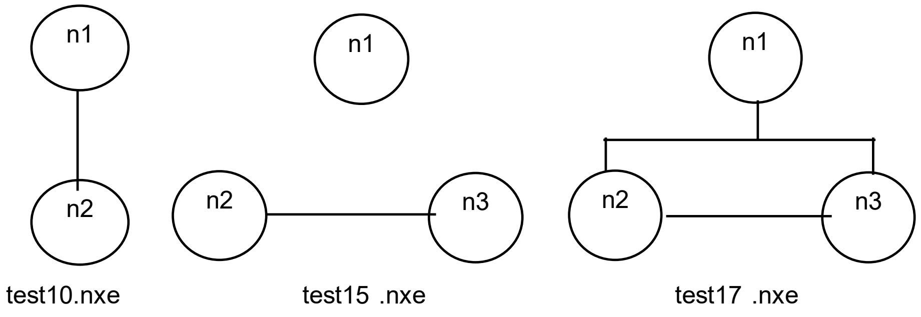


Figure 6.1 Adding a Node and a Hyperedge to a Small Graph

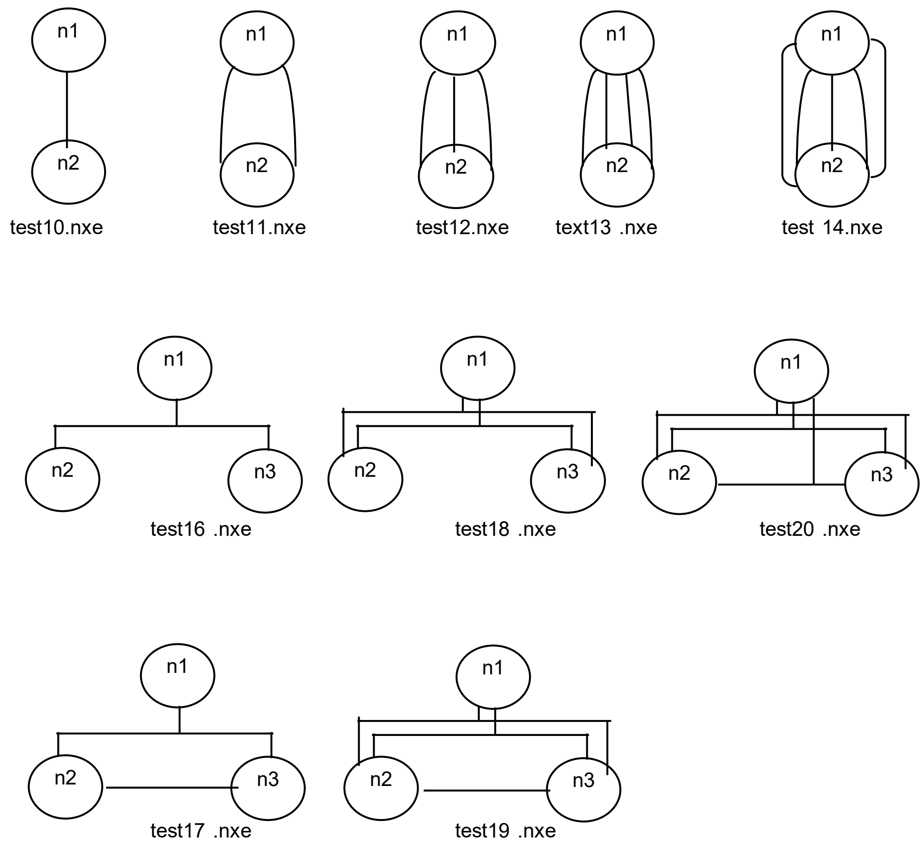


Figure 6.2 Identical Hyperedges Do Not Add Information

hyperedge. Cohesion measurement is zero because there is no intramodule hyperedge, as every node is a module.

Table 6.1 System-level Measurements of NxeGenerator Examples

	Size		Complexity	
	Information	Count	Information	Count
System	(bits)	(nodes)	(bits)	(hyperedges)
test10	1.2	2	1.2	1
test15	3.0	3	1.2	1
test17	4.0	3	5.3	2
test10	1.2	2	1.2	1
test11	1.2	2	1.2	2
test12	1.2	2	1.2	3
test13	1.2	2	1.2	4
test14	1.2	2	1.2	5
test16	1.3	3	2.5	1
test18	1.3	3	2.5	2
test20	1.3	3	2.5	3
test17	4.0	3	5.3	2
test19	4.0	3	5.3	3

6.1.3 Analysis

The graphs in Figure 6.1 illustrate how adding a node increases information size but not information complexity, and how adding a hyperedge increases both information size and information complexity. The graphs in Figure 6.2 illustrate that the information theory-based measurements are not sensitive to multiple hyperedges connected to exactly the same nodes because redundant hyperedges do not affect the estimated probabilities of row pattern.

Figure 6.3 depicts two pairs of binary trees. Trees 1a and 2a have ordinary edges (two connections per edge). Trees 1b and 2b have hyperedges with three connections per edge. Figure 6.4 depicts two pairs of (nonbinary) trees. Trees 3a and 4a have ordinary edges (two connections per edge). Trees 3b and 4b have more than two connections per hyperedge. Table 6.2 presents the system-level metrics for these two figures. Abstractions of software using ordinary edges make a distinction for each edge relationship. Abstractions using hyperedges are appropriate when such distinctions are not relevant and thus information size is smaller. However, we see that information complexities are about the same.

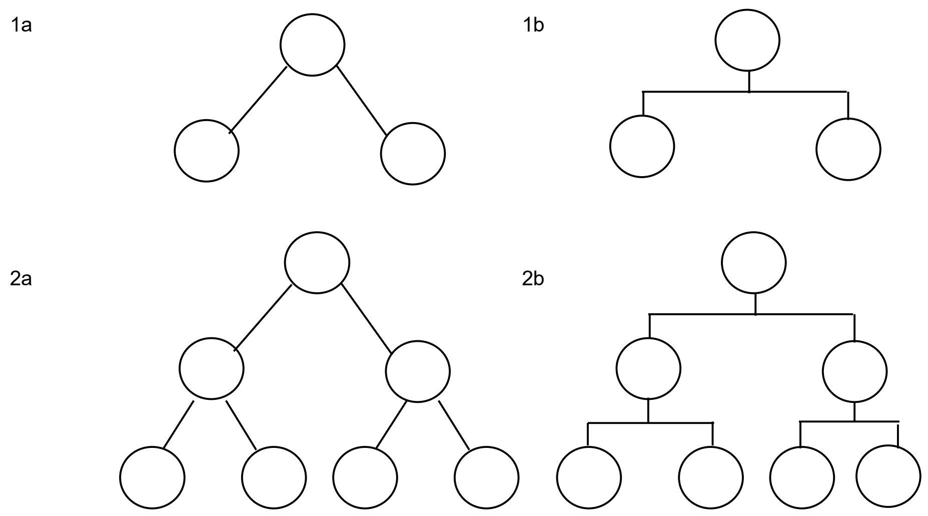


Figure 6.3 Binary Trees with Ordinary Edges vs. Hyperedges

From Table 6.3 through Table 6.6 it is observed empirically that the information complexity of module M1 is negative. For a graph to result in a negative module complexity, we conjecture that the following conditions must be satisfied.

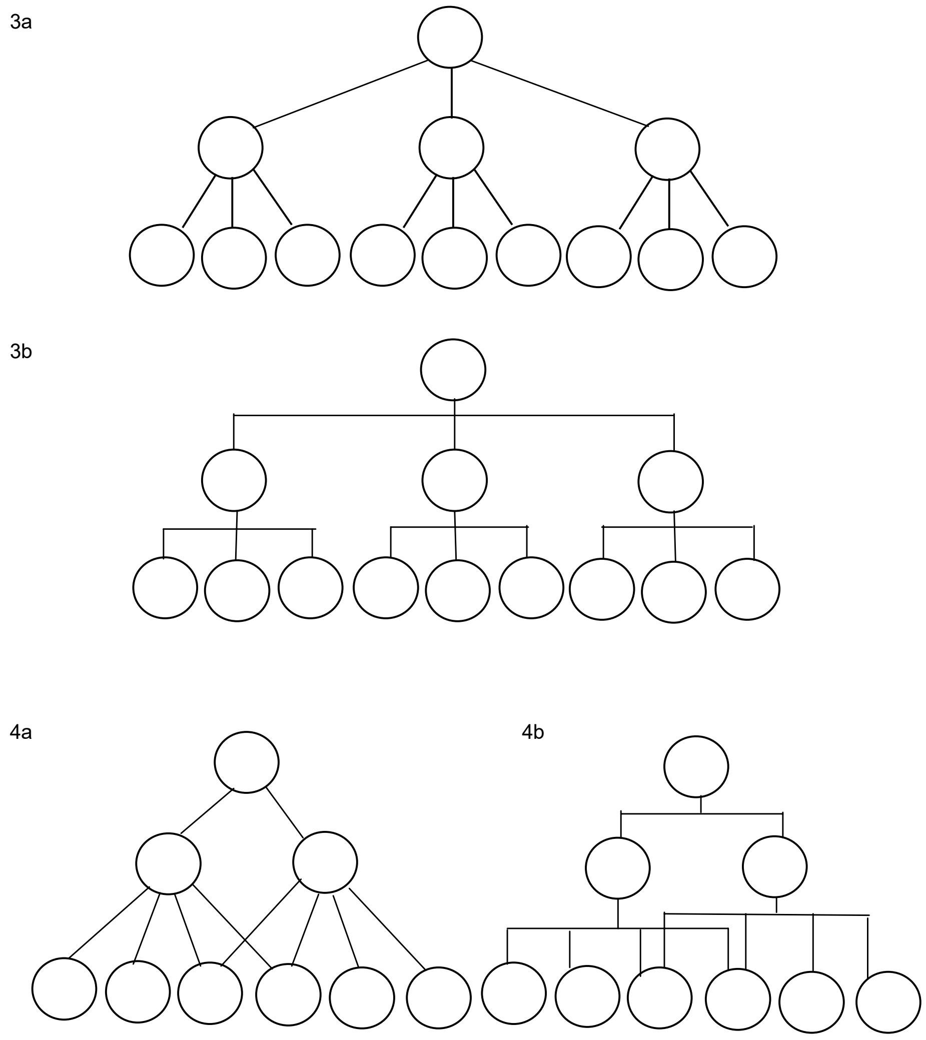


Figure 6.4 Trees with Ordinary Edges vs. Hyperedges

Table 6.2 System-level Measurement of Trees with Ordinary Edges vs. Hyperedges

	Ordinary Edges				Hyperedges			
	Information		Count		Information		Count	
System	1a				1b			
Size	6.0	bits	3	nodes	1.2	bits	3	nodes
Complexity	6.0	bits	2	edges	2.5	bits	1	hyperedge
System	2a				2b			
Size	21.0	bits	7	nodes	17.0	bits	7	nodes
Complexity	45.0	bits	6	edges	45.5	bits	3	hyperedges
System	3a				3b			
Size	49.5	bits	13	nodes	35.2	bits	13	nodes
Complexity	115.1	bits	12	edges	150.2	bits	4	hyperedges
System	4a				4b			
Size	30.0	bits	9	nodes	23.9	bits	9	nodes
Complexity	84.6	bits	10	edges	88.6	bits	3	hyperedges

- The row pattern of all the nodes in a module must be identical.
- A hyperedge associated to a node in that module must also be associated to all other nodes in the system.
- There should be at least one hyperedge associated to that module.
- There should be at least one hyperedge not associated to that module.

Future work will mathematically prove the above conjecture.

For a complete graph it is empirically observed that:

- Information complexity is equal to information coupling.
- Counting system complexity is equal to counting module complexity.
- Counting module coupling is equal to the product of counting module size and counting system complexity or the product of counting module size and counting system coupling.

Table 6.3 Nodes × Hyperedges Table Example 1

Module	Node	Hyperedges
env.	0	00000
M1	**N1**	**10000**
M2	N2	11000
M2	N3	10110
M3	N4	10001

Table 6.4 Nodes × Hyperedges Table Example 2

Module	Node	Hyperedges
env.	0	00000
M1	**N1**	**11110**
M1	**N2**	**11110**
M2	N3	11111
M3	N4	11111
M4	N5	11111

Table 6.5 Nodes × Hyperedges Table Example 3

Module	Node	Hyperedges
env.	0	00000
M1	**N1**	**11100**
M1	**N2**	**11100**
M2	N3	11111
M3	N4	11111
M4	N5	11111

Table 6.6 Nodes $\times$ Hyperedges Table Example 4

Module	Node	Hyperedges
env.	0	00000
M1	**N1**	**10000**
M1	**N2**	**10000**
M2	N3	11111
M3	N4	11111
M4	N5	11111

For a given system with no hyperedges it is empirically observed that the information measurement of all the attributes is zero, whereas the counting size is the number of nodes in the system given by Equation (3.11).

6.2 Physics Programs

The program under study is part of a physics experiment. The number of files included in the project were two C++ (*.cpp) files and three header (*.hpp) files. The program reads different data sets, manipulates the data, and writes the results to an output file.

6.2.1 Data Collection

A hypergraph is derived from the relationships between public variables and the methods that use them. The methods were represented by nodes, and each public variable is represented by a hyperedge. Each class was defined as a module. C++ system classes, methods and public variables are excluded from the analysis. The primary C++ file is preprocessed using the gcc complier to generate a preprocessed (*.ii) file. This step includes all the header files (*.hpp) and subsidiary C++ files (*.cpp) into the resulting file (*.ii). The preprocessed file (*.ii) is parsed using the Datrix metric analyzer, generating an abstract semantic graph(ASG, *.asg). The ASG file (*.asg) is then an input to the Abstractor that generates a nodes $\times$ hyperedges table (*.nxe). The *.nxe file is then an input to the Measurement tool for the metric calculations.

6.2.2 Measurement

Table 6.7 Nodes $\times$ Hyperedges for the Physics Program

Module	Node	Hyperedges
env.	0	00000000000000000000000000000000
1.Atom	getpos	01100100000000010100000101010000
	setid	00000100001010000000000000010000
	setname	00000000000000000000000000000001
2.Element	get_mass	01100000000000000000000000000000
	get_name	01000000000000000000000000010000
	get_weight	00100000000000000100000000010000
	report	00100000000000000000010000000000
	setname	00000000000000001000000100101001
3.Lattice	get_periodic	10010011000011110110010000000111
	get_scale	01001100000000011100000111111001
	get_refvector	00110111110100010100111100010011
	get_parameter	00000100001010100000000000010000
	set_latticetype	00000000000000000100000000000000
	set_parameter	00000000000000000001000000000000

Table 6.7 represents a nodes $\times$ hyperedges graph for the given program. Table 6.8 represents the information theory-based system-level metrics and the counting-based system-

level metrics, and Table 6.9 represents the information theory-based module-level metrics and counting-based module-level metrics.

Table 6.8 System-level Measurements of the Physics Program

	Information	Counting
Size	54.7 (bits)	14 (nodes)
Complexity	366.3 (bits)	32 (edges)
Coupling	341.2 (bits)	15 (edges)
Cohesion	0.02	0.18

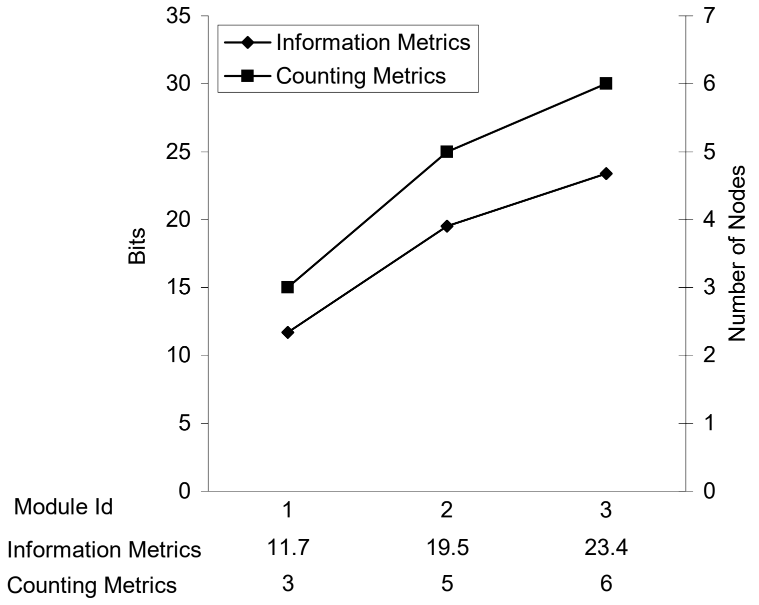
6.2.3 Analysis

Figure 6.5 through Figure 6.8 present a comparision of size, complexity, coupling, and cohesion attributes of information theory-based metrics and counting-based metrics at the module-level, respectively.

Table 6.9 shows that the module “Element” has medium information complexity and module “Atom” has low information complexity. However, the counting complexity is about the same. The module “Lattice” contributes more than the other modules to all the system-level metrics. In this case study, if one uses a metric to order modules, the corresponding information metrics and the counting metrics generally result in the same order.

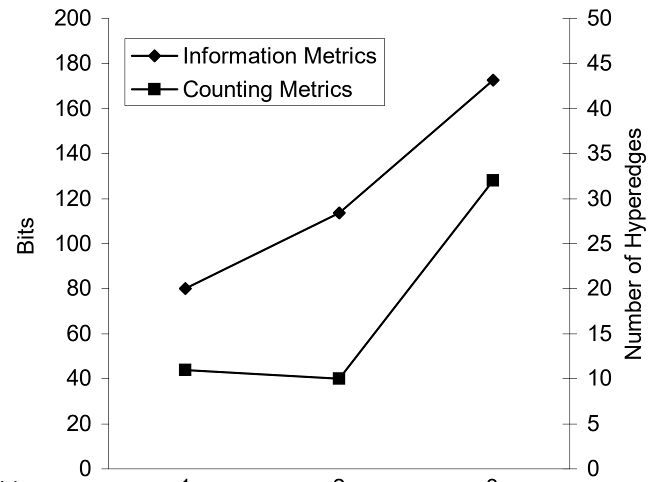
Table 6.9 Module-level Measurements of the Physics Program

	Information Theory-based Metrics			
Module	size (bits)	complexity (bits)	coupling (bits)	cohesion
1.Atom	11.7	**80.1**	76.4	0.00
2.Element	19.5	**113.6**	107.7	0.00
3.Lattice	23.4	172.6	157.1	0.26
	Counting-based Metrics			
Module	size (nodes)	complexity (hyperedges)	coupling (hyperedges)	cohesion
1.Atom	3	**11**	13	0.0
2.Element	5	**10**	14	0.0
3.Lattice	6	32	29	1.7



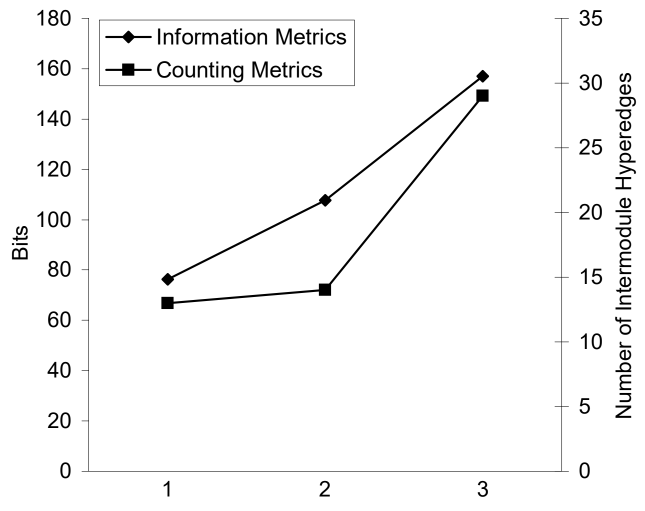
Module Id	1	2	3
Information Metrics	11.7	19.5	23.4
Counting Metrics	3	5	6

Figure 6.5 Module Size Comparison of Physics Program



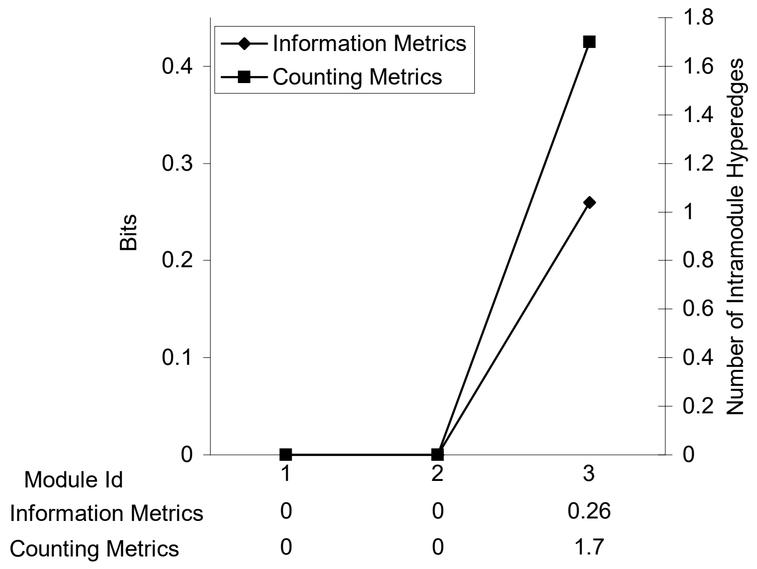
Module Id	1	2	3
Information Metrics	80.1	113.6	172.6
Counting Metrics	11	10	32

Figure 6.6 Module Complexity Comparison of Physics Program



Module Id	1	2	3
Information Metrics	76.4	107.7	157.1
Counting Metrics	13	14	29

Figure 6.7 Module Coupling Comparison of Physics Program



Module Id	1	2	3
Information Metrics	0	0	0.26
Counting Metrics	0	0	1.7

Figure 6.8 Module Cohesion Comparison of Physics Program

6.3 PMLP Examples

The Parallel Mathematical Libraries Project (PMLP) was developed cooperatively by Intel, Lawrence Livermore National Laboratory, the Russian Federal Nuclear Laboratory (VNIIEF), and the High Performance Computing Laboratory at 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 (ILU) preconditioners. Both Windows NT and Linux versions are available; we measured Linux version 3.0. PMLP was implemented in C++ using object-oriented techniques, such as template classes, generic programming, parameterized types, run-time polymorphism, compiler-time polymorphism, and iterators.

6.3.1 Data Collection

Hypergraphs were derived from the relationships between public variables and the methods that use them. The methods were represented by nodes, and each public variable was represented by a hyperedge. Each class was defined as a module. C++ system classes, methods, and public variables were excluded from the analysis. Similarly, some C++ files from the PMLP that had some missing header files, some with no classes, and some that had an error during compiling were also excluded.

Similar to the physics programs, selected C++ files are preprocessed using the gcc complier to generate a preprocessed (*.ii) file. The preprocessed file is parsed using the Datrix parser, which generates an abstract semantic graph (ASG). The ASG is provided to the Abstractor tool to generate a nodes $\times$ hyperedges table (*.nxe). The *.nxe file is given to the Measurement tool for the metric calculations. The results are further analyzed using the SAS tool. Figure 6.9 represents a nodes $\times$ hyperedges graph for dg_data_gen.cpp file. This file was selected as an example since it had multiple classes, one or more methods in each class, and few public variables.

6.3.2 Measurement

Table 6.10 represents the measures of size, Table 6.11 represents the measure of complexity, Table 6.12 represents the measure of coupling, and Table 6.13 represents the measure of cohesion of information theory-based metrics and counting-based metrics of different C++ files of the PMLP software.

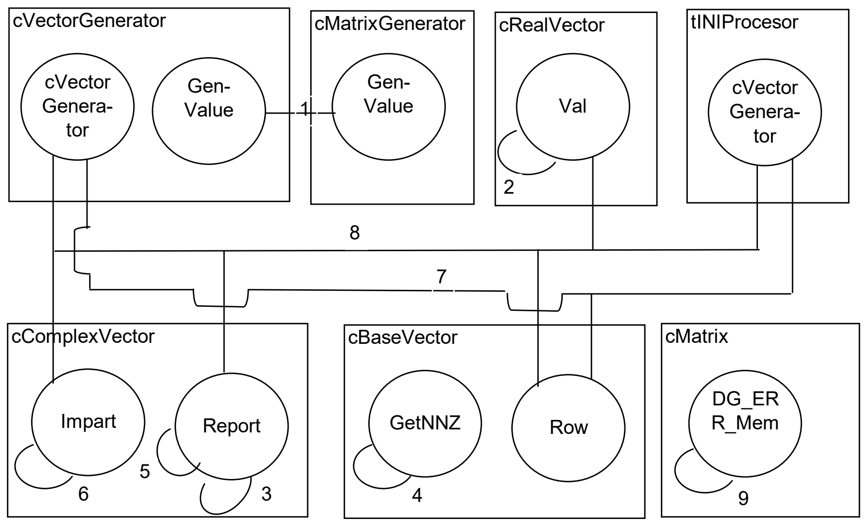


Figure 6.9 Nodes × Hyperedges Graph of dg_data_gen.cpp

Table 6.10 System Size of Selected PMLP Files

*.cpp	Information Metrics (bits)	Counting Metrics (nodes)
Test_Dlg	6.0	3
linear	8.9	5
linear_block	10.9	5
Scatter_block	10.1	6
grid	17.0	7
dg_matrix	21.0	7
dg_data_gen	27.8	10
HB_Util	34.6	10
general	29.4	11
dg_vec_gen	49.5	13
dg_mat_gen	54.7	14
dg_scal_gen	59.85	19
dg_vector	77.4	21

Table 6.11 System Complexity of Selected PMLP Files

*.cpp	Information Metrics (bits)	Counting Metrics (hyperedges)
Test_Dlg	2.5	33
linear	18.4	4
linear_block	9.8	7
Scatter_block	23.2	4
grid	68.0	5
dg_matrix	81.9	32
dg_data_gen	84.5	9
HB_Util	205.4	60
general	153.5	5
dg_vec_gen	266.5	62
dg_mat_gen	225.6	77
dg_scal_gen	402.2	14
dg_vector	1107.6	25

Table 6.12 System Coupling of Selected PMLP Files

*.cpp	Information Metrics (bits)	Counting Metrics (hyperedges)
Test_Dlg	0.0	0
linear	0.0	0
linear_block	0.0	0
Scatter_block	0.0	0
grid	68.0	5
dg_matrix	0.0	0
dg_data_gen	57.3	3
HB_Util	0.0	0
general	0.0	0
dg_vec_gen	93.0	5
dg_mat_gen	0.0	0
dg_scal_gen	361.7	8
dg_vector	1107.6	25

Table 6.13 System Cohesion of Selected PMLP Files

*.cpp	Information Metrics	Counting Metrics
Test_Dlg	0.21	11.00
linear	0.36	0.40
linear_block	0.19	0.70
Scatter_block	0.28	0.20
grid	0.00	0.00
dg_matrix	0.64	1.52
dg_data_gen	0.02	0.13
HB_Util	0.66	1.33
general	0.39	0.09
dg_vec_gen	0.16	0.70
dg_mat_gen	0.32	0.85
dg_scal_gen	0.01	0.04
dg_vector	0.00	0.00

Table 6.14 represents a nodes $\times$ hyperedges graph for the dg_data_gen.cpp file. Table 6.15 represents the information theory-based system-level metrics and the counting-based system-level metrics, and Table 6.16 presents the information theory-based module-level metrics and counting-based module-level metrics for the nodes $\times$ hyperedges table represented by Table 6.14.

6.3.3 Analysis

Figure 6.10 through Figure 6.13 represent the system-level comparison of information theory-based metrics and counting-based metrics of size, complexity, coupling, and cohesion, respectively. The information size and counting size are highly correlated, as were information coupling and counting coupling. Consequently, the order of modules is the

Table 6.14 Nodes $\times$ Hyperedges for dg_data_gen.cpp

Module	Node	Hyperedges
env	0	000000000
cVectorGenerator	GenValue	100000000
	CVectorGenerator	000000110
cMatrixGenerator	GenValue	100000000
cComplexVector	Report	001010010
	Impart	000001010
cBaseVector	GetNNZ	000100000
	Row	000000110
cMatrix	DG_Err_Mem	000000001
tINIProcessor	cVectorGenerator	000000110
cRealVector	Val	010000010

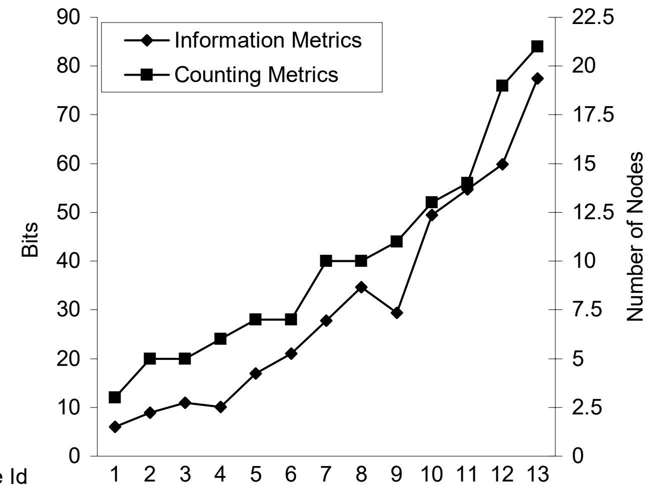
Table 6.15 System-level Measurements of dg_data_gen.cpp

	Information	Counting
Size	27.8 bits	10 nodes
Complexity	84.5 bits	9 edges
Coupling	57.3 bits	3 edges
Cohesion	0.02	0.13

Table 6.16 Module-level Measurements of dg_data_gen.cpp

	Information Theory-based Metrics			
Module	size (bits)	complexity (bits)	coupling (bits)	cohesion
cVectorGenerator	4.3	18.7	15.4	0.00
cMatrixGenerator	2.5	4.8	4.3	0.00
cComplexVector	6.9	20.5	10.2	0.10
cBaseVector	5.3	15.2	11.1	0.05
cMatrix	3.5	1.2	0.0	0.10
tINIProcessor	1.9	13.9	11.1	0.00
cRealVector	3.5	10.2	5.1	0.10
	Counting-based Metrics			
Module	size (nodes)	complexity (hyperedges)	coupling (hyperedges)	cohesion
cVectorGenerator	2	3	3	0.0
cMatrixGenerator	1	1	1	0.0
cComplexVector	2	4	2	3.0
cBasevector	2	3	2	∞
cMatrix	1	1	0	∞
tINIProcessor	1	2	2	0.0
cRealVector	1	2	1	∞

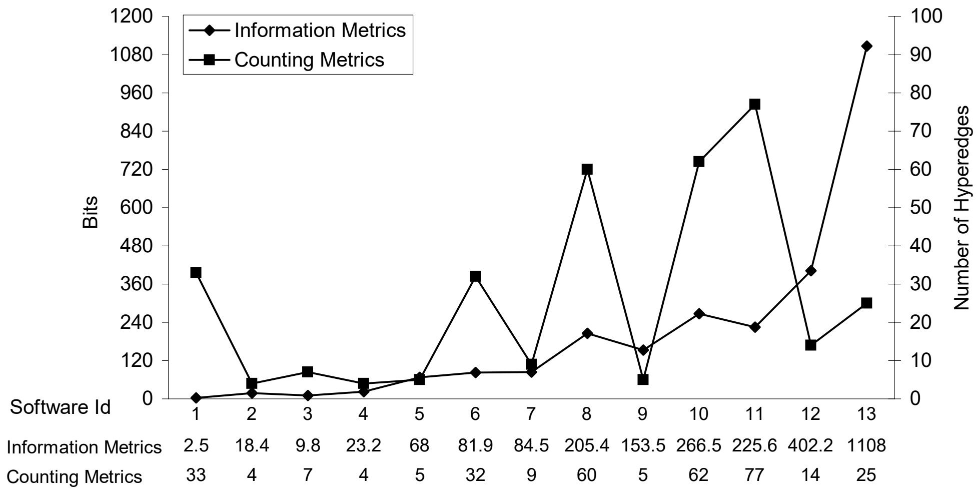
same according to the size or coupling attribute. In other words, they measure similar attributes in this example. However, the complexity measurements are not highly correlated. This means information complexity and counting complexity may be measuring different attributes. Coupling measurements of most of the files are zero, which means methods in those classes did not access public variables in other classes. Software engineers could use the nonzero coupling measurement to identify where public variables are used.



Software Id	1	2	3	4	5	6	7	8	9	10	11	12	13
Information Metrics	6	9	11	10	17	21	28	35	29	50	55	60	77
Counting Metrics	3	5	5	6	7	7	10	10	11	13	14	19	21

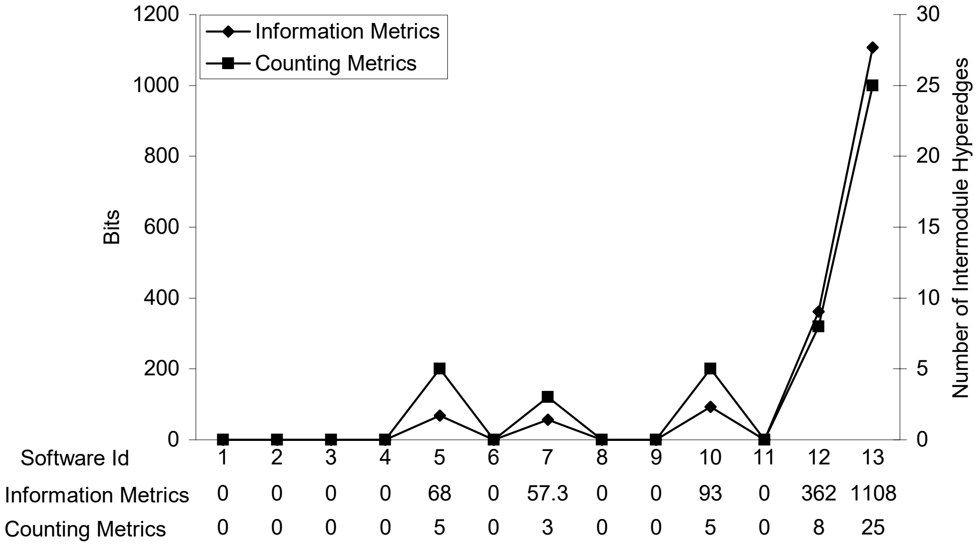
Figure 6.10 System-level Size Comparison of PMLP Files

Table 6.17 represents a graph for dgMatrix.cpp file. The Information complexity of this file is 81.0, and the counting complexity is 32. Table 6.18 and Table 6.19 provide evidence that when adding a hyperedge, the information complexity either increases or is the same, whereas in the case of the counting metrics the complexity will always increase



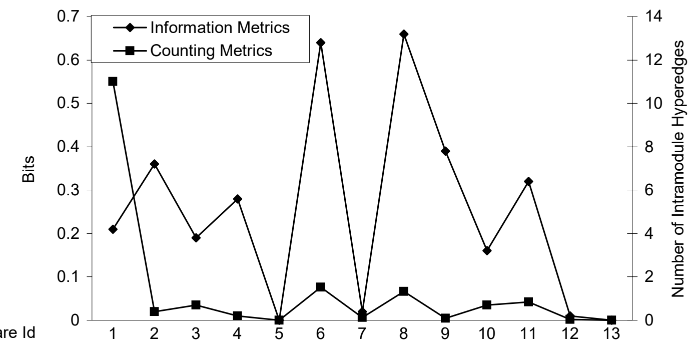
Software Id	1	2	3	4	5	6	7	8	9	10	11	12	13
Information Metrics	2.5	18.4	9.8	23.2	68	81.9	84.5	205.4	153.5	266.5	225.6	402.2	1108
Counting Metrics	33	4	7	4	5	32	9	60	5	62	77	14	25

Figure 6.11 System-level Complexity Comparison of PMLP Files



Software Id	1	2	3	4	5	6	7	8	9	10	11	12	13
Information Metrics	0	0	0	0	68	0	57.3	0	0	93	0	362	1108
Counting Metrics	0	0	0	0	5	0	3	0	0	5	0	8	25

Figure 6.12 System-level Coupling Comparison of PMLP Files



Software Id	1	2	3	4	5	6	7	8	9	10	11	12	13
Information Metrics	0.21	0.36	0.19	0.28	0	0.64	0.02	0.66	0.39	0.16	0.32	0.01	0
Counting Metrics	11	0.4	0.7	0.2	0	1.52	0.13	1.33	0.09	0.7	0.85	0.04	0

Figure 6.13 System-level Cohesion Comparison of PMLP Files

since we calculate the complexity by the number of hyperedges. For Table 6.18 the information theory complexity is 87.0, and the counting complexity is 33, and for Table 6.19 the information theory complexity is 81.0, the same as that represented by Table 6.17, and counting complexity is 33. It can therefore be noticed that information theory complexity measure provides an insight if there is a change in the measure in spite of adding a hyperedge, whereas counting metrics ignore this aspect.

Table 6.20 represents a graph for the grid.cpp file. The information coupling measure of this file is 68.0, and the counting coupling measure is 5. Table 6.21 represents a graph for the dg_scal_gen.cpp file. The information coupling measure for this file is 321.5, whereas the counting coupling measure is 5. Looking at the two graphs it can be noted that each system is coupled differently. The measure obtained from counting metrics ignores this fact, whereas information theory measure shows the differences between the two systems. If we remove an intermodule hyperedge from the given system, then the information coupling and information complexity decrease, information size either de-

Table 6.17 Nodes×Hyperedges Graph for dgMatrix.cpp

Module	Node	Hyperedges
env.	0	00000000000000000000000000000000
Matrix	sort	11111111110111101101111110101101
	Reorganizefull	00000100001101110011000001100011
	Reorganize	00000100001101110111000001100011
	∼cMatrix	00000100001101010011000001100011
	cMatrix	00000100001101010011000001110011
	fstream	00000000000100000000000000100001
	DG_ERR_Mem	00000000000000010000000000010001

Table 6.18 Information Complexity Increases When Adding a Hyperedge to Table 6.17

Module	Node	Hyperedges
env.	0	000000000000000000000000000000000
Matrix	sort	11111111110111101101111110101101**0**
	Reorganizefull	00000100001101110011000001100011**1**
	Reorganize	00000100001101110111000001100011**1**
	∼cMatrix	00000100001101010011000001100011**0**
	cMatrix	00000100001101010011000001110011**1**
	fstream	00000000000100000000000000100001**0**
	DG_ERR_Mem	00000000000000010000000000010001**0**

Table 6.19 Information Complexity Remains the Same When Adding a Hyperedge

Module	Node	Hyperedges
env.	0	000000000000000000000000000000000
Matrix	sort	11111111110111101101111110101101**1**
	Reorganizefull	00000100001101110011000001100011**1**
	Reorganize	00000100001101110111000001100011**1**
	∼cMatrix	00000100001101010011000001100011**1**
	cMatrix	00000100001101010011000001110011**1**
	fstream	00000000000100000000000000100001**0**
	DG_ERR_Mem	00000000000000010000000000010001**0**

creases, increases or remains the same, and the information cohesion remains the same or increases. In the case of the counting metrics the coupling and complexity decrease, cohesion either increases or remains the same, and the size always remains the same. It can be inferred that if an intermodule hyperedge is removed, then the size of the system either changes or remains the same, as shown by the information measures, whereas the counting measure shows that the size is always the same, which is a contradiction.

Table 6.20 Nodes$\times$Hyperedges Graph for grid.cpp

Module	Node	Hyperedges
env.	0	00000
Grid	psp_grid	11001
	operator=	00110
psp_grid	psp_grid	11001
	Get_coor	10110
	Inc	01000
	Dec	01000
	Get_all	01011

Table 6.21 Nodes×Hyperedges Graph for dg_scal_gen.cpp

Module	Node	Hyperedges
env.	0	0000000000000
cRealVector	val	1000000100000
CVectorGenerator	GenValue	1011000100001
	SetNRows	0001000000000
	∼cVectorGenerator	0001000000000
	cVectorGenerator	0000000100000
cMatrixGenerator	Genvalue	1011000110001
	SetSF	0001000000000
	SetNRows	0001000000000
	∼cMatrixGenerator	0001000000000
	FillFullBand	0000000000001
tINIProcesor	ProcessDG_BANDR_DIR	0100000001010
	cVectorGenerator	0000000100000
	ProcessDG_COL_DIR	0000000000011
cBaseVector	GetNNZ	0010000000000
	Row	0000000100000
cComplexVector	Repart	0000010100100
	Impart	0000100100000
cMatrix	DG_ERR_MEM	0001001000000
	∼cMatrix	0001000000000

Table 6.22 Statistical Analysis of testdlg.cpp with Two Modules

	Mean	Std. Dev	Min	Max	Skewness	Variance	Kurtosis	Median	Range	IQR
file1.cpp										
SystemInfoSize	6.00	0.00	6.00	6.00	—	0.00	—	6.00	0.00	0.00
SystemInfoComplexity	2.49	0.00	2.49	2.49	—	0.00	—	2.49	0.00	0.00
SystemInfoCoupling	0.00	0.00	0.00	0.00	—	0.00	—	0.00	0.00	0.00
SystemInfoCohesion	0.21	0.00	0.21	0.21	—	0.00	—	0.21	0.00	0.00
ModuleInfoSize	3.00	1.41	2.00	4.00	—	2.00	—	3.00	2.00	2.00
ModuleInfoComplexity	1.25	0.58	0.83	1.66	—	0.35	—	1.25	0.83	0.83
ModuleInfoCoupling	0.00	0.00	0.00	0.00	—	0.00	—	0.00	0.00	0.00
ModuleInfoCohesion	0.21	0.00	0.21	0.21	—	0.00	—	0.21	0.00	0.00
SystemCountingSize	3.00	0.00	3.00	3.00	—	0.00	—	3.00	0.00	0.00
SystemCountingComplexity	33.00	0.00	33.00	33.00	—	0.00	—	33.00	0.00	0.00
SystemCountingCoupling	0.00	0.00	0.00	0.00	—	0.00	—	0.00	0.00	0.00
SystemCountingCohesion	11.00	0.00	11.00	11.00	—	0.00	—	11.00	0.00	0.00
ModuleCountingSize	1.50	0.71	1.00	2.00	—	0.50	—	1.50	1.00	1.00
ModuleCountingComplexity	16.50	21.92	1.00	32.00	—	480.50	—	16.50	31.00	31.00
ModuleCountingCoupling	0.00	0.00	0.00	0.00	—	0.00	—	0.00	0.00	0.00
ModuleCountingCohesion	32.00	32.00	32.00	32.00	—	—	—	32.00	0.00	0.00

Std. Dev means Standard Deviation.

IQR means Interquartile Range.

Table 6.23 Statistical Analysis of grid.cpp with Two Modules

	Mean	Std. Dev	Min	Max	Skewness	Variance	Kurtosis	Median	Range	IQR
file1.cpp										
SystemInfoSize	17.00	0.00	17.00	17.00	—	0.00	—	17.00	0.00	0.00
SystemInfoComplexity	67.98	0.00	67.98	67.98	—	0.00	—	67.98	0.00	0.00
SystemInfoCoupling	67.98	0.00	67.98	67.98	—	0.00	—	67.98	0.00	0.00
SystemInfoCohesion	0.00	0.00	0.00	0.00	—	0.00	—	0.00	0.00	0.00
ModuleInfoSize	8.50	4.94	5.00	12.00	—	24.50	—	8.50	7.00	7.00
ModuleInfoComplexity	33.99	18.78	20.71	47.27	—	352.68	—	33.99	26.56	26.56
ModuleInfoCoupling	33.99	18.78	20.71	47.27	—	352.68	—	33.99	26.56	26.56
ModuleInfoCohesion	0.00	0.00	0.00	0.00	—	0.00	—	0.00	0.00	0.00
SystemCountingSize	7.00	0.00	7.00	7.00	—	0.00	—	7.00	0.00	0.00
SystemCountingComplexity	5.00	0.00	5.00	5.00	—	0.00	—	5.00	0.00	0.00
SystemCountingCoupling	5.00	0.00	5.00	5.00	—	0.00	—	5.00	0.00	0.00
SystemCountingCohesion	0.00	0.00	0.00	0.00	—	0.00	—	0.00	0.00	0.00
ModuleCountingSize	3.50	2.12	2.00	5.00	—	4.50	—	3.50	3.00	3.00
ModuleCountingComplexity	5.00	0.00	5.00	5.00	—	0.00	—	5.00	0.00	0.00
ModuleCountingCoupling	8.00	4.24	5.00	11.00	—	18.00	—	8.00	6.00	6.00
ModuleCountingCohesion	0.00	0.00	0.00	0.00	—	0.00	—	0.00	0.00	0.00

Std. Dev means Standard Deviation.
IQR means Interquartile Range.

Table 6.24 Statistical Analysis of dg_data_gen.cpp with Seven Modules

	Mean	Std. Dev	Min	Max	Skewness	Variance	Kurtosis	Median	Range	IQR
file1.cpp										
SystemInfoSize	27.84	0.00	27.84	27.84	—	0.00	—	27.84	0.00	0.00
SystemInfoComplexity	84.51	0.00	84.51	84.51	—	0.00	—	84.51	0.00	0.00
SystemInfoCoupling	57.26	0.00	57.26	57.26	—	0.00	—	57.26	0.00	0.00
SystemInfoCohesion	0.02	0.00	0.02	0.02	—	0.00	—	0.02	0.00	0.00
ModuleInfoSize	3.98	1.73	1.87	6.92	0.68	2.98	0.03	3.46	5.04	2.87
ModuleInfoComplexity	12.07	7.08	1.24	20.47	-0.49	50.24	-1.03	13.92	19.23	13.92
ModuleInfoCoupling	8.18	5.24	0.00	15.44	-0.31	27.47	-0.63	10.19	15.44	6.75
ModuleInfoCohesion	0.05	0.05	0.00	0.10	0.00	0.002	-2.60	0.05	0.10	0.10
SystemCountingSize	10.00	0.00	10.00	10.00	—	0.00	—	10.00	0.00	0.00
SystemCountingComplexity	9.00	0.00	9.00	9.00	—	0.00	—	9.00	0.00	0.00
SystemCountingCoupling	3.00	0.00	3.00	3.00	—	0.00	—	3.00	0.00	0.00
SystemCountingCohesion	0.13	0.00	0.13	0.13	—	0.00	—	0.13	0.00	0.00
ModuleCountingSize	1.43	0.53	1.00	2.00	0.37	0.29	-2.80	1.00	1.00	1.00
ModuleCountingComplexity	2.29	1.11	1.00	4.00	0.25	1.24	-0.94	2.00	3.00	2.00
ModuleCountingCoupling	1.57	0.98	0.00	3.00	-0.27	0.95	0.04	2.00	3.00	1.00
ModuleCountingCohesion	0.75	1.50	0.00	3.00	2.00	2.25	4.00	0.00	3.00	1.50

Std. Dev means Standard Deviation.
IQR means Interquartile Range.

Table 6.25 Statistical Analysis of dg_vec_gen.cpp with Four Modules

	Mean	Std. Dev	Min	Max	Skewness	Variance	Kurtosis	Median	Range	IQR
file1.cpp										
SystemInfoSize	49.49	0.00	49.49	49.49	—	0.00	—	49.49	0.00	0.00
SystemInfoComplexity	266.54	0.00	266.54	266.54	—	0.00	—	266.54	0.00	0.00
SystemInfoCoupling	92.98	0.00	92.98	92.98	—	0.00	—	92.98	0.00	0.00
SystemInfoCohesion	0.16	0.00	0.16	0.16	—	0.00	—	0.15	0.00	0.00
ModuleInfoSize	12.37	12.19	3.81	30.46	1.87	148.68	3.62	7.62	26.65	13.33
ModuleInfoComplexity	66.64	74.77	21.69	177.73	1.89	**5591.02**	3.59	33.56	156.04	88.61
ModuleInfoCoupling	23.24	12.93	14.14	41.56	1.43	**167.12**	1.57	18.64	27.42	18.22
ModuleInfoCohesion	0.09	0.15	0.00	0.31	1.94	0.02	3.81	0.03	0.31	0.16
SystemCountingSize	13.00	0.00	13.00	13.00	—	0.00	—	13.00	0.00	0.00
SystemCountingComplexity	62.00	0.00	62.00	62.00	—	0.00	—	62.00	0.00	0.00
SystemCountingCoupling	5.00	0.00	5.00	5.00	—	0.00	—	5.00	0.00	0.00
SystemCountingCohesion	0.73	0.00	0.73	0.73	—	0.00	—	0.73	0.00	0.00
ModuleCountingSize	3.25	3.20	1.00	8.00	1.87	10.25	3.62	2.00	7.00	3.50
ModuleCountingComplexity	17.75	27.51	3.00	59.00	1.99	**756.92**	3.98	4.50	56.00	28.50
ModuleCountingCoupling	5.75	4.27	3.00	12.00	1.73	**18.30**	2.92	4.00	9.00	5.50
ModuleCountingCohesion	1.31	1.14	0.00	2.00	-1.72	1.29	—	1.93	2.00	2.00

Std. Dev means Standard Deviation.
IQR means Interquartile Range.

Table 6.26 Statistical Analysis of dg_scal_gen.cpp with Seven Modules

	Mean	Std. Dev	Min	Max	Skewness	Variance	Kurtosis	Median	Range	IQR
file5.cpp										
SystemInfoSize	59.85	0.00	59.85	59.85	—	0.00	—	59.85	0.00	0.00
SystemInfoComplexity	402.37	0.00	402.37	402.37	—	0.00	—	402.37	0.00	0.00
SystemInfoCoupling	361.70	0.00	361.70	361.70	—	0.00	—	361.70	0.00	0.00
SystemInfoCohesion	0.01	0.00	0.01	0.01	—	0.00	—	0.01	0.00	0.00
ModuleInfoSize	8.55	2.99	4.32	12.90	0.08	8.97	-0.93	8.64	8.53	5.32
ModuleInfoComplexity	57.48	37.72	21.99	113.09	1.03	**1422.76**	-0.96	40.19	91.09	78.40
ModuleInfoCoupling	51.67	36.51	26.30	106.59	1.15	**1333.14**	-0.87	33.36	85.33	73.30
ModuleInfoCohesion	0.06	0.11	0.00	0.27	1.72	0.01	2.13	0.00	0.26	0.14
SystemCountingSize	19.00	0.00	19.00	19.00	—	0.00	—	19.00	0.00	0.00
SystemCountingComplexity	14.00	0.00	14.00	14.00	—	0.00	—	14.00	0.00	0.00
SystemCountingCoupling	8.00	0.00	8.00	8.00	—	0.00	—	8.00	0.00	0.00
SystemCountingCohesion	0.04	0.00	0.04	0.04	—	0.00	—	0.10	0.00	0.00
ModuleCountingSize	2.71	1.38	1.00	5.00	0.71	1.90	-0.32	2.00	4.00	2.00
ModuleCountingComplexity	4.71	2.63	2.00	8.00	0.36	**6.90**	-1.94	4.00	6.00	6.00
ModuleCountingCoupling	5.28	4.61	2.00	12.00	1.89	**21.24**	-0.87	3.00	10.00	10.00
ModuleCountingCohesion	0.86	1.46	0.00	3.00	1.23	2.14	-0.84	0.00	3.00	3.00

Std. Dev means Standard Deviation.
IQR means Interquartile Range.

Table 6.27 Statistical Analysis of dg_vector.cpp with Three Modules

	Mean	Std. Dev	Min	Max	Skewness	Variance	Kurtosis	Median	Range	IQR
file1.cpp										
SystemInfoSize	77.38	0.00	77.38	77.38	—	0.00	—	77.38	0.00	0.00
SystemInfoComplexity	1107.62	0.00	1107.62	1107.62	—	0.00	—	1107.62	0.00	0.00
SystemInfoCoupling	1107.62	0.00	1107.62	1107.62	—	0.00	—	1107.62	0.00	0.00
SystemInfoCohesion	0.00	0.00	0.00	0.00	—	0.00	—	0.00	0.00	0.00
ModuleInfoSize	25.79	8.42	17.54	34.38	0.18	70.96	25.46	16.84	16.84	
ModuleInfoComplexity	369.21	108.56	266.36	482.70	0.44	**11785.39**	—	358.56	216.34	216.34
ModuleInfoCoupling	369.21	108.56	266.36	482.70	0.44	**11785.39**	—	358.56	216.34	216.34
ModuleInfoCohesion	0.00	0.00	0.00	0.00	—	0.00	—	0.00	0.00	0.00
SystemCountingSize	21.00	0.00	21.00	21.00	—	0.00	—	21.00	0.00	0.00
SystemCountingComplexity	25.00	0.00	25.00	25.00	—	0.00	—	25.00	0.00	0.00
SystemCountingCoupling	25.00	0.00	25.00	25.00	—	0.00	—	25.00	0.00	0.00
SystemCountingCohesion	0.00	0.00	0.00	0.00	—	0.00	—	0.00	0.00	0.00
ModuleCountingSize	7.00	2.00	5.00	9.00	0.00	4.00	—	7.00	4.00	4.00
ModuleCountingComplexity	24.33	1.15	23.00	25.00	-1.73	**1.33**	—	25.00	2.00	2.00
ModuleCountingCoupling	57.67	12.58	46.00	71.00	0.58	**158.33**	—	56.00	25.00	25.00
ModuleCountingCohesion	0.00	0.00	0.00	0.00	—	0.00	—	0.00	0.00	0.00

Std. Dev means Standard Deviation.

IQR means Interquartile Range.

Table 6.28 summarizes the statistic variables used for the analysis.

Table 6.28 Statistic Variables

Variable	Definition
Mean	Commonly called the average (sum of all distribution divided by the number of distribution)
Standard Deviation	A measure of how spread out a distribution is (the square root of the variance)
Variance	A measure of how spread out a distribution is (the average squared deviation of each number from its mean)
Skewness	A measure of symmetry
Kurtosis	A measure of whether the distribution is peaked or flat relative to a normal distribution
Median	The central value, lying above and below half of the values.
Range	The difference between the largest and smallest values in the sample
Interquartile range	The difference between the upper and lower quartile

The SAS measurements were interesting only for the module-level metrics rather than the system-level, because the system-level metrics had only one observation. For the *ModuleSize* the mean, standard deviation and variance of the information metric were high compared to the counting metrics but were proportional. This shows that the counting metric is adequate. The skewness of all files and for both metrics were positive. This shows that all the values are bunched to the right of the mean. The kurtosis of information metric and counting metric for some files was opposite (either positive or negative), showing a contradiction between the two metrics. For the *ModuleComplexity*, the mean of the information metric was high and proportional to the counting metric, except for testdlg.cpp. From this it can be said that file1 may be having a same row pattern for some nodes, which may have reduced the complexity in case of information metrics but is ignored by the counting metrics. In such a case the information theory metrics are adequate. The standard deviation and variance were also high but not proportional. The skewness and kurtosis of the counting metric was opposite (either positive or negative) to the information metric in some of the files. For the *ModuleCoupling*, the mean of the information metric and the counting metric were proportional, while the standard deviation and variance were not proportional. The skewness was almost the same for both the information and the counting metric, but the kurtosis was opposite (either positive or negative) for some files. For the *ModuleCohesion* the mean, standard deviation and variance of the information metric were almost proportional to the counting metric except for testdlg.cpp which had a very high cohesion value of 32. The cohesion measure lies between zero and one. A couple of files showed that the counting cohesion was greater than 1. In such a case the information metric is adequate. The skewness and kurtosis of the counting metrics was opposite (either positive or negative) to the information metric in some of the files.

6.4 Analysis of Research Questions

The following are the research questions and answers that provide evidence for the hypothesis.

> Question 1. What are the similarities and differences between the distribution of information theory-based metrics and counting-based metrics?

Due to similarities one may use either the information metrics or the counting metrics. The differences make the analyst think about which metric is suitable for the analysis. From the case study of artificially generated examples, the similarity observed is that when adding a node the information size and the counting size increase. When adding an ordinary edge, the information complexity and counting complexity remain the same, and in the case of adding a hyperedge, the complexity of the two metrics increases. The difference observed between the two metrics is that the counting measurements are sensitive to multiple hyperedges connected to exactly the same nodes, while the information measurements are not.

From the case study of the physics program, the similarity observed is that when one uses a metric to order a module, the corresponding information metric and the counting metric generally result in the same order. From the case study of the PMLP software, the similarity observed is that the information size and counting size are highly correlated, as were information coupling and counting coupling. Consequently, the order of modules is the same according to the size or coupling attributes. In other words, they measure similar attributes. The difference observed is that the information complexity and the counting complexity are not highly correlated. This means that the two measurements are measuring different attributes.

> Question 2. Do the distributions of measurement values yield insight into the software development process and resulting product attributes?

The results from the case studies of the physics program and PMLP provide an insight into the software development process and resulting product attributes. In the case study of the physics programs, it is observed that one module had more complexity than the other modules in the system, while another module contributed more to all the system-level metrics. In the PMLP case study, it is observed that the complexity measurements of some files were high in spite of the files having a low size measure. It is also observed that most of the files had zero coupling, meaning that variables were declared public but were not used as public. Software engineers could use the nonzero coupling measurement to identify where public variables are used.

> Question 3. Does each information theory-based measure preserve our intuition about its attribute?

Almost all of the information theory-based measures satisfy the properties defined by Briand, Morasca and Basili [11]. There were few exceptions, such as the module complexity resulted in a negative measure, and the counting cohesion measured greater than one in some cases.

> Question 4. Does the measurement instrument (tool) precisely specify how to capture measurement data?

The measurement tool calculates the system-level and module-level measurements of size, complexity, coupling, and cohesion of both information theory-based metrics and counting-based metrics. The tool also specifies how to capture the data.

> Question 5. Does the measurement protocol (procedure) assure consistent, repeatable measurements that are independent of the measurer and the measurement environment?

The results of three case studies provide evidence that the measurement protocol assures consistent, repeatable measurements that are independent of the measurer and the measurement environment.

CHAPTER VII

CONCLUSIONS

7.1 Evaluation of Hypothesis

The hypothesis of this research is

> Information theory-based software metrics proposed by Allen [3], namely, size, complexity, coupling, and cohesion, can be useful in real-world software development projects, compared to the counting-based metrics.

The study included an analysis of three case studies. The research questions answered in Chapter VI provide some evidence for the hypothesis.

> Question 1. What are the similarities and differences between the distribution of information theory-based metrics and counting-based metrics?

Similarities were seen in the size measure and the coupling measure. One can either use information metrics or counting metrics for the calculation of the size and the coupling attributes because of the similarities. In case of the complexity measure, information metrics are more sensitive than counting metrics because counting metrics just count the number of hyperedges, whereas information metrics are sensitive to the configuration of the hypergraphs. However, there is a drawback in the module-level complexity measure based on information theory. For certain conditions the modular complexity measure is negative, which is not desirable according to Briand, Morasca, and Basili [11]. In the case of cohesion, the information metric is useful compared to the counting metric because the cohesion measure is a factor that lies between zero and one. Counting cohesion measurement may have values greater than one, which is not a desirable property according to Briand, Morasca, and Basili [11].

> Question 2. Do the distributions of measurement values yield insight into the software development process and resulting product attributes?

The case study found that the distribution of measurement values does provide insight into the development process and the resulting product attributes such as the nonzero coupling measure in the PMLP software.

> Question 3. Does each information theory-based measure preserve our intuition about its attribute?

Each information theory-based measure preserves our intuition about its attribute except for module complexity, which could have negative values in certain circumstances.

> Question 4. Does the measurement instrument (tool) precisely specify how to capture measurement data?

The measurement instrument is software, therefore how to capture measurement data is not ambiguous.

> Question 5. Does the measurement protocol (procedure) assure consistent, repeatable measurements that are independent of the measurer and the measurement environment?

The measurement protocol formally assures consistent, repeatable measurements that are independent of the measurer and the measurement environment because our measurement procedures do not require subjective decisions and they make extensive use of software tools.

One may prefer the information complexity and cohesion measures over the counting metrics because the complexity measure is sensitive to configuration of the hypergraphs and the cohesion measure lies in the range zero to one. Since the size and coupling measures of the information metrics and counting metrics are similar, one may prefer to use the counting metrics because counting the number of nodes or the number of intermodule hyperedges is easier. The information theory-based measures made finer-grain distinctions. Discovery of rare module attributes might require exploiting the finer-grain distinctions offered by the information theory-based metrics in conjunction with the coarser counting-based metrics.

7.2 Future Work

The complexity metrics based on information theory are better than the corresponding counting metrics except for the drawback of the negative module complexity. Future work may mathematically prove our conjecture of conditions that make the metric negative. Further the formula for the counting cohesion measure should be revised so that the measurements lie between zero and one. Additional case studies should be done to further evaluate the hypothesis and the factor of fault-proneness in relation to the metrics may be investigated.

REFERENCES

[1] E. B. Allen, *Information Theory and Software Measurements*, doctoral dissertation, Florida Atlantic University, Boca Raton, Florida, Aug 1995.

[2] E. B. Allen, “Empirical Validation of Information Theory-Based Software Metrics,” Proposal for National Science Foundation, Sept. 2001.

[3] E. B. Allen, “Measuring Graph Abstraction of Software: An Information-Theory Approach,” *Proceedings: Eighth IEEE Symposium on Software Metrics*, Ottawa, Canada, June 2002, IEEE Computer Society, pp. 182–193.

[4] E. B. Allen and S. Gottipati, *Measuring Size, Complexity, and Coupling of Hypergraphs Abstraction of Software: An Information-Theory Approach*, Tech. Rep. MSU-021219, Mississippi State University, Mississippi, Dec 2002.

[5] E. B. Allen and T. M. Khoshgoftaar, “Measurement of Software Design Cohesion,” *Proceedings of 5th ISSAT International Conference on Reliability and Quality in Design*, Las Vegas, Nevada, Aug. 1999, International Society of Science and Applied Technologies, pp. 158–163.

[6] E. B. Allen and T. M. Khoshgoftaar, “Measuring Coupling and Cohesion: An Information-Theory Approach,” *Proceeding: Sixth International Software Metrics Symposium*, Boca Raton, Florida, Nov. 1999, IEEE Computer Society, pp. 119–127.

[7] 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.

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

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

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

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

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

[13] N. E. Fenton and S. L. Pfleeger, *A Rigorous and Practical Approach*, 2nd edition, PWS Publishing Company, Boston, Massachusetts, Jan. 1997.

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

[15] R. A. Johnson and D. W. Wichern, *Applied Multivariate Statistical Analysis*, 2nd edition, Prentice-Hall, Inc, Englewood Cliffs, New Jersey, 1988.

[16] S. Jordan, *Software Metrics Collection: Two New Reasearch Tools*, master's thesis, Florida Atlantic University, Boca Raton, Florida, Dec. 1997.

[17] T. M. Khoshgoftaar and E. B. Allen, "A Practical Classification Rule for Software Quality Models," *IEEE Transaction on Reliability*, vol. 49, no. 2, June 2000, pp. 209–216.

[18] B. A. Kitchenham, S. L. Pfleeger, and N. E. Fenton, "Towards a Framework for Software Measurement Validation," *IEEE Transaction on Software Engineering*, vol. 21, no. 12, Dec. 1995, pp. 929–944, See comments in [19] and [21].

[19] B. A. Kitchenham, S. L. Pfleeger, and N. E. Fenton, "Reply to: Comments on 'Towards a Framework for Software Measurement Validation'," *IEEE Transaction on Software Engineering*, vol. 23, no. 3, Mar. 1997, p. 189.

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

[21] S. Morasca, L. C. Briand, V. R. Basili, E. J. Weyuker, and M. V. Zelkowitz, "Comments on 'Towards a Framework for Software Measurement Validation'," *IEEE Transaction on Software Engineering*, vol. 23, no. 3, Mar. 1997, pp. 187–188, See [18].

[22] S. L. Pfleeger, R. Jeffery, B. Curtis, and B. A. Kitchenham, "Status Report on Software Measurement," *IEEE Software*, vol. 14, no. 2, Mar. 1997, pp. 33–43.

[23] G. Poels and G. Dedene, "Comments on 'Property-Based Software Engineering Measurement': Refining the Additivity Properties," *IEEE Transaction on Software Engineering*, vol. 23, no. 3, Mar. 1997, pp. 190–195, See [11].

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

[25] C. E. Shannon and W. Weaver, *The Mathematical Theory of Communication*, University of Illinois Press, Urbana, Illinois, 1949.

[26] F. Shull, F. Lanubile, and V. R. Basili, "Investigating Reading Techniques for Object-Oriented Framework Learning," *IEEE Transaction on Software Engineering*, vol. 26, no. 11, Nov. 2000, pp. 1101–1118.

[27] M. H. van Emden, "Hierarchical Decomposition of Complexity," *Machine Intelligence*, 1970, pp. 5:361–380.

[28] H. Zuse, "Reply to 'Property-Based Software Engineering Measurement'," *IEEE Transaction on Software Engineering*, vol. 23, no. 8, Aug. 1997, p. 533, See [11].