A NEW COUPLING METRIC: BLENDING STRUCTURAL AND SEMANTIC RELATIONS

Mamdouh Alenezi  
Department of Computer Science  
North Dakota State University  
Fargo, ND 58108, USA  
mamdouh.alenezi@ndsu.edu

Kenneth Magel  
Department of Computer Science  
North Dakota State University  
Fargo, ND 58108, USA  
kenneth.magel@ndsu.edu

ABSTRACT

Coupling metrics represent the relationships between source code artifacts in software systems. Coupling is considered an important concept in measuring design quality and maintainability. Many coupling measures have been proposed in the context of object-oriented systems. A popular way to measure coupling is through structural properties and static code analysis. Another way to measure coupling is through semantic information encoded in identifiers and comments. However, there is still much to be understood about which aspects of coupling affect quality or other external attributes of software. This paper presents a new coupling metric for object-oriented systems that analyzes both structural and semantic relationships between methods and classes. A study is performed on open source software systems to compare the new metric with existing structural and semantic coupling metrics. The study shows that the new metric captures new dimensions of coupling, which are not captured by existing coupling metrics. This paper investigates the use of the new proposed coupling metrics during change impact analysis. By comparing our new metric to other coupling metrics, we show that our new metric is a better predictor for classes impacted by changes.

KEY WORDS
Software Metrics, Software Maintenance and Quality Issues, Object Oriented Systems, Structural Coupling, Semantic Coupling

1 Introduction

Coupling is considered one of the important software relationships that has been utilized for several tasks related to developing and maintaining software. Coupling captures the degree of interaction between source code artifacts, such as methods. Coupling measures relate closely to software maintenance and evolution. Coupling is usually measured at the class-level by measuring the degree to which two classes depend on one another. In general, one of the goals of the software designers is to keep the coupling in an OO system as low as possible.

Coupling metrics are of different types depending on the type of information they use. Structural coupling which captures interactions such as usage relations between classes and methods or execute after associations or dependencies. Semantic coupling captures the semantic or conceptual similarity between classes and methods. Several research studies showed that coupling is a strong candidate to predict external quality attributes such as fault-proneness [6, 3] and change-proneness [12]. Coupling measures also have been used to measure reengineering effort [18], impact analysis [21], change propagation [14], and clone detection [13].

To capture structural coupling, different coupling metrics were introduced such as Coupling Between Objects (CBO) and Response for a Class (RFC) in Chidamber and Kemerers suite of object-oriented metrics [11]. Other structural class coupling metrics, Efferent Coupling (Ce), Afferent Coupling (Ca) were introduced by [17]. Briand et al. [8] built a unified framework for coupling measurement in object oriented systems.

To capture coupling different coupling metrics were proposed. Poshyvanyk and Marcus [19] outlined a coupling metric for classes based on source code textual information, CoCC (Conceptual Coupling of Classes). Ujjhaz et al. [23] added a (threshold t) to CoCC and came up with a new conceptual metric namely CCB (Conceptual Coupling between Object Classes). Another conceptual class coupling metric was defined by Getzers and Poshyvanyk [15] is called RTC (Relational Topic based Coupling).

Works that tried to combine the structural and textual coupling are the work of Revelle et al. [21], Bavota [4] and Bavota et al. [5]. Revelle et al. [21] combined both textual and structural coupling at the feature level in which they added the two metrics with a weighting criteria. They called this metric HFC (Hybrid Feature Coupling). Bavota [4] combined structural and semantic metrics and hierarchical clustering in order to support software refactoring. Bavota et al. [5] proposed an approach that analyzes the structural and semantic relationships between classes in a package identifying chains of strongly related classes. The identified chains are used to define new packages with higher cohesion than the original package. All these studies employed weighting criteria while in our work, we want to come up with a unified measure that captures both structural and semantic relations of methods and classes.
None of the previous work investigated the effect of combining both structural and conceptual coupling at method and class levels on different applications such as software maintainability. In this work, we would like to introduce new coupling metrics that combine both structural and conceptual coupling at both method and class levels. In addition, we would like to evaluate these new metrics theoretically and perform empirical analysis to investigate the effect of applying these metrics on impact analysis.

This paper makes the following research contributions:

- We define a novel coupling metrics based on both structural and semantic relation of classes and methods for coupling measurement.
- We theoretically validate the newly proposed metric.
- We empirically evaluate the newly proposed metrics against a host of existing structural and semantic metrics.
- We compare our new coupling measure with other coupling measures to see how much they support ranking classes during impact analysis.

2 Related Work

Since our measures try to capture both structural and semantic relations of software entities, we present a literature review for both coupling metrics types. Chidamber and Kemerer [11] introduced a metric suite to capture both coupling and cohesion in OO systems. They introduced the following coupling measures: Depth of Inheritance Tree (DIT), Coupling between objects (CBO) and response for a class (RFC). The DIT metric provides for each class a measure of the inheritance levels from the object hierarchy top. For CBO, two classes are coupled when methods in one class use methods or fields defined by the other class. RFC is defined as set of methods that can be potentially executed in response to a message received by an object of that class. In other words, RFC counts the number of methods invoked by a class. Afferent coupling (Ca) and efferent coupling (Ce), that use the term category (a set of classes that achieve some common goal), were identified by Martin [17]. Ca is the number of classes outside the category that depend upon classes within the category, while Ce is the number of classes inside the category that depend upon classes outside the categories.

Bansiya et al. [2] proposed Number of Public Methods (NPM) which simply counts all the methods in a class that are declared as public. Tang et al. [22] proposed Inheritance Coupling (IC) and Coupling Between Methods (CBM). IC provides the number of parent classes to which a given class is coupled. A class is coupled to its parent class if one of its inherited methods functionally dependent on the new or redefined methods in the class. A class is coupled to its parent class if one of the following conditions is satisfied: one of its inherited methods uses an attribute that is defined in a new/redefined method, one of its inherited methods calls a redefined method, or one of its inherited methods is called by a redefined method and uses a parameter that is defined in the redefined method. CBM measures the total number of new/redefined methods to which all the inherited methods are coupled. There is a coupling when at least one of the given in the IC metric definition conditions is held. All of these existing coupling metrics are defined for classes. Our work is different from previous research in a way that it provides a mechanism to capture and analyze the strength of coupling in two granularity levels (class and method). Our coupling metrics define the coupling at three levels: method level, method-pair level, and class-pair level.

For the semantic relations of software entities, Poshyvanyk and Marcus [19] presented a new set of conceptual coupling measures for Object Oriented systems. They outlined a coupling metric for classes based on source code textual information, CoCC (Conceptual Coupling of Classes). They showed by a case study on C++ open source software systems that the conceptual coupling captured a new dimension of coupling not addressed by structural measures. Ujhazi et al. [23] added a (threshold t) to CoCC and came up with a new conceptual metric namely CCBO (Conceptual Coupling between Object Classes). Another conceptual class coupling metric was defined by Gethers and Poshyvanyk [15], the Relational Topic-based Coupling (RTC) metric, which based on a variant of LDA (Latent Dirichlet Allocation) called Relational Topic Models (RTM). RTM extends LDA by explicitly modeling links between documents in the corpus. RTC uses these links to define the coupling between two documents in the corpus. The authors demonstrated that their proposed metric provides value because it is statistically different from existing metrics. Kagdi et al. [16] used a similar metric, the conceptual similarity between pairs of source code methods, as a part of a novel change impact analysis technique. Revelle et al. [21] introduced Textual Feature Coupling (TFC) which measures the coupling between features based on unstructured, textual information in source code using Latent Semantic Indexing (LSI).

3 Combining Both Structural and Semantic Coupling

3.1 The Proposed Metrics

We propose new coupling metrics that analyze the structural and semantic relationships between methods and classes. The combined metric allows us to analyze the relationship from both structural point of view (i.e., dependencies existing between classes) and from semantic point of view (i.e., cosine similarity of semantic representation).
### 3.1.1 System Representation and Definitions

We first define a representation for a software system.

A software system $S$ is an object-oriented system. $S$ has a set of classes $C = \{c_1, c_2, \ldots, c_n\}$. The number of classes in the system is $n = |C|$. A class has a set of methods. For each class $c \in C, M(c) = \{m_1, m_2, \ldots, m_t\}$ are the set of methods in $c$, where $t = |M(c)|$ is the number of methods in a class $c$. The set of all methods in the system $S$ is denoted by $M(S)$.

### 3.1.2 Hybrid Coupling between Methods

We start by calculating the metric at the method level. Hybrid Coupling between methods (HCM) takes into account both methods dependencies and their direct dependencies along with their cosine similarity of their LSI representation. Latent semantic indexing (LSI) is an indexing method which uses singular value decomposition (SVD) to identify patterns in the relationships between the terms and concepts contained in an unstructured collection of text. LSI is based on the principle that words that are used in the same contexts tend to have similar meanings. The cosine between two vectors is used as a measure of semantic similarity between the terms and concepts contained in an unstructured collection of text. LSI is based on the principle that words that are used in the same contexts tend to have similar meanings. The cosine between two vectors is used as a measure of semantic similarity between two documents (methods) (i.e., determine how much relevant semantic information is shared among methods of different classes in the context of the entire system). The normalization is done by dividing the value by the average values of all method pairs in the two subjected classes.

$m_i$ is a method $\in$ class $k$ ($c_k$) and $m_j$ is a method $\in$ class $l$ ($c_l$). $m_i, m_j$ is the method pair. $MD$ represents the number of inbound and outbound dependencies of a method. A dependency is a relationship between two methods where one uses the services provided by the other (e.g., method $m_i$ calls method $m_j$). For example, we say that method $m_i$ depends on method $m_j$. We also say that $m_i$ has outbound dependency and $m_j$ has inbound dependency. $m_i$ is a dependent to another method if it has outbound dependency on that method. A method is a dependee to another method if it has inbound dependency from that method [1]. $vmi$ represents the LSI representation of the textual content of method $m_i$.

$$DEP(m_i, m_j) = \begin{cases} 1 & \text{if } m_i \text{ depends on } m_j \text{ or } m_j \text{ depends on } m_i \\ 0 & \text{otherwise} \end{cases}$$

HCM calculates the number of dependencies of the method pair in different classes ($MD$), the value will increase if there is a direct dependency between the method pair ($DEP$), also the value increases if there is a strong cosine similarity of the methods representations ($COS(vmi, vmj)$). When there is a direct dependency then $DEP$ plus 1 equals 2 which doubles the value of the measure. Also, the cosine value is between -1 an 1, when there is a high similarity between the two methods then the cosine similarity equals to 1 which also doubles the value of the measure.

### 3.1.3 Hybrid Coupling between Method and a Class

$$HCMC(m_i, c_j) = \frac{\sum_{q=1}^{t} HCM(m_i, m_{jq})}{t}$$

where $t$ is the number of methods in $c_j$. Hybrid Coupling between method and a class (HCMC) is the average of the measure between method $m_i$ and all the methods from class $c_j$.

### 3.1.4 Hybrid Coupling between Two Classes

We define the coupling between two classes $c_i \in S$ and $c_j \in S$ as:

$$HCCC(c_i, c_j) = \frac{\sum_{l=1}^{t} HMC(c_i, c_j)}{t}$$

where $t$ is the number of methods in $c_i$. Hybrid Coupling between two classes (HCCC) is the average coupling measure between all unordered pairs of methods from class $c_i$ and class $c_j$. This equation ensures that the coupling between the two classes is symmetrical.

### 3.1.5 Hybrid Coupling of a Class

we define a coupling measure that approximates the coupling of a class in an OO software system. The class coupling measures the degree to which the methods of a class are coupled to the methods of other classes.

$$SSCM(c_i) = \frac{\sum_{k=1}^{n} HCCC(c_i, c_k)}{n - 1}$$

where $n$ is the number of classes.

### 4 Assessment of the Metrics

In this Section, we are evaluating SSCM theoretically and empirically.
4.1 Theoretical Evaluation

We analyze our new metrics by following the Modified Weyukers Properties [11] and Briand et al. Properties [9, 8]. Only six of the nine original properties by Weyuker apply to OO metrics. The six properties are Noncoarseness, Nonuniqueness, Design Details are Important, Monotonicity, Nonequivalence of Interaction, and Interaction Increases Complexity. We also analyze our new metrics by following the five Briand et al. properties: Nonnegativity, Null value, Monotonicity, Merging of classes, and Merging of unconnected classes.

Our measures comply with both sets of properties based on a comprehensive study. These properties hold, given that the mathematical average have these properties. We start by Weyukers Properties, for Noncoarseness: from our measurements, we discovered that our new metrics satisfy this property. Since not all classes can have the same value for a metric, the new measure satisfies this property. For Nonuniqueness: by analyzing the results obtained from processing 1044 classes, we have found that our new measures satisfy this property since you can find two different classes with the same coupling measure. For Design Details are Important: if we have two classes $c_1$ and $c_2$ providing the same functionality but different number and type of attributes/methods, their corresponding measurement value would be different because of the textual cosine similarity of these classes. For Nonequivalence of Interaction, it is possible for any two classes to have same coupling values. For Interaction Increases Complexity, this property can not be satisfied by our metrics. This property conjectures that interaction between classes increases complexity. This is not necessarily true, because if two connected classes are merged, then in the worst case there will be no relationship eliminated. The common relationship between them would have been eliminated and coupling of the combined class will definitely be less than that of the component classes. Briand et al. [9] stated that this particular Weyuker property is the most controversial of all properties.

For Briand et al. Properties, Nonnegativity and Null value are satisfied because none of the measures have null or negative values (Table 2). For Monotonicity: both sets of properties have this property. If we merge two classes, the metric value will be equal or less than the addition of values of these merged classes. The dependencies values still hold and the cosine value will also still holds. Also, if we add a new method that has strong cosine similarities with methods of other classes, then the cosine similarity will also increase which increases the measure. Merging of classes, and Merging of unconnected classes, we have tried different combinations of connected and unconnected classes from different systems and concluded that all these proprieties are satisfied by our new measures. When merging connected classes and merging unconnected classes, the cosine similarity remains the same because the relocation of the methods inside other classes will not affect the semantic similarities of these methods with methods of other classes. Also, the dependencies of these methods will be the same.

4.2 Case Study Design

We address the following research questions (RQ) in our case study.

- **RQ1**: Is SSCM metric orthogonal as compared to existing structural and conceptual coupling metrics?

- **RQ2**: Do SSCM provides better support for ranking classes during impact analysis than any of the following coupling measures: SCM, CCoC, CCBCm?

We perform a case study on several open source systems to evaluate the proposed coupling measure against existing structural and conceptual measures. The goal of the case study is to determine whether our new coupling measure captures new dimensions in coupling measurement.

4.2.1 Coupling Metrics

To see if the newly proposed metric whether captures new dimensions in coupling measurement, we selected nine existing structural metrics for comparison: DIT, CBO, RFC [11], Ca, Ce [17], NPM [2], IC, CBM [22], and SCM and one semantic metric CoCC [19]. The utilized metrics comes from several metrics suites. The SCM (Structural Coupling Metric) is the same as SSCM except that we don’t include the cosine similarity of LSI representations of methods. In order to see if the combination is affected or not by the semantic relations. The SCM metric is kept in our calculations to make sure that our new measures reveal a new dimension that is not covered by both structural and semantic measures. For the definitions and explanations of these measures please refer to Section 2.

<table>
<thead>
<tr>
<th>Num</th>
<th>System</th>
<th>Ver</th>
<th>Packages</th>
<th>Classes</th>
<th>Methods</th>
<th>Source</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Lucene</td>
<td>2.4</td>
<td>13</td>
<td>283</td>
<td>4211</td>
<td><a href="http://lucene.apache.org/">http://lucene.apache.org/</a></td>
</tr>
<tr>
<td>2</td>
<td>POI</td>
<td>3.0</td>
<td>16</td>
<td>386</td>
<td>6387</td>
<td><a href="http://poi.apache.org/">http://poi.apache.org/</a></td>
</tr>
<tr>
<td>3</td>
<td>Synapse</td>
<td>1.2</td>
<td>29</td>
<td>219</td>
<td>1988</td>
<td><a href="http://synapse.apache.org/">http://synapse.apache.org/</a></td>
</tr>
<tr>
<td>4</td>
<td>Log4j</td>
<td>1.2</td>
<td>22</td>
<td>156</td>
<td>1738</td>
<td><a href="http://logging.apache.org/log4j/">http://logging.apache.org/log4j/</a></td>
</tr>
</tbody>
</table>
4.2.2 Subject Software Systems

For our case study we have chosen four various sized open-source software systems from different domains. The summary of the selected software systems’ sizes is outlined in Table 1.

Lucene provides Java-based indexing and search technology, as well as spell-checking, highlighting and advanced analysis/tokenization capabilities. The POI project consists of APIs for manipulating various file formats based upon Microsoft’s OLE 2 Compound Document format, and Office OpenXML format, using pure Java. Synapse is a simple, lightweight and high performance Enterprise Service Bus (ESB) from Apache. Synapse has support for HTTP, SOAP, SMTP, JMS, FTP and file system transports, Financial Information eXchange (FIX) and Hessian protocols for message exchange as well as first class support for standards such as WS-Addressing, Web Services Security (WSS), Web Services Reliable Messaging (WSRM), efficient binary attachments (MTOM/XOP). Log4j is a well known Java-based logging framework.

4.2.3 Measurement Results

The structural coupling measures were downloaded from PROMISE repository 1 whereas CoCC and our measures were computed using our own tool. Table 2 shows descriptive statistics about the coupling measures.

Table 2. Descriptive Statistics for the Coupling Measures

<table>
<thead>
<tr>
<th>Measures</th>
<th>Min</th>
<th>Max</th>
<th>Med</th>
<th>σ</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIT</td>
<td>1</td>
<td>7</td>
<td>1.82</td>
<td>0.88</td>
</tr>
<tr>
<td>CBO</td>
<td>0</td>
<td>214</td>
<td>10.6</td>
<td>14.32</td>
</tr>
<tr>
<td>RFC</td>
<td>2</td>
<td>390</td>
<td>31.18</td>
<td>31.65</td>
</tr>
<tr>
<td>Ca</td>
<td>0</td>
<td>212</td>
<td>4.64</td>
<td>12.44</td>
</tr>
<tr>
<td>Ce</td>
<td>0</td>
<td>133</td>
<td>6.42</td>
<td>7.77</td>
</tr>
<tr>
<td>NPM</td>
<td>0</td>
<td>101</td>
<td>8.94</td>
<td>10.34</td>
</tr>
<tr>
<td>IC</td>
<td>0</td>
<td>3</td>
<td>0.52</td>
<td>0.64</td>
</tr>
<tr>
<td>CBM</td>
<td>0</td>
<td>20</td>
<td>1.41</td>
<td>2.15</td>
</tr>
<tr>
<td>CoCC</td>
<td>0</td>
<td>0.94</td>
<td>0.06</td>
<td>0.09</td>
</tr>
<tr>
<td>SCM</td>
<td>0</td>
<td>5.72</td>
<td>0.99</td>
<td>0.34</td>
</tr>
<tr>
<td>SSCM</td>
<td>0</td>
<td>19.54</td>
<td>0.88</td>
<td>1.01</td>
</tr>
</tbody>
</table>

4.3 Principal Component Analysis

To answer RQ1, we perform Principal Component Analysis (PCA) on the metrics measured at the class level for the software systems in Table 1 in order to understand the underlying, orthogonal dimensions captured by the coupling measures.

4.3.1 Analysis Procedure

Briand et al. [7] proposed a three steps methodology to analyze software engineering data. Their goal was to make the experiments repeatable and the results comparable. The methodology consists of collecting the data, identifying outliers, and performing PCA.

After we eliminated outliers using the Mahalanobis distance, we performed PCA. PCA is used in our case to identify metrics (i.e., groups of variables) which measure the same underlying dimension (i.e., mechanism that defines coupling) of coupling of a classe.

4.3.2 PCA Results

We performed PCA on the set of 998 classes from 4 different open source software systems. All eleven measures were subjected to an orthogonal rotation. We identified seven orthogonal dimensions spanned by 11 coupling measures. The seven principal components (PCs) capture 91.7% of the variance in the data set, which is significant enough to support our findings. The loadings on each measure in each rotated component is presented in Table 3. Values higher than 0.5 are highlighted as the corresponding measures are the ones we look into while interpreting the PCs. For every Principal Component, we provide the variance of the data set explained by the PC and the cumulative variance in Table 3.

Based on our analysis of the coefficients associated with every coupling measure within each of the rotated components, we interpret PCs as the following. PC1 (19%): DIT, IC, and CBM scored more than 0.7 in this component. All three metrics measure the inheritance relationship. PC2 (17%): CBO and Ca count import coupling from non-library classes through method invocations. PC3 (15%): RFC and Ce capture coupling, based on method invocations. PC4 (13%): RFC and NPM count the number of accessible methods. PC5 (9%): SCM measure the direct and indirect dependencies of the method pairs. PC6 (9%): SSCM takes into account both structural and semantic relations. PC7 (9%): CoCC measures semantic coupling of classes.

The PCA results show that SCM is the only significant factor in PC5 and SSCM is the only significant factor in PC6 which indicates that the SCM and SSCM measures define two new dimensions on their own. These results clearly show that our coupling measure captures different types of coupling between classes, than those captured by the structural metrics and the semantic measure. This also shows the fact that SCM and SSCM are coupling measures that are based on completely different ideas and measurements than the existing coupling measures.

In addition, the results of our PCA analysis are very close to those reported in the literature [7, 19]. The PCs and loadings obtained in our case and those reported in the literature do not completely overlap because we used a slightly different set of coupling metrics in our analysis.

1https://code.google.com/p/promisedata/
### 4.4 Using SSCM in Impact Analysis

Impact analysis entails detecting source code elements impacted by a change to a given source code element. The coupling measures can help order (rank) classes in software systems, based on different types of dependencies among classes, captured by the coupling measures [10]. Such coupling measures and derived ranks of classes can be computed automatically. We use a history of changes observed in our selected systems to determine whether our proposed coupling measure can be used to identify classes with common changes (i.e., changes related to the same fault report and having the same ID in the bug tracking system).

In this case study, the SSCM and SCM measures are compared with CoCC and CCBCm measures to evaluate whether they provide better support for impact analysis or not. Previous study [20] showed that CCBCm (The maximum conceptual coupling between two classes) was the best among nine structural measures to provide support for impact analysis. The premise is that given the nature of the captured information by SSCM, it should capture different aspects of coupling among classes as compared to SCM, CoCC and CCBCm which utilize only structural or semantic information.

#### Table 4. Precision (P) and recall (R) values for impact analysis based on 5 cut points

<table>
<thead>
<tr>
<th>Metric</th>
<th>Lucene</th>
<th>POI</th>
<th>Synapse</th>
<th>Log4j</th>
<th>Log4j</th>
<th>Log4j</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSCM</td>
<td>P</td>
<td>R</td>
<td>P</td>
<td>R</td>
<td>P</td>
<td>R</td>
</tr>
<tr>
<td>Lucene</td>
<td>0.12</td>
<td>0.11</td>
<td>0.04</td>
<td>0.03</td>
<td>0.02</td>
<td>0.01</td>
</tr>
<tr>
<td>POI</td>
<td>0.02</td>
<td>0.12</td>
<td>0.12</td>
<td>0.1</td>
<td>0.01</td>
<td>0.18</td>
</tr>
<tr>
<td>Synapse</td>
<td>0.26</td>
<td>0.12</td>
<td>0.13</td>
<td>0.07</td>
<td>0.01</td>
<td>0.15</td>
</tr>
<tr>
<td>Log4j</td>
<td>0.185</td>
<td>0.11</td>
<td>0.143</td>
<td>0.05</td>
<td>0.143</td>
<td>0.02</td>
</tr>
</tbody>
</table>

In order to determine the number of candidate classes suggested for inspection during impact analysis, different strategies can be used. The most common approach is to use a cut point (i.e., select the top n classes from the list).

#### Table 5. Precision (P) and recall (R) values for impact analysis based on 10 cut points

<table>
<thead>
<tr>
<th>Metric</th>
<th>Lucene</th>
<th>POI</th>
<th>Synapse</th>
<th>Log4j</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSCM</td>
<td>P</td>
<td>R</td>
<td>P</td>
<td>R</td>
</tr>
<tr>
<td>Lucene</td>
<td>0.1</td>
<td>0.12</td>
<td>0.024</td>
<td>0.04</td>
</tr>
<tr>
<td>POI</td>
<td>0.22</td>
<td>0.14</td>
<td>0.11</td>
<td>0.12</td>
</tr>
<tr>
<td>Synapse</td>
<td>0.12</td>
<td>0.19</td>
<td>0.07</td>
<td>0.11</td>
</tr>
<tr>
<td>Log4j</td>
<td>0.2</td>
<td>0.15</td>
<td>0.136</td>
<td>0.06</td>
</tr>
</tbody>
</table>

#### Table 6. Precision (P) and recall (R) values for impact analysis based on 15 cut points

<table>
<thead>
<tr>
<th>Metric</th>
<th>Lucene</th>
<th>POI</th>
<th>Synapse</th>
<th>Log4j</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSCM</td>
<td>P</td>
<td>R</td>
<td>P</td>
<td>R</td>
</tr>
<tr>
<td>Lucene</td>
<td>0.06</td>
<td>0.12</td>
<td>0.02</td>
<td>0.05</td>
</tr>
<tr>
<td>POI</td>
<td>0.19</td>
<td>0.15</td>
<td>0.14</td>
<td>0.11</td>
</tr>
<tr>
<td>Synapse</td>
<td>0.08</td>
<td>0.19</td>
<td>0.05</td>
<td>0.11</td>
</tr>
<tr>
<td>Log4j</td>
<td>0.19</td>
<td>0.17</td>
<td>0.147</td>
<td>0.08</td>
</tr>
</tbody>
</table>

In case of each coupling measure we varied a cut point from 5, 10, and 15 classes. Average precision (P) and recall (R) for each class are computed for every metric. Precision is the percentage of classes suggested by a metric that are actually changed together with the given class according to the bug report. Recall is the percentage of the classes that are changed together with the given class and are successfully retrieved using the coupling measures. The results in Tables 4, 5, and 6 show that SSCM is the best indicator among the studied coupling measures of an external property of classes in OO systems-change proneness. This coupling measure can be effectively used to rank relevant classes during impact analysis in OO systems. The measure performed better on average than any of the studied metrics.

We executed Kruskal-Wallis test separately for precision and recall values (see Table 7). In the both tests, since the p-value is very small, the decision is to reject the null hypothesis of absence of differences between studied cou-
Table 7. The results of running two Kruskal-Wallis tests for precision and recall

<table>
<thead>
<tr>
<th></th>
<th>Test 1 precision</th>
<th>Test 2 recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>chi-squared</td>
<td>13.0851</td>
<td>34.3045</td>
</tr>
<tr>
<td>DF</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>p-value</td>
<td>0.004456</td>
<td>1.709e-07</td>
</tr>
</tbody>
</table>

pling metric values. In other words, both tests have shown that the differences between precision and recall values are statistically significant.

5 Threats to Validity

There are several factors that may affect the results of the experiments and they are presented in this section. With any empirical study based on selecting software applications, there is a threat to external validity concerning the representativeness of the applications used. We have demonstrated that our metrics capture new dimensions in coupling measurement; however, we obtained these results by analyzing classes from only four Java open-source systems. In order to allow for generalization of results, large-scale evaluation is necessary, which takes into account systems from different domains, developed using different programming languages.

The structural dependencies were extracted using Dependency Finder tool. To mitigate the risks of using this tool, we create the dependencies for three small Java systems manually and then we compare it with the output created by the Dependency Finder tool. Both sets of data provided consistent results.

Our impact analysis evaluation is based on the changed classes extracted from patches in related bug reports. Some of these patches may contain incomplete information about actually changed classes. We mitigate this issue by considering only officially approved patches in the selected systems.

6 Conclusion and Future Work

The paper defines a novel set of coupling measures, which are theoretically and empirically validated, that capture both structural and semantic relations of software entities. A comprehensive case study showed that these metrics capture new dimensions in coupling measurement, compared to existing structural and semantic metrics. Our new measure SSCM, appears to be a superior indicator of change ripple effects as compared to other coupling measures and can be effectively used to rank classes in the course of impact analysis in a large OO system. In the future, we are investigating the applications of this new measure in change proneness and refactoring.

References


