Software Similarity-Based Functional Cohesion Metric

Jehad Al Dallal

Department of Information Sciences
College for Women
Kuwait University
P.O. Box 5969, Safat 13060, Kuwait
jehad@cfw.kuniv.edu

Abstract

Cohesion is an important factor used in evaluating software design quality and modularity. The cohesion of a module refers to the relatedness of the module components. In software engineering, highly cohesive modules are highly desirable due to their high reusability and maintainability. Cohesion is classified according to levels. Functional cohesion, the strongest level, refers to how closely the module parts that contribute to different outputs are related.

In this paper, a Similarity-Based Functional Cohesion (SBFC) metric is introduced to measure the functional cohesion of a module in a procedural or object-oriented program. The metric uses the degree of similarity between module data slices as a basis for measuring functional cohesion. The appropriateness of the metric is evaluated both theoretically and empirically. The evaluation results show that the metric does as well as some earlier metrics in indicating the cohesiveness level, and it does better than some in terms of providing different values for the modules of different cohesion. In addition, the SBFC metric is used as an indicator for restructuring the weakly cohesive modules.

Keywords
Functional cohesion, module cohesion metric, static program slicing, data slicing.

1. Introduction

Popular goals of software engineering are to develop and use techniques and tools for creating high quality applications. Applications that have high quality and modularity are more stable and maintainable. Cohesion is a major attribute of software modules and is used in software engineering as an important software design quality factor. Module cohesiveness indicates how closely related are the components of a module. A highly cohesive module performs one basic function and cannot be easily split into separate modules. In this paper, we are concerned with measuring the cohesion of basic program modules (i.e., functions or procedures in procedural programs and methods in object-oriented programs).

Yourdon et al. [1] proposed seven levels of module cohesiveness. All of these levels indicate how much the individual module components contribute towards performing one task. The level of cohesion indicates to what extent the relatedness of the module components contribute to different module outputs. When considering module cohesion, the module is viewed as a set of processing components used to compute or produce the module outputs. The module that has a single processing component or is composed of highly related components has high cohesion. In contrast, the module that has two or more independent components has low cohesion. As a result, module cohesion increases as the dependency between the module components increases, and vice versa. Functional cohesion is the most desirable level of cohesion. A highly functional cohesive module is generally reusable and easily maintainable.
Weiser [2] introduced the program slicing concept. Program slicing is the task of finding all statements in a program that directly or indirectly influence the value of a variable occurrence. The set of statements that may affect the value of a variable at some point in a program is called a program slice. Weiser also presented several slice-based metrics but did not identify a meaningful use for them in measuring actual software attributes. Longworth [3] suggested using slice-based metrics as indicators of cohesion. Ott et al. [4] introduced the application of slice-based metrics for slices computed at each output variable of the module under consideration. This is because output variables identify the intended tasks of the module.

Since functional cohesion refers to the degree to which the parts of a module that contribute to different outputs are related, Bieman and Ott [5] introduced the use of data slices computed at output variables to measure the module’s functional cohesion. Bieman and Ott’s data-slicing concept maps the program’s slices onto data definitions and uses in the module. Several metrics have been introduced to measure the cohesion between the methods in a class, including the lack of cohesion of methods (LCOM) [6], cohesion among methods in a class (CAMC) [7], normalized Hamming distance (NHD) [8], scaled NHD (SNHD) [9], Class Cohesion (CC) [10], and Sensitive Class Cohesion Metric (SCOM) [11].

In this paper, a new metric, called Similarity-Based Functional Cohesion (SBFC), is introduced to measure the functional cohesion of a module. This metric can be used to measure the functional cohesion of the functions and procedures in procedural programs as well as of individual methods in object-oriented programs. The SBFC metric measures cohesion similarly to how the earlier Bieman and Ott metrics measured cohesion. However, it requires three Bieman and Ott cohesion metrics to fully measure cohesion, and there are cases in which the three Bieman and Ott cohesion metrics disagree about the cohesion of individual modules. In addition, sometimes individual Bieman and Ott metrics give the same value for two modules of demonstrably different cohesion. The SBFC metric provides a single measure of cohesion, and so no disagreement is possible. In addition, it provides different values for the modules of different cohesion, and so it solves the previous problems with the Bieman and Ott metrics. An empirical study has shown high correlations with SBFC and the Bieman and Ott metrics. Therefore, the SBFC metric does as well overall as the Bieman and Ott metrics for measuring cohesion, but it does not have the problems the Bieman and Ott metrics have. Similarly to the Bieman and Ott metrics, the SBFC metric has been demonstrated to be on an ordinal scale (i.e., a measurement scale based on ranking objects with respect to one another). The SBFC metric satisfies the necessary properties introduced by Briand et al. [12] for any module cohesion metric, and it is useful as an indicator for restructuring weakly cohesive modules.

The paper is organized as follows. Section 2 provides an overview of related work in this field. In Section 3, the SBFC metric is introduced. In Section 4 the proposed metric is theoretically evaluated. An application for the technique is illustrated in Section 5, and in Section 6, a comprehensive correlation study with the related techniques is conducted. Finally, Section 7 provides conclusions and a discussion of future work.

2. Related Work

The SBFC measures the similarity between pairs of data slices. The data slices relate the program slices to the definitions and uses of the program data. This section provides an overview of the concepts of program slicing and data slicing as well as the module-cohesion necessary properties introduced by Briand et al. [12], the Bieman and Ott functional cohesion metrics [5], the similarity-based cohesion metrics, and other related work in the field.

2.1. Program slicing
Program slicing is the task of finding all the statements in a program that directly or indirectly influence the value of a variable occurrence. The set of statements that may affect the value of a variable at some point in a program is called a program slice. Program slicing can be static or dynamic. In static program slicing, it is necessary to find a program slice that involves all statements that may affect the value of a variable at a program point for any input set. In dynamic program slicing, the slice is found with respect to a given input set. Several algorithms have been introduced to find static and dynamic slices. These algorithms compute the slices automatically by analyzing the program data flow and control flow. Computing the slices of a given procedure is called intra-procedural slicing. Computing the slices of a multi-procedural program is called inter-procedural slicing.

Since cohesion is not limited to a set of inputs and the paper deals with modules individually, this paper is concerned with intra-procedural static slicing. The basic algorithms for computing static intra-procedural slices follow three main approaches. The first uses data flow equations (e.g., [2], [13]), the second uses information-flow relations (e.g., [14]), and the third uses program dependence graphs (e.g., [15]). Dependency graph-based slicing algorithms are in general more efficient than the algorithms that use data flow equations or information-flow relations [16].

The Program Dependence Graph (PDG) consists of nodes and direct edges. Each program’s simple statements and control predicates are represented by nodes. Simple statements include assignment, read, and write statements. Compound statements include conditional and loop statements and are represented by more than one node. There are two types of edges in a PDG: data dependence edges and control dependence edges. A data dependence edge between two nodes implies that the computation performed at the sink node pointed to by the edge depends directly on the value computed at the source node. This means that the sink node has the definition of the variable used in the source node. A control dependence edge between two nodes implies that the result of the predicate expression at the sink node pointed to by the edge is the factor for deciding whether to execute the source node. Figure 1 shows a C function example. The function computes the sum, average, and product of numbers from 1 to n, where n is an integer value greater than or equal to 1. Figure 2 shows the PDG of the C function example given in Figure 1. The numbers associated with the nodes of the PDG indicate the line numbers of the statements represented by the nodes. Solid and dotted direct edges represent the control and data dependency edges, respectively.

```c
1    void NumberAttributes(int n, int &sum, double &avg, int &product) {
2        int i=1;
3        sum=0;
4        product=1;
5        while (i<=n) {
6            sum=sum+i;
7            product=product*i;
8            i=i+1;
9        }
10        avg=static_cast<double>(sum)/n;
11    }
```

Figure 1: C function example
Depending on the slicing purpose, slicing can be backward or forward. In backward slicing, finding the set of statements that may affect the value of a variable at some point in a program is required. This can be obtained by walking backwards over the PDG to find all the nodes that have an effect on the value of a variable at the point of interest. In forward slicing, finding the set of statements that may be affected by the value of a variable at some point in a program is required. This can be obtained by walking forward over the PDG to find all the nodes that may be affected by the value of a variable. In this paper, we are interested in backward slicing. Using the PDG shown in Figure 2, we can obtain the backward slices shown in Table 1.

<table>
<thead>
<tr>
<th>Slice attributes</th>
<th>Slice contents (line numbers)</th>
</tr>
</thead>
<tbody>
<tr>
<td>sum</td>
<td>1,2,3,5,6,8</td>
</tr>
<tr>
<td>avg</td>
<td>1,2,3,5,6,8,10</td>
</tr>
<tr>
<td>product</td>
<td>1,2,4,5,7,8</td>
</tr>
</tbody>
</table>

Table 1: Slice examples

2.2. Data slicing

Functional cohesion is defined as the relatedness of the module parts that contribute to different outputs. An output in a procedural program can be a single value output to a file or a device, an assignment to a global variable, a returned value, or an output parameter. An output in an object-oriented program can be a single value output to a file or a device, an assignment to a class attribute, a returned value, or an assignment to an object parameter. Bieman and Ott [5] argued that using a data token (i.e., variable and constant definitions and references) as the basis for slicing ensures that all changes of interest will cause a change in at least one slice of the module. A change of interest is any change that may affect the cohesiveness of the module. Adding code, deleting code, and changing a variable used in a given context are examples of changes of interest. Changing an operator is an example of a change that is not of interest.

Data slicing is performed by mapping a slice to the data tokens included in the slice. A data slice is a sequence of data tokens included in a slice. For example, the data slice for the variable sum at line
number 6 of the function given in Figure 1 is a sequence of the data tokens: \( n_1, sum_1, avg_1, product_1, i_1, l_1, sum_2, 0_1, i_2, n_2, sum_3, sum_4, i_3, i_5, i_6, I_3 \), where \( T_i \) indicates the \( i \)'th data token for \( T \) in the function. Table 2 shows the data slice abstraction for the slices given in Table 1.

<table>
<thead>
<tr>
<th>Data tokens</th>
<th>sum</th>
<th>avg</th>
<th>product</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n_1 )</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>( sum_1 )</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>( avg_1 )</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>( product_1 )</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>( i_1 )</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>( l_1 )</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>( sum_2 )</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>( 0_1 )</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>( product_2 )</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( l_2 )</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( i_2 )</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>( n_2 )</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>( sum_3 )</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>( sum_4 )</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>( i_3 )</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>( product_3 )</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( product_4 )</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( i_4 )</td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>( i_5 )</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>( i_6 )</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>( l_3 )</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>( avg_2 )</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( sum_5 )</td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>( i_3 )</td>
<td></td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Data slice abstraction for the slices given in Table 1

### 2.3. Cohesion metric properties

Briand et al. [12] defined four properties for cohesion metrics. The first property, called non-negativity and normalization, states that the cohesion measure belongs to a specific interval \([0, \text{Max}]\). Normalization allows for easy comparison between the cohesion of different modules. The second property, called null value and maximum value, states that the cohesion of a module equals zero if the module has no cohesive interactions (i.e., interactions that relate module components together) and that the cohesion is equal to \( \text{Max} \) if all possible interactions within the module are present. The third property, called monotonicity, states that adding cohesive interactions to the module cannot decrease its cohesion. The fourth property, called cohesive modules, states that merging two unrelated modules into one does not increase the module’s cohesion. Therefore, given two modules, \( M_1 \) and \( M_2 \), the cohesion of the merged module \( M' \) has to satisfy the condition: \( \text{cohesion}(M') \leq \max\{\text{cohesion}(M_1), \text{cohesion}(M_2)\} \). The cohesion metric has to satisfy all the above properties; otherwise, its use as a cohesion indicator is questionable.

### 2.4. Bieman and Ott’s functional cohesion metric
Bieman and Ott [5] introduced the concept of data slicing and applied it as an abstraction for measuring module functional cohesion. Data slices are found for each output of the module, as illustrated in Section 2. For example, the C function given in Figure 1 has three outputs: sum, average, and product. The data slices for these outputs are given in Table 2.

Bieman and Ott defined three terms: glue token, super-glue token, and glue stickiness. A glue token is a data token that exists in more than one data slice. A super-glue token is a data token that exists in all data slices. The stickiness or adhesiveness of a glue token is the number of data slices that it binds. For example, Table 2 shows that the data token \( n_1 \) is a super-glue token because it exists in all three slices, and it is a glue token because it exists in more than one slice. The data token \( \text{sum}_2 \) is a glue token because it exists in two slices, but it is not a super-glue token because it does not exist in all three slices. The adhesiveness of the super-glue token for \( n_1 \) is 3, and the adhesiveness of the glue token for \( \text{sum}_2 \) is 2.

Three metrics are introduced for a module, including Strong Functional Cohesion (SFC), Weak Functional Cohesion (WFC), and Adhesiveness (A). SFC is the ratio of the number of super-glue tokens to the total number of data tokens in the module. WFC is the ratio of the number of glue tokens to the total number of data tokens in the module. Finally, the adhesiveness of the module is the ratio of the total adhesiveness of all glue tokens to the total possible adhesiveness (i.e., when each data token is used by each data slice). All three of the measures lie on the interval between zero and one. The number of super-glue tokens and glue tokens shown in Table 2 is 11 and 16, respectively. The total number of data tokens is 24. As a result, the SFC of the function given in Figure 1 is \( \frac{11}{24} = 0.46 \), and the WFC of the function is \( \frac{16}{24} = 0.67 \). Finally, the adhesiveness of the function is \( \frac{(11*3+2*5)}{(3*24)} = 0.60 \).

Having three metrics for the cohesion creates two difficulties. The first one is the need for an analysis of the measures to have an idea or to give a decision about the functional cohesion of the module. For example, for a module that has SFC=0, WFC=1, and A=0.67, it is not easy to say whether the module has a high or low cohesion. The second difficulty in having three measures is in comparing the cohesion of two different modules or two implementations of the same module. It is clear that the module that has the cohesion measure values (SFC=0.29, WFC=1, A=0.68) has a higher cohesion than the module that has the cohesion measure values (SFC=0.14, WFC=1, A=0.64). However, it is difficult to compare the cohesion of the module that has the cohesion measure values (SFC=0.25, WFC=0.75, A=0.58) with the cohesion of either of the previous two modules. This demonstrates the need for a single-based measure for the module functional cohesion.

2.5. Similarity-based cohesion metrics

The Class Cohesion (CC) metric [10] uses the degree of similarity between methods as a basis for measuring the class cohesion. The similarity between a pair of methods is defined as the ratio of the number of shared attributes to the number of distinct attributes referenced by both methods. The cohesion is defined as the ratio of the summation of the similarities between all pairs of methods to the total number of possible pairs of methods. The Sensitive Class Cohesion Metric (SCOM) [11] is similar to the CC metric. The only difference is in the definition of similarity. Given a class that has \( I \) attributes, the similarity between a pair of methods \( M_i \) and \( M_j \), which reference the set of attributes \( I_i \) and \( I_j \) respectively, is defined formally as follows:

\[
\text{Similarity}(M_i, M_j) = \frac{|I_i \cap I_j|}{\min(|I_i|, |I_j|)} \cdot \frac{|I_i \cup I_j|}{I}.
\]

Both the CC and SCOM metrics do not satisfy the monotonicity property, in some cases due to their similarity definitions.
2.6. Other cohesion-related work

In 1979, Yourdon et al. [1] proposed seven levels of cohesion: coincidental, logical, temporal, procedural, communicational, sequential, and functional. The cohesion levels are listed in ascending order according to their desirability. Emerson [17] used the control flow graph representation of a module as the basis for measuring cohesion. He proposed three levels of cohesion: data cohesion, control cohesion, and superficial cohesion. Lakhotia [18] introduced the variable dependence graph that abstracts the control and data dependencies between module variables. Rules are used to map the dependences of the variables to the levels of cohesion introduced by Yourdon. The cohesion of the module is undefined if the module has no output variables. The cohesion is otherwise defined to be the lowest cohesion of all pairs of the output variables of the module. The cohesion is functional if the model has only one output variable. Ott et al. [4] introduced the use of the slicing metrics proposed by Weizer [2] for estimating cohesion. The slicing metrics are overlap, tightness, and parallelism. In [19] and [20], cohesion is measured indirectly by examining the quality of the structured design.

For object-oriented programs, most cohesion measurement approaches consider the interactions between methods and instance variables (i.e., class cohesion). A class is cohesive if the same attributes appear in most or all of the methods in a class. Metrics introduced in [6–12] and [21–27] measure the class cohesion of object-oriented programs.

3. Similarity-Based Functional Cohesion (SBFC) Metric

In this paper, we rely on Bieman and Ott’s work [5], in which the data slices of the module outputs are used as abstractions for measuring the functional cohesion. The relationship between the data slices of the module outputs is measured in terms of the shared data tokens. In this paper, the same model abstraction (i.e., data slices) is used. Instead of using all data slices at once, as in [5], each pair of data slices is considered individually. The SBFC metric introduced in this paper measures the relationship between each pair of data slices. This causes the metric to be more precise and sensitive to the changes in the module’s cohesive interactions, as discussed later in Section 4.3. Functional cohesion is viewed as the average of these relations over all pairs of data slices.

3.1. Definition of measures

The data slice abstraction can be modeled in a binary matrix $m$ of size $t \times s$, where $t$ is the number of data tokens and $s$ is the number of data slices for the outputs of a module of interest. The matrix has rows indexed by the data tokens and columns indexed by the module outputs, and for $1 \leq i \leq t$, $1 \leq j \leq s$,

$$m_{ij} = \begin{cases} 1 & \text{if } i\text{th data token belongs to the data slice of } j\text{th output,} \\ 0 & \text{otherwise} \end{cases}$$

The matrix models the interactions between the data slices of the module outputs. A data slice of an output has a cohesive interaction with a data slice of another output if they share a common data token. Therefore, a cohesive interaction is represented in the matrix by two columns sharing binary values 1 in a row. The similarity between a pair of data slices is equal to the number of module data tokens.

\[ns(i, j) = \frac{\sum_{x \in \mathcal{S}} (m_{ix} \land m_{jx})}{t},\]

(1)
where \( \land \) is the logical and relation (i.e., equals 1 if the values of the two operands are equal to 1). Cohesion refers to the degree of similarity between module components. The overall functional cohesion of a module \( M \), denoted \( SBFC \), equals 1 if the module has only one data slice; otherwise, \( SBFC \) equals the average of the functional cohesion of all pairs of module data slices and is defined formally as follows:

\[
SBFC (M) = \begin{cases} 
1 & \text{if } s = 1, \\
\frac{2}{s(s-1)} \sum_{i=1}^{s-1} \sum_{j=i+1}^{s} ns(i, j) & \text{otherwise.} 
\end{cases}
\]  

By substituting Formula 1 into Formula 2, the \( SBFC \) of module \( M \) is calculated as follows:

\[
SBFC(M) = \begin{cases} 
1 & \text{if } s = 1, \\
\frac{2}{s(s-1)} \sum_{i=1}^{s-1} \sum_{j=i+1}^{s} (m_{iw} \land m_{jw}) & \text{otherwise.} 
\end{cases}
\]  

The following metric is an alternative form of the \( SBFC \) metric, which facilitates the analysis of the metric and speeds up its computation:

\[
SBFC(M) = \begin{cases} 
\sum_{i=1}^{s} x_i (x_i - 1) & \text{if } s = 1, \\
\frac{x_i}{t.s(s-1)} & \text{otherwise.} 
\end{cases}
\]  

where \( x_i \) is the number of 1s in the \( i \)th row of the matrix.

**Proof:** By definition, when \( s = 1 \), Formula 3 and 4 are equal. Otherwise, for the \( i \)th row, there are \( x_i(x_i - 1)/2 \) similarities between the data slices, and therefore,

\[
SBFC(M) = \frac{2}{t.s(s-1)} \sum_{i=1}^{s-1} \sum_{j=i+1}^{s} (m_{iw} \land m_{jw}) = \frac{2}{t.s(s-1)} \sum_{i=1}^{s} x_i (x_i - 1) 2, 
\]

which equals to Formula 4. \( \Box \)

For example, the data slice abstraction given in Table 2 has eleven rows containing data tokens shared between three data slices, five rows containing data tokens shared between two data slices, and eight rows containing non-shared data tokens. As a result, by applying Formula 4, the functional cohesion of the function given in Figure 1 is:

\[
SBFC(Number\text{Attributes}) = \frac{(11)(3)(2) + (5)(2)(1) + 8(1)(0)}{24(3)(3-1)} = 0.53 .
\]

The formulation of the SBFC metric has some similarities to the formulations of the CC [10] and SCOM [11] metrics, although they are based on different kinds of data. However, formulations of the CC and SCOM metrics lead to a violation of the monotonicity property in some cases, whereas the SBFC metric is free from this violation of monotonicity property because of its particular formulation.

### 3.2. Module splitting threshold

Based on the above metric definition, a threshold can be defined and used as a support for deciding whether a module must be kept as is or split into several modules. A module should be split into several modules if the cohesion of the software consisting of the split modules is greater than the cohesion of the module before splitting. The cohesion of software consisting of several modules is defined as the weighted cohesion of the modules, where the weight of a module equals \( t \) if \( s \) equals one; and the weight of a module equals the denominator of Formula 4, otherwise. For simplicity, the case in
which we have to decide whether the module should be split into two modules is considered here. The same concept can be applied to a decision about splitting the module into more than two modules. For a module $M$, given that the nominator and denominator of Formula 4 are denoted by $N(M)$ and $D(M)$, respectively, module $M$ should be split into two modules $M_1$ and $M_2$ when

$$\frac{N(M) + N(M_1)}{D(M)} > \frac{N(M_1) + N(M_2) + C_{12}}{D(M)}.$$  

(5)

As discussed in the proof of Formula 4, $N(M)$ is twice the number of similarities between the output data slices of module $M$. Therefore, $C_{12}$ is twice the number of similarities between the output data slices of module $M_1$ and the output data slices of module $M_2$ and can be calculated as $2\sum_{i=1}^{t} x_i y_i$, where $t$ is the number of data tokens in module $M$ and $x_i$ and $y_i$ are the number of 1s in row $i$ of the matrix in the columns representing data slices of modules $M_1$ and $M_2$, respectively. By substituting Formula 4 in Inequality 5, the threshold on $C_{12}$ is defined as follows:

$$C_{12} < \frac{[D(M) - D(M_1) - D(M_2)](N(M_1) + N(M_2))}{D(M_1) + D(M_2)} = \frac{[t s (s-1) - t_1 s_1 (s_1 - 1) - t_2 s_2 (s_2 - 1)](\sum_{i=1}^{t} x_i (x_i - 1) + \sum_{i=1}^{t} y_i (y_i - 1))}{t_1 s_1 (s_1 - 1) + t_2 s_2 (s_2 - 1)}$$

(6)

where $t$, $t_1$, $t_2$, $s$, $s_1$, and $s_2$ are the number of data tokens and output data slices in modules $M$, $M_1$, and $M_2$, respectively.

For example, for the function given in Figure 1, the following is the application of the above inequality to decide whether to split the function in two, one containing the outputs sum and avg and the other containing the output product. In this case, refer to the data slice abstraction given in Table 2, $x_1 y_1=2*1=2$, where $x_1$ is the number of 1s in the first row of the matrix in the columns representing data slices of the outputs sum and avg, and $y_1$ is the number of 1s in the first row of the matrix in the columns representing data slice of the output produce. Similarly, $x_i y_i$ can be calculated for each of the 24 rows in the matrix. As a result, $C_{12}=\sum_{i=1}^{24} x_i y_i = 2[2(1) + 2(1) + \ldots + 1(0)] = 2(22) = 44$. In addition, referring to the data given in Table 2, $t=24$, $t=19$ (i.e., number of data tokens included in the data slices of the module containing the outputs sum and avg), $t_2=16$, $s=3$, $s_1=2$, and $s_2=1$, and therefore, the inequality is applied as follows:

$$2(22) < \frac{[24(3)(3-1) - 19(2)(2 - 1) - 16][16(2)(1) + 16]}{19(2)(2 - 1) + 16}.$$  

Since the inequality is true, we conclude that the function should be split as specified above. In contrast, the following application of the inequality shows that the function should not be split into two functions: one containing the output sum and the other containing the outputs avg and product. In this
case, \( x_1 y_1 = 1 \times 2 = 2 \), where \( x_1 \) is the number of 1s in the first row of the matrix in the column representing data slice of the output \( sum \), and \( y_1 \) is the number of 1s in the first row of the matrix in the columns representing data slices of the outputs \( avg \) and \( produce \). Similarly, \( x_i y_i \) can be calculated for each of the 24 rows in the matrix. As a result, \( C_{12} = \sum_{i=1}^{24} x_i y_i = 2[(1(2)+1(2)+...+0(1)] = 2(27) = 54 \). In addition, referring to the data given in Table 2, \( t = 24, t_1 = 16, t_2 = 24, s = 3, s_1 = 1, \) and \( s_2 = 2, \) and therefore, the inequality is applied as follows:

\[
2(27) > \frac{[24(3-1) - 16 - 24(2)(2-1)][16 + 11(2)(1)]}{16 + 24(2)(2-1)},
\]

which indicates that the function should not be split as specified above.

4. Theoretical Evaluation

In this section, the usefulness of the SBFC metric as a cohesion measure is validated through discussion and proof of its satisfaction of the cohesion-measure necessary properties. In addition, the values and sensitivity of SBFC and Bieman and Ott's cohesion metrics are theoretically compared. Finally, it is shown that the SBFC is on an ordinal scale.

4.1. Cohesion metric property satisfaction

The SBFC metric is validated as follows, using the necessary properties for a module cohesion metric proposed by Briand et al. [12] and discussed in Section 2.

**Property SBFC.1: SBFC metric satisfies the non-negativity and normalization property**

**Proof:** The minimum value for the SBFC metric for a module is equal to 0 when none of the module output data slices share common data tokens. The maximum value for the SBFC metric for a module is equal to 1 when the module has one output or each of the output data slices includes all the module data tokens. As a result, the SBFC metric ranges in the interval \([0, 1]\) and therefore satisfies the non-negativity and normalization property. ■

**Property SBFC.2: SBFC metric satisfies the null and maximum values property**

**Proof:** Given a module with the set of outputs and data tokens, if none of the data tokens is shared between a pair of output data slices (i.e., the module does not have any cohesive interaction), the value of the SBFC metric will be zero. If, however, each data token is shared between each pair of output data slices, the value of the SBFC metric will be 1 (i.e., the maximum possible value). Hence, the SBFC metric satisfies null and maximum values property. ■

**Property SBFC.3: SBFC metric satisfies the monotonicity property**

**Proof:** The addition of a cohesive interaction to the matrix is represented by changing an entry value from 0 to 1 in a row that includes at least another entry value 1. Changing an entry value from 0 to 1 increases the number of 1s in a row. This results in increasing the numerator value in Formula 4. As a result, adding a cohesive interaction to the module always increases the SBFC value, which means that the SBFC metric satisfies the monotonicity property. ■

**Property SBFC.4: SBFC metric satisfies the cohesive modules property**

**Proof:** Merging two unrelated modules \( M_1 \) and \( M_2 \) means that none of the data tokens in each of the two split modules is shared between the sets of output data slices of each of the two split modules. Therefore, the number of rows and columns in the matrix of the merged module is equal to the summation of the number of rows and columns in the matrices of the split modules, respectively. In addition, the number of 1’s in each row or column in the matrix of the merged module is the same as the
number of 1’s in the corresponding row or column in the matrices of the split modules. Therefore, for a \( l \times k \) matrix representing module \( M_1 \), a \( n \times m \) matrix representing module \( M_2 \), and a \( (l+n) \times (k+m) \) matrix representing the merged module \( M_3 \):

\[
\sum_{i=1}^{l} x_i (x_i - 1) + \sum_{j=1}^{n} x_j (x_j - 1) = \sum_{i=1}^{l} x_i (x_i - 1) .
\]

Suppose that \( SBFC(M_1) \geq SBFC(M_2) \), then

\[
\frac{\sum x_i (x_i - 1)}{l(k+1)} \geq \frac{\sum x_i (x_i - 1)}{nm(m-1)} \Rightarrow nm(m-1) \sum x_i (x_i - 1) \geq l(k+1) \sum x_i (x_i - 1)
\]

\[
\Rightarrow [nm(m-1)+lk(k-1)] \sum x_i (x_i - 1) \geq lk(k-1) \sum x_i (x_i - 1) + \sum x_i (x_i - 1)]
\]

\[
\Rightarrow \frac{\sum x_i (x_i - 1)}{lk(k-1)} \geq \frac{\sum x_i (x_i - 1)}{nm(m-1)+lk(k-1)} > \frac{\sum x_i (x_i - 1)}{nm(m-1)+lk(k-1)+(l+n)km+(lm+kn)(k+m-1)}
\]

\[
\Rightarrow \frac{\sum x_i (x_i - 1)}{lk(k-1)} > \frac{\sum x_i (x_i - 1)}{(l+n)(m+k)(k+m-1)} \Rightarrow SBFC(M_1) > SBFC(M_3)
\]

Thus, \( Max\{SBFC(M_1),SBFC(M_2)\} > SBFC(M_3) \). This means that the SBFC metric satisfies the cohesive modules property.

### 4.2. Value comparison

By definition, when the number of slices equals one, \( SBFC=SFC=A=WFC \). The following proofs are for the cases when the number of slices is greater than one, given that \( y \) and \( w \) are sets of rows in the matrix indexed by glue and super-glue tokens, respectively.

#### Relation SBFC.1: \( SBFC(M) \geq SFC(M) \)

**Proof:**

\[
SBFC(M) = \sum_{i=1}^{s} x_i (x_i - 1) = \sum_{y \in yw} \frac{x_i (x_i - 1)}{ts(s-1)} \geq \sum_{y \in yw} \frac{x_i (x_i - 1)}{ts(s-1)} = \frac{|y|s(s-1)}{ts(s-1)} = \frac{|y|}{t} = SFC(M)
\]

#### Relation SBFC.2: \( SBFC(M) \leq A(M) \)

**Proof:**

\[
SBFC(M) = \sum_{i=1}^{s} x_i (x_i - 1) \leq \sum_{i=1}^{s} x_i (s-1) = \frac{(s-1) \sum x_i}{ts(s-1)} = \frac{\sum x_i}{ts} \leq \frac{\sum x_i}{ts} = A(M)
\]

#### Relation SBFC.3: \( SBFC(M) \leq WFC(M) \)

**Proof:** Given the proof of Relation SBFC.2,

\[
A(M) = \frac{\sum x_i}{ts} \leq \frac{|y|}{ts} = \frac{|y|}{t} = WFC(M) .
\]

Using Relation SBFC.2, \( SBFC(M) \leq A(M) \leq WFC(M) \)

From the above relations, we conclude that for any module where \( s \geq 1 \), \( SFC(M) \leq SBFC(M) \leq A(M) \leq WFC(M) \).
4.3. Sensitivity comparison

Table 3 shows several representative patterns for matrices used by the SFC, WFC, A, and SBFC metrics. The table shows that the value of the SFC metric is the same for modules A and B, despite the fact that intuition informs us that module B is more cohesive than module A because the output data slices of module A are completely disjoint, whereas some of the output data slices of module B are related to each other. This is due to the fact that the SFC metric is sensitive only to the changes in the cohesive interactions that relate all output data slices together. The WFC metric violates intuition by giving the same results for modules B and D, despite the fact that some of the output data slices of module B are related to each other, whereas all of the output data slices of module D are related to each other. This is due to the fact that the value of the WFC metric is not sensitive to the size of the glue tokens. The A metric violates intuition by giving the same result for modules B and C, despite the fact that only two pairs of data slices of module B are related to each other, whereas all pairs of data slices of module C are related to each other. This is due to the fact that the A metric does not distinguish between two matrices that have the same number of 1’s included in glue tokens, regardless of the patterns in which the 1’s are organized. The SBFC metric follows intuition for all of the listed cases. The metric results show that cohesion(A)<cohesion(B)<cohesion(C); this is expected, because none of the pairs of output data slices of module A share any data token, whereas some of and all of the pairs of output data slices of modules B and C, respectively, share data tokens. Finally, the SBFC metric results show that cohesion(C)<cohesion(D), which is expected because all pairs of output data slices of both modules share common data tokens; however, the pairs of output data slices of module C share fewer common data tokens than those of module D. This shows that the SBFC metric is more sensitive than the other three metrics, and it gives more meaningful and representative results. In addition, the examples demonstrate the usefulness of the SBFC metric as an indicator for restructuring modules. Modules A and B have low SBFC values, which indicates their need for restructuring. This decision can be concluded by intuition from the pattern of the matrices representing the two modules. In contrast, the A metric gives the same results for modules B and C, which violates the restructuring intuition.

<table>
<thead>
<tr>
<th>Module</th>
<th>Matrix pattern</th>
<th>SFC</th>
<th>WFC</th>
<th>A</th>
<th>SBFC</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td><img src="image" alt="Matrix A" /></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td><img src="image" alt="Matrix B" /></td>
<td>0</td>
<td>1</td>
<td>0.5</td>
<td>0.167</td>
</tr>
<tr>
<td>C</td>
<td><img src="image" alt="Matrix C" /></td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>D</td>
<td><img src="image" alt="Matrix D" /></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 3: Values of different cohesion metrics on 4 sample modules

4.4. The SBFC metric and the ordinal scale
Validating that the module functional cohesion measure is a proper numerical characterization of the module functional cohesion is very difficult [5]. In this case, as suggested by Bieman et al. [5], we can rely on human intuition to determine the consistency of the module cohesion with the measure values. This intuition can be based on defining finite and complete set of transformations on a program representation and demonstrating that the ordering imposed by the measure is consistent with the set of transformations. In this section, we rely on the set of transformations defined in [5] as follows:

**Base case:** A single-data slice module: A one slice module is completely cohesive and it should have the highest possible functional cohesion value. The definition of the SBFC, as stated in Section 3, is that it is equal to 1 (i.e., highest value of functional cohesion) if the module is a single-data slice module. Therefore, the SBFC metric satisfies the intuition here.

**Transformations:**

1. Add one slice. A slice can be added to the single-data slice module (i.e., the module formed using the base case) or to a multiple-data slice module (i.e., a module formed using the base case and repetitions of transformations) in two ways:
   a. By adding a new output to the module, which leads to adding at least one new data token that does not belong to any of the existing data slices.
   b. By changing a non-output data token into an output data token. This creates a new data slice without adding any data tokens.

   In cases a and b, the intuition is that the new cohesion depends on the relations, in terms of shared data tokens, between the new data slice and the existing ones. If there are many such relations, the cohesion after the transformation is higher than the cohesion before the transformation and vice versa.

   In terms of SBFC, the cohesion of the module after the transformation becomes higher if the average similarity between pairs of data slices after transformation is greater than the average similarity between pairs of data slices before the transformation. This occurs when the average similarity between the added data slice and each of the existing data slices is greater than the average similarity between the existing pairs of data slices. Using the similarity function given in Formula 1, the similarity between pairs of data slices increases as the number of shared data tokens increases. In other words, the similarity between pairs of data slices increases as the relations between the data slices increases. As a result, the cohesion of the module after transformation becomes greater if the average relations between the added data slice and the existing data slices is greater than the average relations between the pairs of existing data slices, which is consistent with the intuition.

2. Extend a slice by adding one data token to it. There are two cases for the added data token, as follows:
   a. A data token which does not exist in any of the existing data slices. In this case, the intuition is that the overall differences between data slices are increased, and therefore, the overall relative relations between pairs of data slices are decreased. As a result, the cohesion decreases. In terms of SBFC metric, when adding a data token to one data slice, the number of shared data tokens between the updated data slice and each of the other data slices remains the same and the total number of data tokens increases by one, which causes the similarity function given in Formula (1) to decrease. Consequently, the average of the similarity between pairs of data slices decreases, and the SBFC value decreases, which is consistent with the intuition.
   b. A data token that exists in one or more of the other data slices. In this case, the intuition is that the relations between the updated data slices and one or more of the other data slices will increase. As a result, the overall cohesion of the module increases. In terms of the SBFC metric, when adding a data token to a data slice and the data token exists in one or more of other data slices, the number of shared data tokens between the updated data slice and one or more of the other data slices increases. Consequently, the average of the similarity between pairs of data slices increases, and the SBFC value increases, which is consistent with the intuition.
Any data slice abstraction can be built using the base case and repetitions of the transformations discussed above. Removing a data slice and deleting a data token from a data slice are inverse operations of the two transformations discussed above. As a result, the set of transformations is complete and the SBFC metric is on an ordinal scale.

5. Experimental Study

The SBFC metric introduced here was used to compare the quality of six applications in terms of functional cohesion. The six applications were developed by groups of senior undergraduate students for a project in a networking course. The project required building a simple client-server application. All the applications had the same specifications, but they were different in terms of implementation. Because measuring the functional cohesion was a computation- and labor-intensive task, we developed a tool to fully automate the functional cohesion measuring task for modules written in C programming language. For comparison purposes, the tool computed the functional cohesion using SBFC as well as Bieman and Ott’s metrics. The tool is called Similarity-based Functional Cohesion Measure (SiFCoM) tool and it uses Aristotle Analysis System [28]. We have used the tool to measure the SBFC of each function in the six applications. The SBFC of each application was computed as the weighted summation of the SBFC of the functions included in the application. Table 4 summarizes the results per application. The first column of the table shows the application identifier. The second and third columns show the number of lines of code (not including comments and blank lines) and the number of functions in each application, respectively. The fourth, fifth, and sixth columns report the maximum, minimum, and average SBFC result among the functions of the application. The seventh column reports the SBFC of the applications. Table 4 shows that the sixth application was the best in terms of functional cohesion, and its adoption and use are recommended.

<table>
<thead>
<tr>
<th>Application</th>
<th>Number of lines of code</th>
<th>Number of functions</th>
<th>Minimum SBFC</th>
<th>Maximum SBFC</th>
<th>Average SBFC</th>
<th>Application SBFC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>997</td>
<td>15</td>
<td>0.053</td>
<td>1</td>
<td>0.606</td>
<td>0.256</td>
</tr>
<tr>
<td>2</td>
<td>279</td>
<td>5</td>
<td>0.005</td>
<td>1</td>
<td>0.416</td>
<td>0.014</td>
</tr>
<tr>
<td>3</td>
<td>787</td>
<td>20</td>
<td>0.001</td>
<td>1</td>
<td>0.264</td>
<td>0.004</td>
</tr>
<tr>
<td>4</td>
<td>1113</td>
<td>20</td>
<td>0.109</td>
<td>1</td>
<td>0.700</td>
<td>0.295</td>
</tr>
<tr>
<td>5</td>
<td>855</td>
<td>22</td>
<td>0.014</td>
<td>1</td>
<td>0.539</td>
<td>0.025</td>
</tr>
<tr>
<td>6</td>
<td>536</td>
<td>9</td>
<td>0.118</td>
<td>1</td>
<td>0.568</td>
<td>0.320</td>
</tr>
</tbody>
</table>

Table 4: The characteristics and SBFC results of the client-server applications

6. Correlation Study

The six applications used in the study provided in Section 5 were also used to compare experimentally the proposed cohesion metric with the Bieman and Ott ones. The purpose of this study was to illustrate the extent to which the metrics of the two techniques correlate. The tool was used to compute SBFC, SFC, WFC, and A values for each function in the six applications. Table 5 shows Pearson’s correlation coefficient (PCC) [29] between the SBFC metric and the other three cohesion metrics. PCC measures the likelihood of the existence of a linear relationship between two sets of data. Its absolute value is less than or equal to 1, where an absolute value 1 indicates a perfect correlation, and
a value 0 indicates no correlation. In addition, a positive value indicates a linear relationship with a positive gradient, and vice versa. According to [30], the correlation is called trivial, minor, moderate, large, very large, or almost perfect when the PCC value is in the range 0-0.1, 0.1-0.3, 0.3-0.5, 0.5-0.7, 0.7-0.9, or 0.9-1, respectively. Since Table 5 shows that the correlation between the SBFC metric and the other three cohesion metrics is almost perfect, we can infer that the SBFC metric provides a useful alternative to SCF, WFC, and A metrics.

<table>
<thead>
<tr>
<th>Measuring metric</th>
<th>SFC</th>
<th>WFC</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>SBFC</td>
<td>0.980</td>
<td>0.948</td>
<td>0.985</td>
</tr>
</tbody>
</table>

Table 5: Correlation between the cohesion metrics

The C functions of the same applications were also used to study experimentally the correlation between the characteristics of the modules under consideration (MUC) and the cohesion results computed using the SBFC metric and Bieman and Ott’s ones. The considered MUC characteristics are the number of lines of code, the number of nodes in the PDG, the number of edges in the PDG, the number of data slices, the number of data tokens, and the McCabe’s Cyclomatic Complexity (MCC). The correlation results are shown in Table 6. The first column of the table shows the considered characteristics for the MUC. The next three columns show the correlation between the Bieman and Ott’s metrics and MUC characteristics. The last column shows the correlation between the SBFC metric and MUC characteristics.

<table>
<thead>
<tr>
<th>Module characteristic</th>
<th>Bieman's cohesion measure</th>
<th>SBFC</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SFC</td>
<td>WFC</td>
</tr>
<tr>
<td>Number of line of code (LOC)</td>
<td>-0.425</td>
<td>-0.236</td>
</tr>
<tr>
<td>Number of PDG nodes</td>
<td>-0.428</td>
<td>-0.224</td>
</tr>
<tr>
<td>Number of PDG edges</td>
<td>-0.384</td>
<td>-0.200</td>
</tr>
<tr>
<td>Number of data tokens</td>
<td>-0.384</td>
<td>-0.185</td>
</tr>
<tr>
<td>Number of required slices</td>
<td>-0.606</td>
<td>-0.405</td>
</tr>
<tr>
<td>MCC</td>
<td>-0.414</td>
<td>-0.201</td>
</tr>
</tbody>
</table>

Table 6: Correlations between the module characteristics and the cohesion measures

The results reported in Table 6 are used to draw the following observations that support considering SBFC metric as a cohesion indicator.

1. It is most likely that highly modular modules are relatively simple and have a small number of outputs. Such modules most likely are highly cohesive and vice versa. The results reported in Table 6 show that the highest correlation is between the number of required slices (i.e., number of module outputs) and each of the SBFC and Bieman and Ott’s metrics. In this case the correlation is large and negative, which supports the intuition. In addition, we have noticed that the average SBFC value for the simple functions (MCC=0) that have a number of outputs less than 5 is 0.794. On the other hand, the average SBFC value for the complex functions (MCC>10) that have a number of outputs greater than or equal to 5 is 0.1777, which also supports the intuition.
2. It is most likely that small-sized modules perform either one basic function or strongly related functions, and therefore, these modules are relatively highly cohesive. The second highest correlation reported in Table 6 is between each of the number of LOC and number of nodes, and the SFC and SBFC metrics. Both of the number of nodes and LOC indicate the module size. In this case, the correlation is moderate and negative which supports the intuition. In addition, we have noticed that the average SBFC value for the small-sized functions (LOC<40) is 0.647, whereas the average SBFC value for the other functions is 0.203, which also supports the intuition.

7. Conclusions and Future Work

A similarity-based metric for module functional cohesion, called SBFC, is introduced in this paper. The metric uses the average similarity between pairs of data slices of module outputs as an indicator for the functional cohesion of the module. The metric can be used directly to comment on the level of functional cohesion of the module and to compare the functional cohesion of different modules or different implementations of the same module. The simplicity of using the proposed metric is expected to encourage software developers to apply it to assessing the quality and modularity of their products and, therefore, to modifying the products’ code if they have low functional cohesion to improve the products’ quality and modularity. This improvement is expected to reduce the maintenance cost of the products in the long run. In addition, the simplicity of understanding and using the introduced metric is expected to encourage professional software developers who produce high quality applications to provide their measured SBFC values with their products as a promotion. When educated about cohesion as one of the important quality factors, people are willing to purchase and use highly cohesive applications because they are more stable and have lower maintenance costs.

SBFC satisfies the necessary properties of cohesion metrics, is sensitive to changes in the cohesive interactions defined by the module, is easy to be computed, and can be used as an indicator for restructuring the weakly cohesive modules. As a result, the introduced metric is a more useful alternative than pre-existing metrics for measuring a module’s functional cohesion.

An experimental study is introduced to compare the SBFC metric to Bieman and Ott’s metrics and to study the correlation between cohesion metrics and module characteristics. The study observations support the hypothesis that SBFC is a cohesion indicator. The correlation study was performed using a relatively small number of modules collected from student projects, but it could be extended by considering a larger number of modules collected from industrial projects. Currently, research is progressing in three directions. First, the same concepts introduced in this paper are being used to introduce a similarity-based metric for object-oriented class cohesion. Second, the introduced cohesion metric is being used in automating the module decomposition process. Third, the relationship between the cohesion metric introduced in this paper and the program testability and maintainability is being studied empirically.

Acknowledgment
The authors would like to acknowledge the support of this work by Kuwait University Research Grant WI03/04.

References


