Measuring Coupling and Cohesion of Software Modules: An Information-Theory Approach

Edward B. Allen
Mississippi State University
Mississippi USA

Taghi M. Khoshgoftaar*
Ye Chen
Florida Atlantic University
Boca Raton, Florida USA

Abstract

Coupling of a subsystem characterizes its interdependence with other subsystems. A subsystem's cohesion, on the other hand, characterizes its internal interdependencies. When used in conjunction with other attributes, measurements of a subsystem's coupling and cohesion can contribute to software quality models.

An abstraction of a software system can be represented by a graph and a module (subsystem) by a subgraph. Software-design graphs depict components and their relationships. Prior work by Allen and Khoshgoftaar proposed information theory-based measures of coupling and cohesion of a modular system. This paper proposes related information theory-based measures of coupling and cohesion of a module. These measures have the properties of module-level coupling and cohesion defined by Briand, Morasca, and Basili. We define cohesion of a module in terms of intramodule coupling, normalized to between zero and one.

We illustrate the measures with example graphs and an empirical analysis of the call graph of a moderate-size C program, the Nethack computer game. Preliminary analysis showed that the information-theory approach has finer discrimination than counting.

Keywords: *software metrics, coupling, cohesion, call graph, information theory, entropy, excess entropy*

1. Introduction

Identification of components and their relationships is the essence of high-level software design. Graphs are often used to depict various aspects of object-oriented software architectures [31]. Similarly, a structure chart, depicts the hierarchy of control among procedural components. At this level, design decisions consist of identifying nodes in a software design diagram and drawing a structure of relationships.

The software engineering community often discusses the complexity, coupling, or cohesion of software designs. Software-design text books admonish would-be designers to "reduce coupling among modules and improve cohesion within modules" [26, 28]. Such dictums are difficult to put into practice without measurement of relevant attributes. Measurements early in the design process can give insight into design attributes, and may be inputs to models that predict software quality. If the attributes or predicted quality are unsatisfactory, then one can revise the design before changes become too expensive. Recent research on coupling and cohesion has focused on attributes of object-oriented designs [4, 7, 8, 11, 15, 16].

We have previously defined information theory-based measures of coupling and cohesion of a system as a whole [2]. This paper extends that work, proposing related coupling and cohesion measures for each module in the context of its system.

Briand, Morasca, and Basili, propose a set of properties in a formal framework to define coupling and cohesion more precisely [9]. *Coupling* and *cohesion* are the names of families of measures on graphs that have certain properties in common. More recently, Morasca and Briand extended their framework from graphs (binary relations) to relations in general [24]. In this paper, we adopt their definitions, and propose spe-

*Readers may contact the authors through Taghi M. Khoshgoftaar, Empirical Software Engineering Laboratory, Dept. of Computer Science and Engineering, Florida Atlantic University, Boca Raton, FL 33431 USA. Phone: (561)297-3994, Fax: (561)297-2800, Email: taghi@cse.fau.edu, URL: www.cse.fau.edu/esel/. This work was done while Edward B. Allen was at Florida Atlantic University.

1530-1435/01 $10.00 © 2001 IEEE

cific measures at the module level based on information theory, namely, intermodule coupling, intramodule coupling, and cohesion. They can be used to measure many kinds of software design abstractions, and thus, many kinds of software coupling. We illustrate the measures with examples and an empirical study.

Information theory is a promising foundation for software measurement that accounts for patterns in software attributes [1]. Khoshgoftaar and Allen's survey [19] discusses numerous proposed software metrics based on information theory. Several are entropy-based measures of graph attributes, such as the entropy of control-flow graph nodes by their degree [14], the entropy of a data structure graph by its topology [22], and the entropy of object-oriented class interfaces by keywords [3]. In this paper, we focus on information in graphs represented by an object-predicate table [32].

The remainder of this paper summarizes the underlying definitions of Briand, Morasca, and Basili [9], presents our abstractions of a software system, explains relevant information theory and its application to the problem at hand, and defines coupling of a module, intramodule coupling of a module, and cohesion of a module. Finally, we present an empirical case study of the call graph of a computer game consisting of over 1,600 functions in C. The case study compared the variation of the proposed information theory-based measures to similar measures based on counting.

2. Related work

Briand, Morasca, and Basili [9] define a *modular system* as the context for coupling and cohesion, where a module is a subsystem.

Definition 1 (Modular System [9])
A modular system, *MS*, *is a special case of a software system represented by a graph,* **S**, *that has n nodes partitioned into modules,* $m_k, k = 1, \ldots, n_M$.

Figure 1 is an example of a modular-system graph. This is similar to Figure 2 in [9]. The modules (dashed boxes) partition the nodes in the system.

Briand, Morasca, and Basili [9] define families of measures through sets of properties for size, length, complexity, coupling, and cohesion. This paper builds on their definitions for coupling and cohesion at the module level. Note that an *intermodule edge* has end points in different modules. Conversely, an *intramodule edge* has end points in the same module. We reserve the term *coupling of a module* for measures that have the properties in Table 1, which are essentially the same as in [9]. However, we substitute "intermodule" for "output" edge in Properties 2 and 3. Practically speaking,

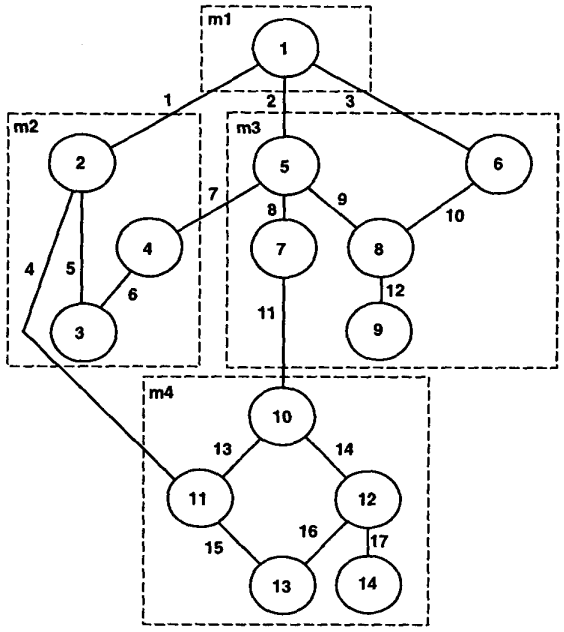


Figure 1. Example of a modular system [2]

Table 1. Properties of coupling of a module [9]

Concept/Properties
1. Nonnegativity. Coupling of a module is nonnegative.
2. Null value. Coupling of a module is null if its set of intermodule edges is empty.
3. Monotonicity. Adding an intermodule edge to a module does not decrease its module coupling.
4. Merging of modules. If two modules, m_1 and m_2, are merged to form a new module, $m_{1\cup 2}$, that replaces m_1 and m_2, then the module coupling of $m_{1\cup 2}$ is not greater than the sum of the module couplings of m_1 and m_2.
5. Disjoint module additivity. If two modules, m_1 and m_2, which have no intermodule edges between nodes in m_1 and nodes in m_2, are merged to form a new module, $m_{1\cup 2}$, that replaces m_1 and m_2, then the module coupling of $m_{1\cup 2}$ is equal to the sum of the module couplings of m_1 and m_2.

Table 2. Properties of cohesion of a module [9]

Concept/Properties
1. Nonnegativity and Normalization. Cohesion of a module belongs to a specified interval, $[0, \text{Max}]$.
2. Null value. Cohesion of a module is null if its set of intramodule edges is empty.
3. Monotonicity. Adding an intramodule edge to a module does not decrease its module cohesion.
4. Merging of modules. If two unrelated modules, m_1 and m_2, are merged to form a new module, $m_{1\cup 2}$, that replaces m_1 and m_2, then the module cohesion of $m_{1\cup 2}$ is not greater than the maximum of the module cohesion of m_1 and m_2.

our only difference with Briand, Morasca, and Basili, is we make no distinction between inbound and outbound coupling. Briand, Morasca, and Basili [9] also propose the set of properties shown in Table 2 that defines the concept *cohesion of a module*. These families of measures of coupling and cohesion on undirected graphs are readily applied to a variety of software engineering abstractions.

Our prior work [2] defined information theory-based measures of coupling and cohesion of a modular system as a whole. This paper proposes related measures for each module in the context of its modular system.

3. Measurement protocol

A *measurement protocol* is a set of conditions that assures consistent repeatable measurement of an attribute [20]. A valid protocol is independent of the measurer and the measurement environment. A valid protocol is also compatible with the desired unit of measure and the purpose of the measurement.

Given a software system, various protocols derive abstractions that are suitable for measurement. For example, Briand, Daly, and Wüst [7, 8] survey numerous proposed measures of coupling and cohesion for object-oriented designs. Various abstractions are measured, such as class inheritance, class type, method invocation, and class-attribute references, and thus, various software attributes are measured. Bieman et al. propose cohesion measures based on input/output dependence graphs of designs [5], and slicing of code [6]. Our approach is compatible with all of these abstractions, as well as graphical abstractions, such as call graphs, for the procedural development paradigm.

We begin with any measurement protocol that produces a graph representing some aspect of software design. Many design methods produce graphs as artifacts. An artifact created during the design phase represents the design decisions made at that time, and embodies the abstraction of the software that was used by the designer.

Coupling and cohesion are defined by Briand, Morasca, and Basili [9] in the context of a modular-system graph. So the software-design graph should be partitioned into subsystems (modules). We define a *system graph* to model explicitly the lack of relationship between the system and its environment.

Definition 2 (System graph)
Given a modular system, MS, with n nodes partitioned into n_M modules, its system graph, *S, is all nodes in MS and all its edges, plus a disconnected node representing the system's environment. Without loss of generality, index the environment node as $i = 0$, and the nodes in MS as $i = 1, \ldots, n$.*

The system scope is defined by the given nodes and edges. We explicitly represent the unspecified environment by a single disconnected node. For measurement of coupling, we make a further abstraction, an *intermodule-edges graph*, which is a subgraph of S. Note that the properties of coupling in Table 1 focus on intermodule edges.

Definition 3 (Intermodule-edges graph)
Given a modular system, MS, and its system graph, S, its intermodule-edges graph, *S^*, consists of all nodes in S and all its intermodule edges.*

It is not necessary for subgraph S^* to be a connected graph. For example, Figure 2 depicts the intermodule-edges subgraph, S^*, for the modular system in Figure 1.

For measurement of cohesion, we make a further abstraction, an *intramodule-edges graph*, which is a subgraph of S. Note that the properties of cohesion in Table 2 focus on intramodule edges.

Definition 4 (Intramodule-edges graph)
Given a modular system, MS, and its system graph, S, its intramodule-edges graph, *S°, consists of all nodes in S and all its intramodule edges.*

Similarly, it is not necessary for subgraph S° to be a connected graph. For example, Figure 3 depicts the intramodule edges subgraph, S°, for the modular system in Figure 1.

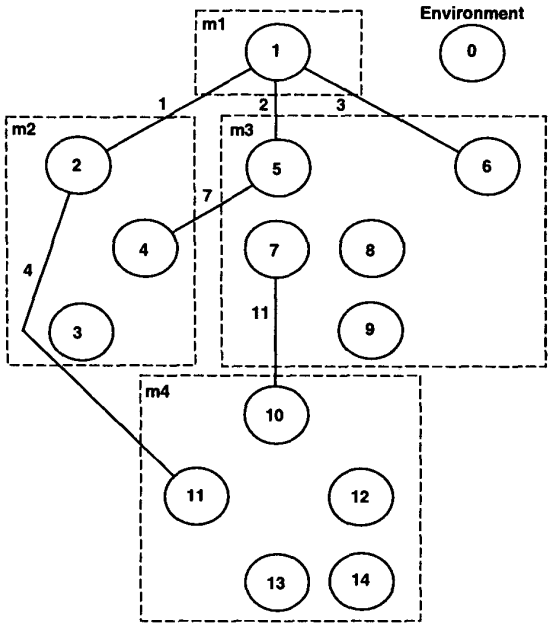


Figure 2. Example intermodule-edges graph [2]

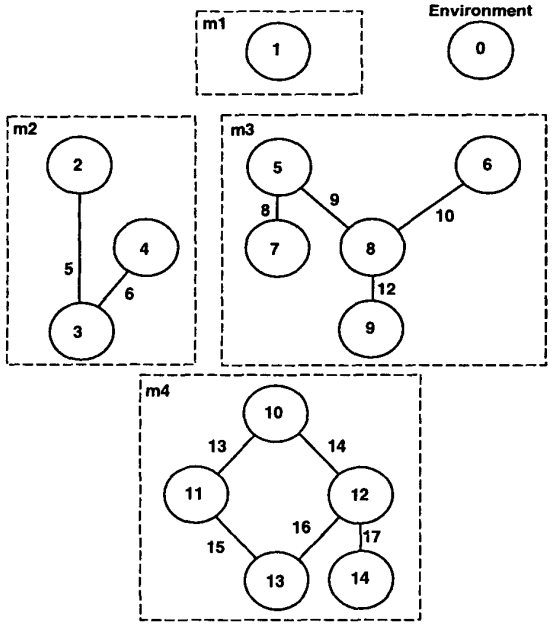


Figure 3. Example intramodule-edges graph [2]

Table 3. Example intermodule-edges graph [2]

		Edge						
Module	Node	1	2	3	4	7	11	$p_{L(i)}$
Environ.	0	0	0	0	0	0	0	0.467
m_1	1	1	1	1	0	0	0	0.067
m_2	2	1	0	0	1	0	0	0.067
	3	0	0	0	0	0	0	0.467
	4	0	0	0	0	1	0	0.067
m_3	5	0	1	0	0	1	0	0.067
	6	0	0	1	0	0	0	0.067
	7	0	0	0	0	0	1	0.133
	8	0	0	0	0	0	0	0.467
	9	0	0	0	0	0	0	0.467
m_4	10	0	0	0	0	0	1	0.133
	11	0	0	0	1	0	0	0.067
	12	0	0	0	0	0	0	0.467
	13	0	0	0	0	0	0	0.467
	14	0	0	0	0	0	0	0.467

In order to analyze the patterns of relationships in S^* or S°, we label each node with the set of edges that are incident to it. Because diagrams in software engineering, such as structure charts, identify each node with a component name, we choose an abstraction that maintains a distinct identity for each relationship, rather than considering topology alone [22]. A convenient representation of the graph is a nodes × edges table where each cell indicates whether the node is an end point of the edge, or not. A nodes × edges table fully specifies an undirected graph. Each node's label is encoded as the binary pattern of values in a row of the table. Table 3 shows the nodes × edges table corresponding to Figure 2. (Column $p_{L(i)}$ is explained below.)

The abstraction of a nodes × edges table is an instance of an object-predicate table, where nodes are objects and each predicate is of the form, "Is this node related to another node by this edge?" Object-predicate tables are useful for analysis of complex relationships among objects [32]. Because object-predicate tables are suitable for relations in general, future research will extend this to more general measurement protocols [24].

In summary, we begin with any protocol that results in a graphical abstraction of a modular system; we make a further abstraction to an intermodule-edges graph, S^*, or an intramodule-edges graph, S°; and then translate that subgraph into a nodes × edges table.

4. Measurement instrument

A *measurement instrument* is a tool for mapping an abstraction of software to a number or category [20]. An instrument is usually designed to detect a particular unit of measure, and assures that each unit of an attribute is equivalent within the constraints of measurement error. We adopt information theory [13, 30] as the basis for our measurement instrument, because we view the design decisions embodied by the graph as information [2].

Software design is an intellectual process. Since human short-term memory capacity is limited [23], we suspect that one important source of mistakes is information overload. A designer may not consider all relevant information when making a decision, because the amount of information is too large to work with in short-term memory. Our working hypothesis is that assessment of designs in terms of amounts of various kinds of information can indicate potential information overload. Information is stored in human short-term and long-term memory when a person comprehends a design abstraction. The amount of such information may have useful statistical relationships with quality factors [17]. Future empirical research will investigate the relationship between information overload and software quality.

Most traditional software metrics count artifact features, as though each item is equal in the mind of a designer. For example, Briand, Daly, and Wüst [8] propose an integrated measurement framework for object-oriented coupling metrics, based on counts. The framework successfully encompasses many metrics proposed in the literature. Each qualifying metric is equivalent to the number of edges of an appropriate graph. They also propose a similar framework for object-oriented cohesion metrics [7]. We suspect that an approach that is more sophisticated than counting will be useful. Information theory is attractive, because it measures symbolic content in a way that accounts for redundancy and patterns. Repetitive patterns are easy for designers to remember, but counts do not reflect this. According to information theory, repetitive patterns (i.e. high probability) have low information content, because the symbolic content can be compactly described [13]. The foundation of our measurement instrument is information theory, rather than just counting.

The notions of coupling and cohesion are based on relationships among nodes in a modular system [9]. In the following, S represents the nodes $\times$ edges table of any of the graphs S, S^*, or S°. Let us model S as a probability distribution on the row patterns of its nodes $\times$ edges table, $p_l, l = 1, \ldots, n_S$, where n_S is the number of possible distinct row patterns. The *entropy* of the distribution of row patterns [30] is the following.

$$H(S) = \sum_{l=1}^{n_S} (-p_l \log p_l) \quad (1)$$

In this application, entropy is the average information per node. Entropy is always nonnegative. When logarithms are base two, the unit of measure is a bit. All logarithms in this paper are base two.

When we estimate the distribution by the proportions of distinct row patterns that exist in S [32], entropy can be estimated by the following.

$$H(S) = \sum_{i=0}^{n} \frac{1}{n+1} \left(-\log p_{L(i)}\right) \quad (2)$$

where the summation is over the individual nodes, and $L(i)$ is a function that gives the pattern index, l, of the i^{th} row. Table 3 lists $p_{L(i)}$ for each row of the example; each estimated $p_{L(i)}$ in the table is the proportion of that row's pattern out of fifteen rows. If a graph S_0 has no edges, its nodes $\times$ edges table is equivalent to one with an arbitrary number of columns and zeros in all cells, and thus, all patterns are the same (zeros), $p_l = 1$ and $H(S_0) = 0$. A graph S with redundant edges has the same entropy, $H(S)$, as a similar graph without redundant edges. Thus, our measurement instrument is not sensitive to multiple edges between the same pair of nodes. In other words, our approach is appropriate when multiple edges between the same pair of nodes are not interesting.

Because communications systems are an important application of information theory, many theoretical results are concerned with finding optimal ways to code information for efficient transmission. An *instantaneous code* is a set of strings (code words) where no string in the set is a prefix of another. For a discrete random variable x, let $E(\text{Len}(x))$ be the expected length of an instantaneous code for a sample from the domain of x. Information theory gives the following result [13, p.86].

$$E(\text{Len}(x)) \geq H(x) \quad (3)$$

where the base of the logarithm of entropy in Equation (1) is the number of symbols in the code alphabet. In other words, the entropy is the minimum expected length of an instantaneous code for one sample object. In our application, a node corresponds to a sample object. Because row patterns of our nodes $\times$ edges tables are encoded by the alphabet $\{0, 1\}$, we use base 2 logarithms to calculate entropy. If we use an instantaneous code to describe row patterns, the *minimum expected*

length of a description of a set S consisting of $n+1$ nodes is the following.

$$I(S) = (n+1)\,H(S) \quad (4)$$

$$I(S) = \sum_{i=0}^{n} (-\log p_{L(i)}) \quad (5)$$

We interpret $I(S)$ as the amount of *information* in the set S.

Definition 5 (Node subgraph)
Given a graph S, the subgraph S_i consists of all the nodes in S and the edges of S that have the i^{th} node as an end point.

Disconnected nodes are included in this subgraph. Similar to S, we label each node with the set of edges incident to it, and we represent S_i by a nodes × edges table. We also model S_i as a probability distribution on its row patterns, estimated by the proportions of distinct row patterns, p_l. Similar to Equation (2), the entropy of the distribution of row patterns is the following.

$$H(S_i) = \sum_{j=0}^{n} \frac{1}{n+1} (-\log p_{L_i(j)}) \quad (6)$$

where $L_i(j)$ is a function that gives the pattern index, l, of the j^{th} row of S_i. The minimum expected length of a description of S_i is the subgraph's information.

$$I(S_i) = \sum_{j=0}^{n} (-\log p_{L_i(j)}) \quad (7)$$

Because it is an information theory-based measure, the amount of information accounts for patterns in relationships, and consequently, discriminates among graph structures in a way that counting measures cannot.

5. Coupling of a module

Coupling of a module is meaningful in the context of its modular system. It quantifies relationships between the module and the rest of the system by measurement of the intermodule-edges graph. Using information theory as the basis for measurement, we define coupling of a module as the sum of the amount of information in the node subgraphs of all nodes in the module.

Definition 6 (Coupling of a module)
Given a modular system, MS, its intermodule-edges graph, S^*, and a module, m_k, the coupling of the module is given by

$$Coupling(m_k|MS) = \sum_{i \in m_k} I(S_i^*)$$

Table 4. Properties of intramodule coupling of a module

Concept/Properties
1. Nonnegativity. Intramodule coupling of a module is nonnegative.
2. Null value. Intramodule coupling of a module is null if its set of intramodule edges is empty.
3. Monotonicity. Adding an intramodule edge to a module does not decrease the intramodule coupling value.
4. Merging of modules. If two modules, m_1 and m_2, are merged to form a new module, $m_{1\cup 2}$, that replaces m_1 and m_2, then the intramodule coupling of module $m_{1\cup 2}$ is not less than the sum of the intramodule couplings of modules m_1 and m_2.
5. Disjoint module additivity. If two modules, m_1 and m_2, which have no intermodule edges between nodes in m_1 and nodes in m_2, are merged to form a new module, $m_{1\cup 2}$, that replaces m_1 and m_2, then the intramodule coupling of module $m_{1\cup 2}$ is equal to the sum of the intramodule couplings of modules m_1 and m_2.

The unit of measure is a bit, which is a conventional measure of information.

Briand, Daly, and Wüst [8] found that many metrics claim to measure some aspect of "coupling", but some do not fulfill all the properties specified in Table 1. We prefer to reserve the name *coupling of a module* for those measures that have all the specified properties.

Theorem 1 (Coupling of a module)
$Coupling(m_k|MS)$ has the properties specified in Table 1, and thus, is a member of the family of measures called "coupling of a module".

Chen [12] gives a proof.

6. Intramodule coupling of a module

Ordinary coupling is an attribute of intermodule relationships. We propose a measure called *intramodule coupling of a module*, which is closely related to the ordinary coupling family. We transformed the properties of coupling of a module in Table 1 into the properties of intramodule coupling in Table 4.

The following is a summary of the differences [2]. Given the properties in Table 1, (1) we substitute

IntramoduleCoupling for *Coupling*; (2) we substitute "intra-" for "inter-"; and (3) we change the direction of the inequality in Property 4, so that instead of "not greater than", it reads "not less than". This inequality accounts for the possibility that an intermodule edge between two modules may become an intramodule edge when they are merged. Otherwise, the properties of intramodule coupling are the same as those of coupling. We reserve the term *intramodule coupling of a module* for measures that have these properties.

Intramodule coupling is applicable to the same software engineering abstractions as ordinary coupling. Our measurement protocol for intramodule coupling is similar to ordinary coupling's protocol, except that we consider intramodule edges instead of intermodule edges. Our measurement instrument for intramodule coupling is information theory, the same as for ordinary coupling.

We define intramodule coupling of a module in the same way as ordinary coupling, but we substitute the intramodule-edges graph, S°, for the intermodule-edges graph, S^*, in Definition 6.

Definition 7 (Intramodule coupling of a module)
Given a modular system, MS, and its intramodule-edges graph, S°, and a module, m_k, the intramodule coupling of the module is given by

$$IntramoduleCoupling(m_k|MS) = \sum_{i \in m_k} I(S_i^\circ)$$

Like ordinary coupling, the unit of measure is a bit,

Theorem 2 (Intramodule coupling of a module)
$IntramoduleCoupling(m_k|MS)$ has the properties specified in Table 4 and thus, is a member of family of measures called "intramodule coupling of a module".

Chen [12] gives a proof.

7. Cohesion of a module

Intramodule coupling and cohesion are measures of different attributes of relationships within modules; intramodule coupling is an amount of information with no upper limit in principle, but according to Table 2, cohesion is normalized between zero and a maximum.

Note that a *complete graph* has all possible edges among its nodes. A module that is a complete graph is the most cohesive possible. Before we define cohesion of a module, consider the following lemma.

Lemma 1 (Complete module)
Suppose module $m_k^{(n_k)}$ in S° is a complete graph, and the number of nodes in m_k and S° are n_k and n respectively. When $n_k = 1$,

$$IntramoduleCoupling(m_k^{(n_k)}|MS) = 0$$

When $n_k = 2$,

$$IntramoduleCoupling(m_k^{(n_k)}|MS) = n_k(n+1)\log(n+1) - n_k(n+1-n_k)\log(n+1-n_k) - n_k^2$$

When $n_k > 2$,

$$IntramoduleCoupling(m_k^{(n_k)}|MS) = n_k(n+1)\log(n+1) - n_k(n+1-n_k)\log(n+1-n_k)$$

Chen [12] gives a proof. This lemma shows that the intramodule coupling of a module that is a complete graph is a straightforward function of the numbers of nodes, n_k and n.

Cohesion is applicable to the same software engineering abstractions as intramodule coupling. Our measurement protocol for cohesion is the same as for intramodule coupling. Our measurement instrument for cohesion is similar to the instrument for coupling, based on information theory, because we define cohesion of a module in terms of intramodule coupling of a module.

Definition 8 (Cohesion of a module)
Given a module m_k with n_k nodes, in a modular system MS, the module's cohesion is given by

$$Cohesion(m_k|MS) = \frac{IntramoduleCoupling(m_k|MS)}{IntramoduleCoupling(m_k^{(n_k)}|MS)}$$

for $n_k > 1$. By convention, $Cohesion(m_k|MS) = 0$ when $n_k = 1$.

Cohesion of a module is a dimensionless ratio of bits. It has a minimum of zero and a maximum of one. Definition 7 gives the numerator, and Lemma 1 gives formulas for the denominator.

Like coupling, Briand, Daly, and Wüst [7] found that many metrics claim to measure some aspect of "cohesion", but some do not fulfill all the properties specified in Table 2. We prefer to reserve the name *cohesion of a module* for those measures that have all the specified properties.

Theorem 3 (Cohesion of a module)
$Cohesion(m_k|MS)$ has the properties specified in Table 2, and thus, is a member of the family of measures called "cohesion of a module".

Table 5. System profile

Name	Nethack	
Application	Adventure game	
User interface	X-Windows	
Language	ANSI C	
Operating System	UNIX	
	Baseline	Current
Files	107	111
Functions	1685	2008

Table 6. Summary statistics

	Intermodule Coupling		Cohesion	
	Baseline	Current	Baseline	Current
Files	107	111	107	111
Average	3673	5183	0.183	0.198
Std Dev	4490	5967	0.155	0.117
Maximum	33113	43746	1.000	1.000
Median	2332	3361	0.145	0.162
Minimum	0	0	0.000	0.000

Chen [12] gives a proof.

8. Empirical case study

8.1. System description

The case study examined the computer game, Nethack, version 3.2.1.[1] Nethack is an adventure game inspired by the games *Dungeons and Dragons*® and *rogue*. We analyzed a baseline version and a current version [12]. Table 5 gives a profile of the program.

A *call graph* has nodes that represent C functions and edges that represent the potential for one or more calls between a pair of functions. Jordan developed a research tool which derives a call graph at the function level from ANSI C source code [18]. Using this tool, we derived a call graph for each of the two versions. We considered a source file to be a "module". Most source files contain multiple functions.

8.2. Results

We calculated the coupling and cohesion of each source file based on the call graph. Table 6 [12] gives summary statistics over each version's set of files. An inspection of the detailed results revealed that the *call-graph coupling* of almost all source files increased from the baseline to the current version [12]. *Call-graph cohesion* of the source files varied, but Table 6 shows that call-graph cohesion of source files generally increased also. Common software engineering wisdom would welcome the increase in cohesion, but the increase in coupling was probably making the program more difficult to maintain.

Table 6 indicates a skewed distribution for coupling: the average coupling is higher than the median, and the standard deviation is larger than the average. This was due to a few source files that were highly coupled to other source files, such as those containing primitive utility functions. Table 6 also indicates that the distribution of cohesion values was somewhat skewed toward the low end as well.

The number of intermodule edges of a module, $IntermoduleEdges(m_k)$, is a simple measure that conforms to Briand, Morasca, and Basili's definition of coupling of a module [9]. Similarly, the ratio of intramodule edges of a module, $IntramoduleEdges(m_k)$, to the number of edges in a complete module-graph, $IntramoduleEdges(m_k^{(n_k)})$, is a simple measure that conforms to their definition of cohesion of a module [9].

$$IntramoduleRatio(m_k) = \frac{IntramoduleEdges(m_k)}{IntramoduleEdges(m_k^{(n_k)})} \quad (8)$$

When we compared information theory-based call-graph coupling and cohesion to these counting-based measures, we found that $Coupling(m_k|MS)$ and $Cohesion(m_k|MS)$ were highly correlated with $IntermoduleEdges(m_k)$ and $IntramoduleRatio(m_k)$, respectively, over this set of source files. However, the information theory-based measures made finer-grain distinctions. When one is interested in exploiting overall variation in call-graphs, this case study indicated that counting measures may be adequate. However, discovery of rare module attributes might require exploiting the finer-grain distinctions offered by the information theory-based metrics in conjunction with the coarser count-based metrics. This modeling issue is a topic for future research.

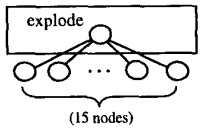
To illustrate the difference between counting and information theory-based metrics, Figure 4 depicts intermodule-edges subgraphs for several Nethack source files that have approximately the same number of intermodule edges. (Only nodes relevant to the source files are shown, and the drawing does not imply the directions of edges.)

Note that coupling has different values for modules with approximately the same number of intermodule

[1] Nethack is available from `http://www.win.tue.nl/games/roguelike/nethack`.

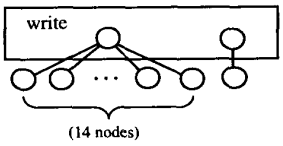
a.

IntermoduleEdges(explode) = 15 edges
Coupling(explode|*MS*) = 529.3 bits



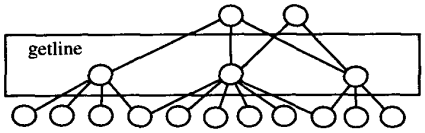
b.

IntermoduleEdges(write) = 15 edges
Coupling(write|*MS*) = 539.5 bits



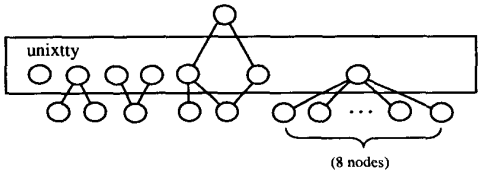
c.

IntermoduleEdges(getline) = 18 edges
Coupling(getline|*MS*) = 614.3 bits



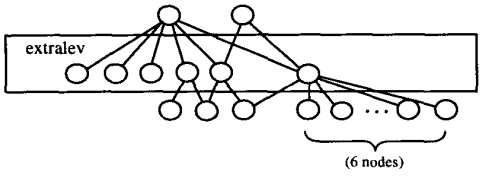
d.

IntermoduleEdges(unixtty) = 17 edges
Coupling(unixtty|*MS*) = 630.6 bits



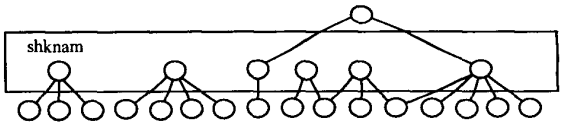
e.

IntermoduleEdges(extralev) = 19 edges
Coupling(extralev|*MS*) = 648.8 bits



f.

IntermoduleEdges(shknam) = 20 edges
Coupling(shknam|*MS*) = 738.1 bits



g.

IntermoduleEdges(o_init) = 19 edges
Coupling(o_init|*MS*) = 742.2 bits

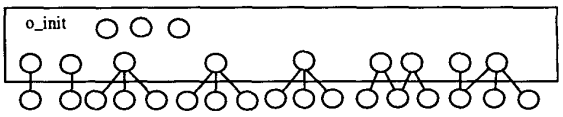


Figure 4. Examples of call-graph coupling

edges. Subgraphs a. and b. each have 15 edges, but different coupling. Similarly, subgraphs e. and g. each have 19 edges, but different coupling. Also compare subgraphs c. *vs.* d. and f. *vs.* g. These examples illustrate how coupling distinguishes patterns of connections in a way that counting cannot.

We saw similar fine-grain distinctions made by $Cohesion(m_k|MS)$ compared to $IntramoduleRatio(m_k)$. However, space does not permit presentation here.

9. Conclusions

Software architects and designers use many kinds of graphs to depict abstractions of software. For example, call graphs have been used by software developers for many years to depict potential calling relationships. The coupling and cohesion of modules in such a graph are just two attributes among many that may be useful for software quality modeling.

The information-theory approach to software metrics differs from counting in the way patterns in a graph can have distinct measurements. We model the pattern of edges incident to each node as a random variable. The distribution of that random variable is the basis for calculating the amount of information. In software engineering terms, we are modeling both the design decisions to connect each pair of nodes, and the decisions not to connect the others. The benefit of this approach is a set of measures that are sensitive to patterns of connections. This is attractive, because software engineers recognize patterns as well.

In this paper, we define metrics that conform to the properties proposed by Briand, Morasca, and Basili [9], and therefore, we call them (intermodule) *coupling of a module*, *intramodule coupling of a module*, and *cohesion of a module*. The definitions are based on information theory, taking patterns of relationships into account, rather than just counts of graph features.

These measures may be applied to any graph, and consequently, are suitable for many software engineering abstractions, and thus, for many kinds of coupling and cohesion. The practical interpretation of each measure depends on the underlying abstraction. Accordingly, the usefulness of each measure depends on the relevance of the abstraction to the development process.

Our empirical study examined a moderate-size computer game, calculating coupling and cohesion of a function call-graph where a source file was considered a "module". We found that the overall variation of these metrics was similar to that of corresponding count-based metrics, i.e., they were highly correlated. However, the information theory-based metrics made finer grain distinctions.

Future research will further compare counting-based and information theory-based metrics. Future work will also investigate coupling and cohesion of abstractions of software other than call graphs, which may well have radically different statistical distributions. Future research will validate the usefulness of coupling and cohesion of a module in the context of models that predict software quality [29]. Future research will also apply information theory to other families of metrics such as size, length, and complexity [9].

Acknowledgments

We thank Sylviane Jordan for developing the call graph tool. We also thank the anonymous reviewers for their thoughtful comments. This work was supported in part by the National Science Foundation grant CCR-9803853.

References

[1] E. B. Allen. *Information Theory and Software Measurement.* PhD thesis, Florida Atlantic University, Boca Raton, FL, Aug. 1995. Advised by Taghi M. Khoshgoftaar.

[2] E. B. Allen and T. M. Khoshgoftaar. Measuring coupling and cohesion: An information-theory approach. In *Proceedings of the Sixth International Software Metrics Symposium*, pages 119–127, Boca Raton, Florida USA, Nov. 1999. IEEE Computer Society.

[3] J. Bansiya, C. G. Davis, and L. Etzkorn. An entropy basd complexity measure for object-oriented designs. *Theory and Practice of Object Systems*, 5(2):1–9, May 1999.

[4] V. R. Basili, L. C. Briand, and W. Melo. A validation of object-oriented design metrics as quality indicators. *IEEE Transactions on Software Engineering*, 22(10):751–761, Oct. 1996.

[5] J. M. Bieman and B.-K. Kang. Measuring design-level cohesion. *IEEE Transactions on Software Engineering*, 24(2):111–124, Feb. 1998.

[6] J. M. Bieman and L. M. Ott. Measuring functional cohesion. *IEEE Transactions on Software Engineering*, 20(8):644–657, Aug. 1994.

[7] L. C. Briand, J. W. Daly, and J. Wüst. A unified framework for cohesion measurement in object-oriented systems. In *Proceedings of the Fourth International Symposium on Software Metrics*, pages 43–53, Albuquerque, NM USA, Nov. 1997. IEEE Computer Society.

[8] L. C. Briand, J. W. Daly, and J. K. Wüst. A unified framework for coupling measurement in object-oriented systems. *IEEE Transactions on Software Engineering*, 25(1):91–121, Jan. 1999.

[9] L. C. Briand, S. Morasca, and V. R. Basili. Property-based software engineering measurement. *IEEE Transactions on Software Engineering*, 22(1):68–85, Jan. 1996. See comments in [10, 27, 35].

[10] L. C. Briand, S. Morasca, and V. R. Basili. Response to: Comments on "Property-based software engineering measurement": Refining the additivity properties. *IEEE Transactions on Software Engineering*, 23(3):196–197, Mar. 1997. See [9, 27].

[11] H. S. Chae and Y. R. Kwon. A cohesion measure for classes in object-oriented systems. In *Proceedings Fifth International Software Metrics Symposium*, pages 158–166, Bethesda, MD USA, Nov. 1998. IEEE Computer Society.

[12] Y. Chen. Measurement of coupling and cohesion of software. Master's thesis, Florida Atlantic University, Boca Raton, Florida USA, May 2000. Advised by Taghi M. Khoshgoftaar.

[13] T. M. Cover and J. A. Thomas. *Elements of Information Theory*. John Wiley & Sons, New York, 1991.

[14] J. S. Davis and R. J. LeBlanc. A study of the applicability of complexity measures. *IEEE Transactions on Software Engineering*, 14(9):1366–1372, Sept. 1988.

[15] R. Harrison, S. J. Counsell, and R. V. Nithi. Coupling metrics for object-oriented design. In *Proceedings Fifth International Software Metrics Symposium*, pages 150–157, Bethesda, MD USA, Nov. 1998. IEEE Computer Society.

[16] R. Harrison, S. J. Counsell, and R. V. Nithi. An evaluation of the MOOD set of object-oriented software metrics. *IEEE Transactions on Software Engineering*, 24(6):491–496, June 1998.

[17] L. Hatton. Reexamining the fault density — component size connection. *IEEE Software*, 14(2):89–97, Mar. 1997. Corrected Figure 2 is available through http://www.computer.org/software/so1997/s2toc.htm.

[18] S. Jordan. Software metrics collection: Two new research tools. Master's thesis, Florida Atlantic University, Boca Raton, FL, Dec. 1997. Advised by Taghi M. Khoshgoftaar.

[19] T. M. Khoshgoftaar and E. B. Allen. Applications of information theory to software engineering measurement. *Software Quality Journal*, 3(2):79–103, June 1994.

[20] B. A. Kitchenham, S. L. Pfleeger, and N. E. Fenton. Towards a framework for software measurement validation. *IEEE Transactions on Software Engineering*, 21(12):929–944, Dec. 1995. See comments in [21, 25].

[21] B. A. Kitchenham, S. L. Pfleeger, and N. E. Fenton. Reply to: Comments on "Towards a framework for software measurement validation". *IEEE Transactions on Software Engineering*, 23(3):189, Mar. 1997. See [20, 25, 34].

[22] K. S. Lew, T. S. Dillon, and K. E. Forward. Software complexity and its impact on software reliability. *IEEE Transactions on Software Engineering*, 14(11):1645–1655, Nov. 1988.

[23] G. A. Miller. The magical number 7 plus or minus two: Some limits on our capacity for processing information. *Psychological Review*, 63:81–97, 1957.

[24] S. Morasca and L. C. Briand. Towards a theoretical framework for measuring software attributes. In *Proceedings of the Fourth International Symposium on Software Metrics*, pages 119–126, Albuquerque, NM USA, Nov. 1997. IEEE Computer Society.

[25] 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 Transactions on Software Engineering*, 23(3):187–188, Mar. 1997. See [20, 34].

[26] D. L. Parnas. On the criteria to be used in decomposing systems. *Communications of the ACM*, 15(12):1053–1058, Dec. 1972.

[27] G. Poels and G. Dedene. Comments on "Property-based software engineering measurement": Refining the additivity properties. *IEEE Transactions on Software Engineering*, 23(3):190–195, Mar. 1997. See [9].

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

[29] N. F. Schneidewind. Methodology for validating software metrics. *IEEE Transactions on Software Engineering*, 18(5):410–422, May 1992.

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

[31] M. Shaw, R. DeLine, D. V. Klein, T. L. Ross, D. M. Young, and G. Zelesnik. Abstractions for software architecture and tools to support them. *IEEE Transactions on Software Engineering*, 21(4):314–335, Apr. 1995.

[32] M. H. van Emden. Hierarchical decomposition of complexity. *Machine Intelligence*, 5:361–380, 1970. See also [33] for details.

[33] M. H. van Emden. *An Analysis of Complexity*. Number 35 in Mathematical Centre Tracts. Mathematisch Centrum, Amsterdam, 1971.

[34] E. J. Weyuker. Evaluating software complexity measures. *IEEE Transactions on Software Engineering*, 14(9):1357–1365, Sept. 1988.

[35] H. Zuse. Reply to: "Property-based software engineering measurement". *IEEE Transactions on Software Engineering*, 23(8):533, Aug. 1997. See [9].