Semantic Coupling Between Classes: Corpora or Identifiers?

Nemitari Ajienka
Brunel University London
Kingston Lane, Uxbridge
Middlesex, UB8 3PH

Andrea Capiluppi
Brunel University London
Kingston Lane, Uxbridge
Middlesex, UB8 3PH

(nemitari.ajienka,andrea.capiluppi)@brunel.ac.uk

ABSTRACT

Context: Conceptual coupling is a measure of how loosely or closely related two software artifacts are, by considering the semantic information embedded in the comments and identifiers. This type of coupling is typically evaluated using the semantic information from source code into a words *corpus*. The extraction of words corpora can be lengthy, especially when systems are large and many classes are involved.

Goal: This study investigates whether using only the class identifiers (e.g., the class names) can be used to evaluate the conceptual coupling between classes, as opposed to the words corpora of the entire classes.

Method: In this study, we analyze two Java systems and extract the conceptual coupling between pairs of classes, using (i) a corpus-based approach; and (ii) two identifier-based tools.

Results: Our results show that measuring the semantic similarity between classes using (only) their identifiers is similar to using the class corpora. Additionally, using the identifiers is more efficient in terms of precision, recall, and computation time.

Conclusions: Using only class identifiers to measure their semantic similarity can save time on program comprehension tasks for large software projects; the findings of this paper support this hypothesis, for the systems that were used in the evaluation and can also be used to guide researchers developing future generations of tools supporting program comprehension.

Keywords

Semantic coupling; Semantic similarity; Corpora; Corpus; Vector Space Model (VSM); Latent Semantic Indexing (LSI); Object-oriented software (OO); Open-source software (OSS)

1. INTRODUCTION

In general, humans seamlessly process synonyms and other word relations. For example, a developer skimming for code related to "*removing an item from a shopping cart*" understands that the method *deleteCartItem(Item)* is relevant, even though it uses the synonym 'delete' instead of 'remove'.

The large size and complexity of today's software systems necessitates the development of automatic tools to effectively and efficiently help in program comprehension tasks [18]. Such tools use similarity measures on software artifacts (source code, documentation, maintenance requests, version control logs, etc.) to facilitate the comprehension of object-oriented (OO) coupling [15]; inform change impact analysis [4]; and focus refactoring, for example by detecting *antonym* methods, i.e. those methods performing opposite functionality within the same class [18].

Semantic similarity between classes has been mostly achieved by reducing classes to *corpora*: Information Retrieval (IR) techniques such as Latent Semantic Indexing (LSI) and the Vector space model (VSM) approach are the better known ones [4, 9, 15, 19]. The basic idea is that two classes are related when similar terms are present in their comments and identifiers. This approach was introduced by Poshyvanyk and Marcus [15] in a study in which they computed the conceptual similarity between OO software classes written in C++.

Differently from corpora-based techniques, short terms and sentences have been used to detect the semantic similarity between classes (e.g., Path, Information Content and Gloss Based Techniques [18]). These tools are all based on WordNet [11], a lexical database of English word and sense relations. However, since these techniques compute similarity scores based on a probability distribution from English text, their performance deteriorates on a specialized domain, such as software [18].

The objective of this paper is to evaluate these two families of approaches, and to establish whether the conceptual coupling between classes using the classnames of Java files, produces comparable results to using the corpora of the classes content (i.e., the class own source code). If a similarity was to be observed, practitioners could focus their efforts on establishing a conceptual similarity between class names, rather than extracting the corpora of classes, a step that requires more time to complete than just considering the class name.

We consider one small data storage for Java (UrSQL[1], some 1K lines of source code) and a larger Java media server (Ps3MediaServer[2], 40K lines of source code), to test our approach. One corpora-based approach (VSM) is adapted and

ESEM '16, September 08-09, 2016, Ciudad Real, Spain

DOI: http://dx.doi.org/10.1145/2961111.2962622

[1] https://code.google.com/archive/p/ursql/

[2] https://code.google.com/archive/p/ps3mediaserver/

streamlined through scripts; and two sentence-based tools (N-Gram distance and Disco) are tested as the identifier-based techniques. The conceptual coupling is evaluated between each pair of classes in the latest available revision of the two projects. We test the following null hypothesis (H_0): *The semantic similarity between the identifiers of software classes is not a reflection of the semantic similarity between their corpora.*

This paper is organized as follows: in Section 2 we discuss the related studies and outline the methodology in Section 3. Section 4 sets out and discusses the results. In Section 5 we outline the threats to the validity of this study and the measures taken to address them. Finally, in Section 6 we conclude and briefly discuss future opportunities in this area.

2. RELATED WORK

In this section, we report the related work regarding the use of semantic similarity metrics between software artifacts to infer their semantic coupling. We also report the related studies that have compared such tools and techniques.

Comparative Studies.

Software maintenance has been split into three main categories in the literature and these are: adding of new features, correction of software faults and refactoring to accommodate future changes [12]. In a study by Mockus and Votta [12] to classify maintenance efforts into one of these categories based on the textual description of changes, they do not use the synonym capability of WordNet because the relationships between concepts used in software maintenance might differ from the relationships found in natural language. They remove non-alphanumeric symbols, convert all words to lowercase and only use WordNet to generate the stem of words to reduce the number of keywords e.g. mapping fixed and fixing to a single term fix.

Sridhara *et al.* [18] suggest that applying English-based semantic similarity techniques to software without any customization could be detrimental to the performance of the tools. This is a similar reason for which [12] did not rely on WordNet for the synonyms of software terms present in source-code (e.g., dump ↔ export).

In a comparative study by Kumar *et al.* [8], the authors provide an insight into Information Retrieval (IR) techniques using the classical vector space model (VSM), its variant LSI and mathematical lattice based Formal Concept Analysis (FCA). Their results show that the retrieval performance of these techniques is similar.

Semantic Similarity between Software Artifacts.

Poshyvanyk and Marcus [15] defined a coupling metric for classes based on textual information extracted from source code identifiers and comments. Their conceptual coupling metric, CoCC (which stands for Conceptual Coupling of Classes), captures a new dimension of coupling not addressed by structural or dynamic measures. More recently, Ujhazi *et al.* [19] extended the CoCC, defining the new CCBO metric (Conceptual Coupling between Object Classes).

Fluri *et al.* use a set-based similarity metric to explore how comments and code evolve over time [1]. Kuhn *et al.* [7] proposed the use of IR techniques to exploit linguistic information found in source code, such as identifier (i.e., class or method) names and comments. Revelle *et al.* [17] define new feature coupling metrics based on structural and textual source code information.

Kagdi *et al.* [4] in their study on integrating conceptual and logical coupling metrics for change impact analysis identified that the measurement of the conceptual metrics is better at the class level of granularity than at the method level. This is because the documents are reduced in terms (and frequency). In other words, a corpus for a class is typically much "bigger" than a corpus for a method [4]. This informs our choice of conducting this study at the class level of granularity. In most of these studies, they extract the semantic similarity between software classes using the LSI or VSM approach.

3. METHODOLOGY

This paper investigates the effectiveness of computing the semantic similarity between classes using (i) their corpora and (ii) only their identifiers. In this section we describe the steps taken to evaluate the two sets of results, and how the statistical comparison was achieved. We describe the methodology with examples, in order to help the replicability of this study: an overview of the steps performed in this study is visible in Figure1.

Step 1: Data Extraction.

The subjects of this study are pairs of object-oriented software classes written in Java. Due to space constraints, in this study we only analyze classes collected from two randomly selected software systems of different sizes.

Table 1 shows some basic demographics about these projects: UrSQL is a small project, with 4 classes, and 6 class pairs. Given that the semantic similarity measure is symmetric, i.e., the similarity between class pairs A ↔ B and B ↔ A is always the same, from the 4 classes A, B, C and D we only obtain the following 6 possible combinations of class pairs: (AB, AC, AD, BC, BD, CD). On the other hand, Ps3MediaServer is a larger project, with 189 classes and around 40 KLOCs, resulting in 17K pairs of classes for which the conceptual coupling was evaluated.

The same Table reports also on the time taken to extract the two sets of results, one from the corpora-based technique and the other set using two identifier-based tools. The Δ is also evaluated for the two systems, as the comparison of the identifier-based tools versus the corpora-based technique.

Step 2a: Corpora-based technique (VSM).

In this paragraph, we outline the steps which we streamlined to extract and analyze the semantic information embedded in the comments and identifiers of a software class as a whole [16]. The underlying mechanism used to extract and analyze the semantic information from the source code is the Vector Space Model (VSM) technique. The three main steps of VSM are as follows:

1. **Building the corpus**: The Java source code of the software system is converted into a text corpus where each line contains elements of the implementations of a class. The corpus in this case ("dictionary" of terms derived from comments, identifiers in source code) is built based on the class level of granularity [4] such that after the source code is parsed – each line in the extracted corpus will include class identifiers, method names and comments for each class. Mappings be-

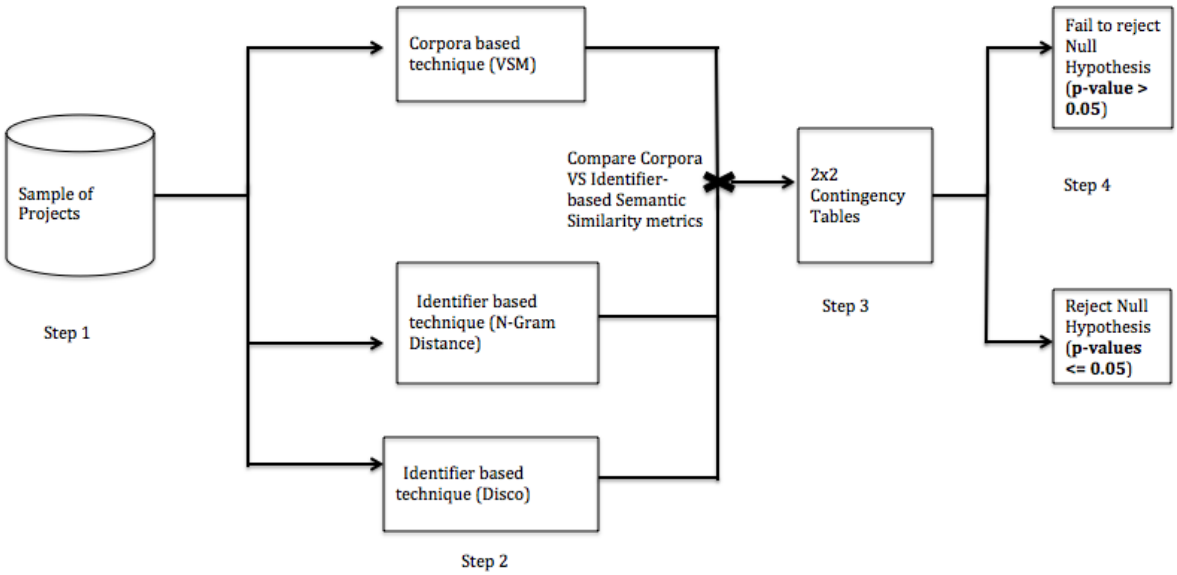

Figure 1: Summary of the methodology

Table 1: Characteristics of the software systems used in the empirical study

Project	Classes	Class Pairs	LOC (with comments)	Time to Analyze Corpora (mins)	Time to Analyze Identifiers (mins)	Δ mins
UrSQL	4	6	1,194	0.006	0.05	< 1%
Ps3MediaServer	189	17,766	39,816	6	0.05	119%

tween classes, and their indexes in the system corpus are generated in this step. Pre-processing of the system corpus is performed to eliminate common keywords, stop words, split and to stem identifiers [9]. After this, each line corresponding to each class in the corpus are saved in separate text (.txt) files/documents.

```
1  package tmacsoftware.ursql;
2
3  public class UrSQLEntry
4  {
5
6      private String key;
7      private String value;
8      private String firstKey;
9      private String firstValue;
10     private UrSQLEntity entity;
11
12     public UrSQLEntry()
13     {
14     }
15
16     public UrSQLEntry(String query)
17     {
18         String[] split = query.split
   (UrSQLController.KEY_VALUE_SEPARATOR);
19         this.key = split[0];
20         this.value = split[1];
21         this.firstKey = this.key;
22         this.firstValue = this.value;
23     }
```

Figure 2: UrSQLEntry.java Source Code Snippet

As an example, for the lines of code shown in Figure 2 (the UrSQLEntry.java class from the UrSQL project), we derive the following corpus using the tool: {*ur sql entri string kei string valu string kei string valu ur sql entiti entiti ur sql entri ur sql entri string queri string split queri split ur sql control kei valu separ kei split valu split kei kei valu valu*}.

2. **Indexing the corpus**: Using the documents generated in the previous step, the tool creates a term-by-document matrix, which captures the distribution of terms in classes and transforms each document into a vector, in three steps:
(1) the *frequency* of each term in every document is calculated, to measure the number of times a term occurs in a document and this is normalized. Suppose for a document relating to the UrSQLEntry.java class in Figure 2, there are overall 50 terms and the term "sql" occurs 25 times, its term frequency will be 25/50 = 0.5.
(2) The *inverse document frequency* (IDF) is calculated for each term. This is done to minimize the impact of words with very high frequencies, as well as weigh up the effect of less occurring terms with the help of a logarithmic function.
(3) For each term we compute the *product* of its normalized term frequency with its IDF on each document (TF*IDF); from which we derive a vector for each document.[3] The cosine is used as a measure of semantic similarity between two documents.
These three steps are explained in more detail in [3]. Just as cosine values range from −1 to 1, so do textual similarities. The closer a value is to one, the more similar the texts of the classes are. These values are used to infer how pairs of classes are closely related (semantically coupling).

[3]The document vectors have been implemented in the form of arrays in the tool

3. **Compute semantic similarities**: the Vector Space Model (VSM) information retrieval technique is used to retrieve information for class-to-class similarity in semantic space based on the corpora extracted from all classes. The last two steps involving the computation of the similarities have been implemented using MATLAB in most of the semantic coupling studies [4].

Step 2b: Identifiers-based tools.

The steps taken to compute the semantic similarity between pairs of class identifiers are as follows:

1. **CamelCase Conversion**: For each identifier, we developed a shell script to transform them from CamelCase to Snake Case, and split them into shorter sentences. for example, the *'UrSQLEntry'* class in the UrSQL project is parsed to derive *'Ur S Q L Entry'*.
2. **Semantic Similarities – N-Gram Technique**: We compute the pair-wise sentence similarities using a Java implementation of the N-Gram distance algorithm [5] introduced by Kondrak [6]. This technique is based on the edit distance and shared sub-strings of length n between sentences and it has been widely used in the literature on text analysis [5]. An example is the semantic similarity between the identifiers *'Ur S Q L Controller'* and *'Ur S Q L Entry'* which returns a value of 0.6 for shared sub-strings between 0 and 4.We have used n-grams of size 4 in this study as information retrieval research [5, 10] has shown that n=4 increases precision for words or terms in several languages including English, French, German, Italian and Swedish. In addition, long lengths of n increase lexicon size, will not represent short words well and processing n-grams sizes larger than 10 is slow [5].
3. **Semantic Similarities – DISCO Tool**: We use the DISCO publicly available sentence similarity Java tool,[6] that measures the semantic similarity between sentences according to the synonyms of their words. DISCO is an enhancement of the Vector-Space analysis found within the Classifier4j[7] which has been used in IR research. The tool is based on a dictionary of the EOWL list of words[8], while the synonyms for each word are calculated using the DISCO's semantics[9] to get groups of the most similar words in the dictionary. For example, **shy** ↔ *timid, quiet, introverted, lonely, cautious, awkward, clumsy, soft-spoken* and *gentle*. An example of using the tool to compute the semantic similarity between two class identifiers *'Ur S Q L Controller'* and *'Ur S Q L Entry'* returns a value of 0.89 with a computation time of 0.003 minutes.

[4]MATLAB is a high-level technical computing language and interactive environment for algorithm development, data analysis and visualization
http://uk.mathworks.com/products/matlab/
[5]Source code available at https://github.com/tdebatty/java-string-similarity\#n-gram
[6]Available at https://sourceforge.net/projects/semantics/?source=directory
[7]http://classifier4j.sourceforge.net/
[8]http://dreamsteep.com/projects/the-english-open-word-list.html
[9]http://www.linguatools.de/disco/disco_en.html

Step 3: Statistical Test for Correlation.

In this step, we performed a Chi-square statistical test to discover if the similarity measures from one class identifier-based technique (say, the N-Gram) produces similar results to the corpus-based technique (VSM).

For each project, we populated a 2X2 contingency table, composed of row (i.e., *groups*) and column (i.e., *outcomes*) variables. The first contingency table visible in Table 2 is a generic 2x2 contingency Table, with the corpus-based outcomes (VSM) as the outcomes variable, and the identifier-based outcomes (N-Gram and Disco) as the groups variable. For the statistical test, we used three semantic similarity thresholds $t = 0.25$, 0.50 and 0.75. These thresholds have been used in various controlled experiments [20].

If s is the semantic similarity between pairs, and using a similarity threshold t (with a lower t implying a weaker similarity), the items of the contingency table are:

- A: pair of classes with $s \geqslant t$ for both Corpora-based and Identifier-based techniques;
- B: pair of classes with $s < t$ for one technique but $\geqslant t$ for the other;

The following are the possible outcomes observed for the threshold t – for the Identifier-based technique:

- C: pair of classes with $s \geqslant t$ for one technique but $< t$ for the other;
- D: pair of classes with $s < t$ for both techniques.

Generic Contingency Table

	Corpora-Based (VSM)	
Identifier-Based	A	B
	C	D

VSM vs N-Gram Comparison - **UrSQL** project (p=.000532)

	$\geqslant 0.25$	< 0.25
$\geqslant 0.25$	3	0
< 0.25	0	3

VSM vs Disco Comparison - **UrSQL** project (p=.000532)

	$\geqslant 0.25$	< 0.25
$\geqslant 0.25$	3	0
< 0.25	0	3

Table 2: Contingency Tables: generic (top) and populated (middle and bottom) with Identifier (either N-Gram or Disco) vs Corpus Based (VSM) techniques

The other two tables (middle and bottom of Table 2) report the values and results for (i) VSM as the column variable, and N-Gram as the row variable and (ii) VSM as the column variable, and Disco as the row variable for the UrSQL Project, with $t = 0.25$.

After populating the contingency Tables, we tested for the association between the semantic similarity measures derived from the pairs of techniques (the identifier and corpus

based) using the Chi-square test. This test is used to compare categorical data. It asserts the independence of the two techniques, with a null hypothesis H_0 of *no association between their outcomes*. We set the p-value at 0.05 as the threshold to reject the null hypothesis and compute the chi-square tests for each project.

4. RESULTS AND DISCUSSION

Similarly to the test results presented in Table 2, we present the results for the semantic similarity thresholds ($t = 0.25; t = 0.5; t = 0.75$) tested for the two projects studied in Table 3. The first column in Table 3 contains the statistical test ID, the second to the fifth column contain the OSS project name, the semantic similarity techniques involved in the test, the semantic similarity threshold used and the outcome of the test (p-value) which we rely on for rejecting or failing to reject the null hypothesis (if p-value $\leqslant 0.05$) as shown in Figure 1 respectively.

In the smaller project (UrSQL) we can reject the null hypothesis H_0, but only when the semantic similarity threshold is set to 0.25. When the threshold is set to 0.75, there is a division by zero error because all the outcomes are in cell D (refer to Table 2 - generic). For the UrSQL there is not a statistical significant evidence of association between the corpora-based technique and the identifiers-techniques.

Considering the larger project (*Ps3MediaServer*) we see a relationship between the corpora-based technique (VSM) and the identifier based techniques (Disco and N-Gram) for all the semantic similarity thresholds used. The p-values for all the tests are < 0.0001, thus we can reject the null hypothesis and accept the alternative hypothesis H_1 for these tests - *There is an association between the semantic similarity measures of the corpora and identifier based techniques.*

These preliminary results indicate that measuring the semantic similarity between object-oriented (OO) software classes based on their identifiers alone can reflect the same results as when the whole source code corpus is extracted. This is an important result, also considering the efficiency of both approaches (corpora and identifiers) in terms of computation. The fifth column in Table 1 shows that time was saved when we computed the semantic similarity between classes using their identifiers (0.05 minutes or 3 seconds) for the bigger project (Ps3MediaServer) compared to the corpora (6 minutes), however more time was taken to analyze the class identifiers in smaller project (UrSQL). This is important for large scale software projects with hundreds of thousands of lines of code which could be lengthy to be analyze in terms of time with the corpora based technique (VSM), whereas just seconds or minutes are needed with the identifier based techniques.

5. THREATS TO VALIDITY

The following are the threats to validity that we identified for this study. In this section we also propose our take on how to solve them.

Construct Validity.

The Vector Space Model technique, just like LSI is an IR technique that uses the co-occurrences of words in documents to discover hidden semantic relations between words. Since the technique is based on co-occurrences of words, the resulting word relations are not guaranteed to be semantically similar [18]. However this technique is widely used in semantic coupling research. Similarly, the N-Gram method is also based on shared sub-strings between sentences. We have also used three semantic similarity thresholds (0.25, 0.5 and 0.75) to compare the three different similarity techniques used in this study.

Internal Validity.

There were usually higher values in cell D (refer to the generic Table in Table 2 for the larger project studied. This might have caused bias in the chi-square test computation. On the other hand, this means that in the larger project analyzed, most of the classes had no semantic similarity and both the corpora and the identifier based techniques were able to capture this. Other OO software projects could have majority of classes that are highly semantically related.

External Validity.

We have only used two software projects of different sizes in terms of number of class pairs and it is possible that conclusions drawn from this study may not generalize to all class pairs in general. It is possible that different conclusions could be drawn when considering a more extensive set of class pairs. Therefore, we encourage an extension of this study on more software projects.

6. CONCLUSION

The work presented here sits in the areas of program comprehension and software maintenance. This study seeks to answer the question: "can semantic coupling between classes be captured via class names, rather than with source code corpora?". We used Corpora-based and Identifier-based techniques to assess whether either could provide a better view of semantic similarity between pairs of classes.

Our contributions have shown that:
(i) identifier-based techniques (i.e., using class names) are more efficient in terms of computation time (faster) and
(ii) the similarity measures derived from the class identifiers and the corpus-based approaches tend to reflect each other, so complex corpora provide relatively similar semantic information as class names.

Our findings are relative to only two systems analysed, but we concluded that measuring the semantic similarity between class identifiers is a more efficient approach when trying to identify the semantic coupling between classes. In addition, researchers who might want to carry out semantic coupling research or replicate our work do not need to be experts in data analysis as we have implemented a tool for the corpus based approach [10] [11].

As further work, recent studies [2, 14] have shown that it is possible that the structural coupling and co-evolution of OO software classes are caused by other types of relationships (e.g., semantic dependencies). The need to study the interplay between semantic dependencies and both coupling and co-evolution has been presented in [13, 14].

As future work, we plan to study the interplay between semantic and co-change dependencies in open-source software projects to ascertain the evolutionary consequences of

[10]The tool can be downloaded at: https://github.com/najienka/SemanticSimilarityJava

[11]The two projects studied have also been added to the tool repository for replication.

Table 3: Summary of Chi-Square (Contingency Table) Test Results

Test ID	Project	Chi-Square Test (Technique A vs B)	Semantic Similarity Threshold t	p-value	Reject or Fail to Reject H_0?
1	UrSQL	VSM ↔ N-Gram	0.25	0.000532	Reject
2		VSM ↔ Disco		0.000532	Reject
3		VSM ↔ N-Gram	0.5	0.3711	Fail to Reject
4		VSM ↔ Disco		0.121	Fail to Reject
5		VSM ↔ N-Gram	0.75	NA	NA
6		VSM ↔ Disco		NA	NA
7	Ps3MediaServer	VSM ↔ N-Gram	0.25	< 0.0001	Reject
8		VSM ↔ Disco		< 0.0001	Reject
9		VSM ↔ N-Gram	0.5	< 0.0001	Reject
10		VSM ↔ Disco		< 0.0001	Reject
11		VSM ↔ N-Gram	0.75	< 0.0001	Reject
12		VSM ↔ Disco		< 0.0001	Reject

semantic dependencies between classes.

7. REFERENCES

[1] B. Fluri, M. Würsch, and H. C. Gall. Do code and comments co-evolve? on the relation between source code and comment changes. In *Reverse Engineering, 2007. WCRE 2007. 14th Working Conference on*, pages 70–79. IEEE, 2007.

[2] M. M. Geipel and F. Schweitzer. The link between dependency and cochange: empirical evidence. *Software Engineering, IEEE Transactions on*, 38(6):1432–1444, 2012.

[3] Q. Guo. The similarity computing of documents based on vsm. In *Network-Based Information Systems*, pages 142–148. Springer, 2008.

[4] H. Kagdi, M. Gethers, and D. Poshyvanyk. Integrating conceptual and logical couplings for change impact analysis in software. *Empirical Software Engineering*, 18(5):933–969, 2013.

[5] V. Kešelj, F. Peng, N. Cercone, and C. Thomas. N-gram-based author profiles for authorship attribution. In *Proceedings of the conference pacific association for computational linguistics, PACLING*, volume 3, pages 255–264, 2003.

[6] G. Kondrak. N-gram similarity and distance. In *String processing and information retrieval*, pages 115–126. Springer, 2005.

[7] A. Kuhn, S. Ducasse, and T. Gírba. Semantic clustering: Identifying topics in source code. *Information and Software Technology*, 49(3):230–243, 2007.

[8] C. A. Kumar, M. Radvansky, and J. Annapurna. Analysis of a vector space model, latent semantic indexing and formal concept analysis for information retrieval. *CYBERNETICS AND INFORMATION TECHNOLOGIES*, 12(1), 2012.

[9] A. Marcus, A. Sergeyev, V. Rajlich, J. Maletic, et al. An information retrieval approach to concept location in source code. In *Reverse Engineering, 2004. Proceedings. 11th Working Conference on*, pages 214–223. IEEE, 2004.

[10] P. Mcnamee and J. Mayfield. Character n-gram tokenization for european language text retrieval. *Information retrieval*, 7(1-2):73–97, 2004.

[11] G. Miller and C. Fellbaum. Wordnet: An electronic lexical database, 1998.

[12] A. Mockus and L. G. Votta. Identifying reasons for software changes using historic databases. In *Software Maintenance, 2000. Proceedings. International Conference on*, pages 120–130. IEEE, 2000.

[13] G. A. Oliva and M. Gerosa. Experience report: How do structural dependencies influence change propagation? an empirical study. In *Proceedings of the 26th IEEE International Symposium on Software Reliability Engineering*, 2015.

[14] G. A. Oliva and M. A. Gerosa. On the interplay between structural and logical dependencies in open-source software. In *Software Engineering (SBES), 2011 25th Brazilian Symposium on*, pages 144–153. IEEE, 2011.

[15] D. Poshyvanyk and A. Marcus. The conceptual coupling metrics for object-oriented systems. In *Software Maintenance, 2006. ICSM'06. 22nd IEEE International Conference on*, pages 469–478. IEEE, 2006.

[16] D. Poshyvanyk, A. Marcus, R. Ferenc, and T. Gyimóthy. Using information retrieval based coupling measures for impact analysis. *Empirical software engineering*, 14(1):5–32, 2009.

[17] M. Revelle, M. Gethers, and D. Poshyvanyk. Using structural and textual information to capture feature coupling in object-oriented software. *Empirical software engineering*, 16(6):773–811, 2011.

[18] G. Sridhara, E. Hill, L. Pollock, and K. Vijay-Shanker. Identifying word relations in software: A comparative study of semantic similarity tools. In *Program Comprehension, 2008. ICPC 2008. The 16th IEEE International Conference on*, pages 123–132. IEEE, 2008.

[19] B. Újházi, R. Ferenc, D. Poshyvanyk, and T. Gyimóthy. New conceptual coupling and cohesion metrics for object-oriented systems. In *Source Code Analysis and Manipulation (SCAM), 2010 10th IEEE Working Conference on*, pages 33–42. IEEE, 2010.

[20] J. Winkelman, K. D. Sethi, C. Kushida, P. Becker, J. Koester, J. Cappola, and J. Reess. Efficacy and safety of pramipexole in restless legs syndrome. *Neurology*, 67(6):1034–1039, 2006.