An Empirical Study on the Interplay between Semantic Coupling and Co-Change of software classes

Article *in* Empirical Software Engineering · October 2017

CITATIONS
0

READ
1

3 authors, including:

Andrea Capiluppi
Brunel University London
94 PUBLICATIONS **991** CITATIONS

SEE PROFILE

Noname manuscript No.
(will be inserted by the editor)

An Empirical Study on the Interplay between Semantic Coupling and Co-Change of software classes

Nemitari Ajienka · Andrea Capiluppi · Steve Counsell

Received: date / Accepted: date

Abstract Software systems continuously evolve to accommodate new features and interoperability relationships between artifacts point to increasingly relevant software change impacts. During maintenance, developers must ensure that related entities are updated to be consistent with these changes. Studies in the static change impact analysis domain have identified that a combination of source code and lexical information outperforms using each one when adopted independently. However, the extraction of lexical information and the measure of how loosely or closely related two software artifacts are, considering the semantic information embedded in their comments and identifiers has been carried out using somewhat complex information retrieval (IR) techniques. The interplay between software semantic and change relationship strengths has also not been extensively studied. This work aims to fill both gaps by comparing the effectiveness of measuring semantic coupling of OO software classes using (i) simple identifier based techniques and (ii) the word corpora of the entire classes in a software system. Afterwards, we empirically investigate the interplay between semantic and change coupling. The empirical results show that: (1) identifier based methods have more computational efficiency but cannot always be used interchangeably with corpora-based methods of computing semantic coupling of classes and (2) there is no correlation between semantic and change coupling. Furthermore we found that (3) there is a directional relationship between the two, as over 70% of the semantic dependencies are also linked by change coupling but not vice versa.

Keywords information retrieval (IR) · co-change · co-evolution · clustering · coupling · change impact analysis (CIA) · object-oriented (OO) · open-source software · software components · hidden dependencies (HD)

N. Ajienka, A. Capiluppi and S. Counsell
Brunel University London
Kingston Lane, Uxbridge
Middlesex, UB8 3PH
E-mail: (nemitari.ajienka,andrea.capiluppi,steve.counsell)@brunel.ac.uk

1 Introduction

Software Change Impact Analysis (CIA) is an essential technique for identifying the potential ripple effects caused by software changes during software maintenance and evolution [9,70]. CIA techniques can be typically static or dynamic [62], depending on how the information is collected to analyse its change impact. Dynamic techniques rely on information gathered during program execution to compute the change impact set while static techniques are centred around the source code, semantic information and change dependencies. Because of the many false positives and the effort required in dynamic analysis (collecting data during execution and analyzing data during execution), static techniques have gained popularity [62].

Most studies on static impact analysis have shown that certain classes, identified by patterns or metrics, are more likely to be impacted by a change and, hence, practitioners will need to invest extra effort in their future maintenance. Other studies, specifically addressed at establishing a link between coupling and co-change, have found that the set of co-changed classes was much larger compared to the set of structurally coupled classes [21,25,46,47]. This implies that not all of the change dependencies are related to structural dependencies and there could be other reasons for software artefacts to be change dependent [47]. High coupling between classes in an OO design can increase system complexity by introducing multiple inter-dependencies among the classes [61]. Moreover, excessive coupling can complicate testing, make additional changes problematic and limit possibilities for reuse [54]. Software that is not flexible or tolerant to modification is usually destined to abandonment or replacement [45].

Kagdi and Maletic have estimated that there is a *hidden dependency* (HD) between two classes or two methods if the classes or the methods are changed at the same time in the past [33]. As Yu *et al.* stated [76]: '*hidden dependencies among software artefacts make both understanding and maintenance difficult*'. Briand *et al.* showed that if developers are required to handle a large set of dependencies, they would miss a significant number of them [9]. Poshyvanyk and Marcus detected dependencies using information retrieval techniques [52]. In a similar way to HD, complex dependencies are captured by semantic information which is hard to detect by traditional program analysis techniques [65]. Some CIA tools do not discover HD, and it is the responsibility of the programmer to correctly identify and trace HD during change impact analysis [51].

In the last few years, a new dimension has been identified as a hidden dependency, termed "semantic" coupling, that could have an influence on coupling and co-change. Simply defined, semantic coupling is a measure of how loosely or closely related two software artefacts are, by considering the semantic information embedded in the comments and identifiers. According to Bavota *et al.* [6]:

> 'the peculiarity of the semantic coupling measure allows it to better estimate the mental model of developers than the other coupling mea-

sures. This is because, in several cases, the interactions between classes are encapsulated in the source code vocabulary (...)'.

In the conceptual framework for software dependency management proposed by Oliva and Gerosa [45], semantic coupling is not considered as one of the dependencies to be measured. They state that software dependencies are the 'primary' subject of management, and the identification of dependencies involves capturing structural and logical dependencies. Nonetheless, the same authors claim that there is still a need to study the interplay between semantic and logical coupling in OO software as well as the interplay between structural and semantic coupling [46, 47]. They identified a small intersection between the sets of structural and logical dependencies after analyzing commits from the Apache Software Foundation repository. When directly assessing semantic coupling, researchers in the software evolution and dependency domain have demonstrated that semantic coupling metrics can outperform structural metrics in identifying classes that might be impacted by a given change request [53]. Semantic and logical coupling metrics have also been combined in change impact analysis [35, 40].

Researchers have suggested that frequent change coupling indicates a strong structural coupling between the corresponding modules, sub-modules, or files as well as possible shortcomings in the design of a software system [21]. However, the frequency of change couplings have not yet been studied in relation to semantic coupling. The computation of semantic coupling in studies in this domain have been done by using information retrieval techniques such as latent semantic indexing (LSI) and vector space modelling (VSM) [34, 52, 53] to analyze the corpora of OO software classes (after transforming the semantic information from source code into a text or words *corpus*).

In a pilot study, we observed that the extraction of word corpora can be time-consuming, especially when systems are large and many classes are involved [2]. With the goal of identifying how the computation of the semantic coupling of classes can be improved, we statistically compared the metrics derived from analyzing the corpora of classes against an analysis of only their identifiers. Results revealed that identifier based metrics reflect the corpora based measurements. In addition, identifier based measurements were more efficient in terms of computation time, especially when analyzing large software classes (e.g., $>$ 1000k lines of source code). It is important to further validate results derived from the pilot study with a larger sample of projects to improve generalizeability. In addition, the results will further contribute to knowledge on how to ease semantic coupling measurement in further studies that rely on semantic coupling information of classes in OO software.

Given the current state of the art in the area of software coupling, and extending our previous work, we shift the focus of the change impact analysis to the semantic link between object-oriented (OO) software classes in 79 OSS projects (written in Java). This paper examines the *strength of semantic coupling* [52] between pairs of classes, through the evolution of various software systems, and correlates it with the likelihood of their future co-change.

Establishing whether there is an interplay between logical and semantic coupling has several applications in software engineering including:

1. **Co-change inferred by semantic coupling**: understanding the influence of semantic coupling on co-change can also help to infer the co-change frequency of software classes based on semantic coupling strengths, i.e., semantic coupling metrics can be used to directly inform practitioners about potential unplanned co-changes of classes in OO software projects.
2. **Improving software tools to detect hidden dependencies**: "hidden" dependencies not detected by software maintenance tools during change impact analysis that cause co-change would be detected with significant precision.
3. **Minimizing historical data extraction and analysis efforts**: semantic coupling metrics will be used to inform or predict the strength of the logical dependencies between classes without the need to analyze historical data of software projects, thus reducing the effort required (i.e., computation time and data storage) in the detection of logical dependencies *via* mining software repositories. The semantic similarity between class identifiers will also be used in the ranking of classes that might be impacted by a given change request without having to analyze software evolution or historical data, thus minimizing the effort required in change impact analysis [34].

This work is articulated as follows: in Section 2 we describe the definitions, research goals and steps taken to carry out this study, with the help of a worked example to show the empirical approach. Sections 3 and 4 highlight the results of our study, followed by a discussion on the importance of these findings. Section 5 highlights the threats to validity and in Section 6 we summarise the related work. Finally, our conclusions and areas for further research are presented in Section 7.

2 Research Methodology

In this section, we present the definitions for the different types of coupling (2.1), the motivating scenario (2.2) and the goals of this research (2.3). Additionally, we highlight, with the use of worked examples, the steps performed in the methodology: data collection (2.4); computing the coupling types (2.5); evaluating their intersection (2.6); and performing the statistical tests (2.7).

2.1 OO Software Dependencies

A dependency is a semantic relationship that indicates that a client element may be affected by changes performed in a supplier element [47]. In sections 2.1.1 and 2.1.2, we introduce semantic and logical dependencies and discuss how they can be operationalised in an OO context.

2.1.1 Logical Coupling

According to Wiese *et al.*, "*change coupling is a phenomenon associated with recurrent co-changes found in the software evolution history*" [68]. Co-evolution of classes can be represented with their change, logical or evolutionary coupling [75,77] (as shown in Figure 2). Therefore, the logical coupling of any two classes is based on their change history, and is a measure of the observation that the two classes always co-evolve or change together [15, 23, 24, 67]. They are commonly treated as association rules [78], which means that when X_1 is changed, X_2 is also changed [47]. Furthermore, X1 and X2 are called the antecedent (i.e., left-hand-side, LHS) and the consequent (i.e., right-hand-side, RHS) of the rule, respectively. For example, the rule {A, B}→ C found in the sales data of a supermarket indicates that a customer who buys A and B together, is also likely to buy C [47].

Two classes change at the same time when changes in one class A are made in response to a change in another class B. Kagdi *et al.* [34] state that logical coupling captures the extent to which software artefacts co-evolve and this information is derived by analysing patterns, relationships and relevant information of source code changes mined from multiple versions (of software systems) in software repositories (e.g., Subversion and Bugzilla). According to Lanza *et al.* [14] it is useful to study logical coupling because it can reveal dependencies not revealed by analyzing the source code [75] only. These sort of dependencies are the most troublesome and are the source of many defects in software. In this study, we adapt the methods proposed by Zimmermann *et al.* [77] to represent logical dependencies.

Operationalisation The logical dependency between classes and its degree, is evaluated in this work using the **support** and **confidence** metrics. By doing so, we evaluated the *significance* of the association rules between classes [47], and across the lifespan of a software project (i.e., taking all versions of the software system into consideration).

The *support* value counts the number of revisions where two software artifacts (i.e., classes) were changed together. In other words, the probability of finding both the antecedent and consequent in the set of revisions. For example, in Figure 1, class A was modified in 3 transactions (where 3 is the *"Transaction Count"* [75]). Out of these 3 transactions, 2 also included changes to the class C. Therefore, the support for the logical dependency $A \rightarrow C$ will be 2. By its own nature, support is a symmetric metric, so the $A \rightarrow C$ dependency also implies $A \leftarrow C$. The support value of a given rule determines how *evident* the rule is [69].

On the other hand, the *confidence*[1] value of a dependency link measures the degree of the logical dependency and normalizes the support value by the total number of changes of the causal class, or the antecedent of the association rule. Numerically, it is the ratio of the support count to transaction count: from

[1] Also called the support ratio [75]

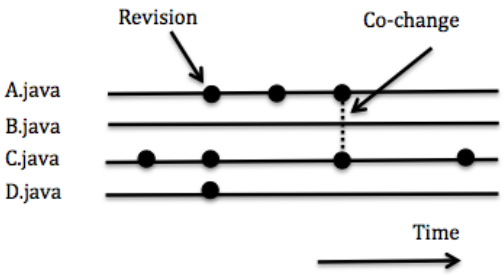

Fig. 1: Association rule example for confidence and support metrics

Figure 1, the confidence value for the association rule $A \rightarrow C$ (which states that C depends on A) will have a high confidence value of $2/3 = 0.67$. In contrast, the rule $C \rightarrow A$ (which states that A depends on C) has a lower confidence value of $2/4 = 0.5$. In other words, the confidence is directional, and determines the strength of the consequence of a given (directional) logical dependency. The confidence value is the *strength* of a given association rule [69].

Logical coupling is directional, thus $A \rightarrow C$ (changes made to class A resulted in changes in C) and $C \rightarrow A$ (changes in C caused changes in A) will have different meanings. As a result, the confidence for these two cause $\rightarrow$ effect rules can be different.

2.1.2 Semantic Coupling

Some studies have used the term *"semantic"* [4,6–8,26,35,53,55], while others have used the term *"conceptual"* [26] to describe the same concept. Poshyvanyk *et al.* [53] state that *conceptual coupling* captures the degree to which the identifiers and comments from different classes relate to each other [4,6–8,35, 55]. Gethers *et al.* [26] add a twist to the definition and state that conceptual coupling captures the extent to which domain concepts/features and software artefacts are related to each other. However, both definitions have things in common. They are limited to the underlying meanings of unstructured text in the source code of software entities (e.g., classes) and how these underlying meanings relate to each other. Furthermore, this relationship is derived in the form of metrics (-1 to 1, where 1 = high semantic coupling [52]).

Identifiers used by developers for names of classes, methods, or attributes in source code or other artifacts contain important information and account for approximately half of the source code in software [34]. These names often serve as a starting point in many program comprehension tasks. Hence, it is essential that these names clearly reflect the concepts that they are supposed to represent, as self-documenting identifiers decrease the time and effort needed to acquire a basic comprehension level for a programming task [34].

2.2 Motivating scenario

Figure 2 shows a simplified scenario that underlies our motivation: previous research [75] (pictured inside the grey box) has shown that the structural coupling between classes causes them to be co-changed, and it plays an important role in the measurement of co-evolution [75]. However Yu [75] has used correlation to infer a causal relationship and research has shown that correlation does not always mean causality [18]; there are different ways to identify causal relationships. In addition to correlation studies which investigate linear relationships, we have also investigated the presence of a directional relationships between semantic and logical coupling. Our contribution, expressed in this research, includes semantic coupling in the picture: we posit that the semantic coupling of classes leads to their co-change and logical coupling (evolutionary dependencies).

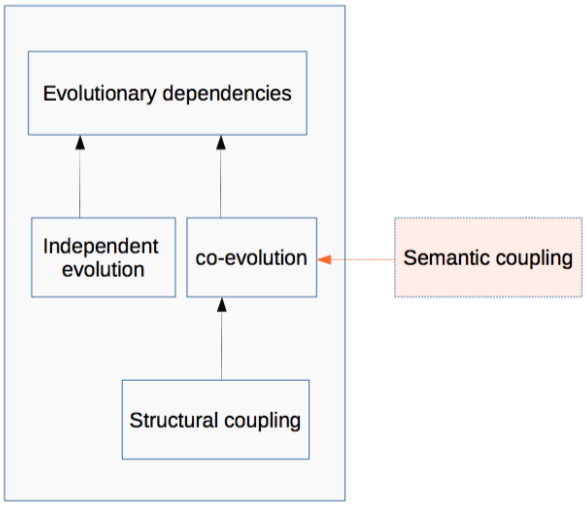

Fig. 2: Motivating example: structural coupling → co-evolution [75] and semantic → co-evolution (to be checked in this work)

2.3 Research goals

The work we present is based on the three following **goals**:

G1: to establish with a larger sample of OSS projects whether the semantic coupling between classes using the class names of Java files produces comparable results to using the corpora of the classes content (i.e., the class own source code) [2];

G2: to investigate how the semantic coupling **strength** between classes has an impact on their future co-changes;

G3: to investigate the **directionality** of the relationship between logical and semantic (as motivated by Figure 2) by identifying the proportion of logical dependencies that involve semantically related elements (*"hidden dependencies"* [65]) and *vice-versa*.

Research questions were derived from each goal, and testable hypotheses formulated for each question, as summarised in Table 1.

Table 1: Research Goals and Questions

Goals	Research Questions	Null Hypothesis H_0
G1	[Q1] *Can semantic coupling between classes be captured via class names, rather than with source code corpora?*	There is no association between the semantic similarity measures of the corpora and identifier based techniques
G2	[Q2] *Is there a linear relationship between logical and semantic coupling?*	No linear relationship between the strengths of logical and semantic dependencies
G3	[Q3] *Is there a directional relationship between semantic and logical coupling?*	No directional relationship

2.4 Empirical Data collection

In the next subsections, we present how and what kind of data we collected from the repositories of the studied sample of OO software projects.

2.4.1 Selection of a sample of OSS projects

Leveraging the FlossMole project, we used its latest available data dump to determine the population of GoogleCode: a total of 2,593,222 projects are listed in the November 2012 dump.[2] Given their language descriptions, we extracted the subset of Java projects from that population, obtaining 49,459 Java projects. Each project in the subset was given a unique ID: using a 95% confidence level, and a 5% confidence interval, a random sample of 380 IDs were extracted, and linked to the Java projects' names.

2.4.2 Storage of projects metadata and revisions

The first phase of this activity was centered on obtaining the metadata (e.g, name of developers, date and time of changes, etc.) of each project in the sample. The repository of each project was downloaded and stored, with its

[2] Data dump is available at http://flossdata.syr.edu/data/gc/2012/2012-Nov/

metadata, using the CVSAnalY set of tools.[3] The process to obtain the metadata for all the projects took around 48 hours: `sleep` statements were inserted in a routine not to overload the online servers, and to make sure that the latest versions of the files were downloaded. The metadata allowed us to obtain the list of revisions for each class, and for the whole project. The second phase was to get all the revisions of each project; from this we could identify the trivial projects (with < 20 revisions) and exclude these from the study. As a result, we ended up with 79 non-trivial Java open-source software projects with 117 revisions on average.

There is a chance that re-sampling to retrieve a larger sample of projects could result in a larger number of trivial projects with less than 79 left. Previous studies have also excluded a number of OSS projects after their initial sampling. Samoladas *et al.* [58] and Gousios *et al.* [27] applied certain selection criteria to exclude projects from their initial selection. Midha and Palvia [44] based on certain project selection criteria, reduced their initial sample from 887 to 283. Haefliger and Spaeth [28] reduced their selected sample of projects to 6 OSS projects with variance on their sampling criteria. The studied sample included a wide variety of software products such as office software, games, a hardware driver, and an instant messenger client and this reduced sampling bias [60]. Similarly in this study, the resulting non-trivial sample of 79 OSS projects are of different domains, sizes and levels of activity. The sample selection criteria widely used in OSS research [13, 56] and adopted by Haefliger and Spaeth, includes: 1) the project is under active development, allowing the tracking of its development activity [4], 2) the source code modifications of the project need to be available online, and 3) the project should have been in existence for at least a year.

Because the process of analyzing the correlation between identifier and corpora based methods of computing semantic coupling is labour-intensive [5], we focused our attention on a subset of this sample of projects when answering RQ1 [12] while ensuring that the subset consists of projects of varying sizes representative of the overall sample.

[3] http://metricsgrimoire.github.io/CVSAnalY/

[4] Prior research [36] shows that 75% of OSS projects on Github have over 20 commits and 90% have less than 50 commits. We selected projects with above 20 commits to retrieve a variety of projects with varying levels of development activity in our sample, improve generalizeabiliy of the study as well as extract substantial change history to understand logical coupling.

[5] To answer RQ1, it becomes imperative to compute VSM using a Java tool to parse the corpus of classes after the stemming of words, converting class identifier names from camel case to snake case with a Shell script, and computing correlations in the R statistical environment

2.5 Identifying class dependencies (RQ1)

In the following subsections, we present how the class dependencies were calculated with examples. We also present assumptions and decisions made during this task.

2.5.1 Logical Coupling

For each project, we extracted the number of revisions, based on the tables built by CVSAnalY. This task was a pure SQL extraction task, so it did not pose a time issue. For all revisions, we extracted the list of pairs of classes that were co-evolving in that revision and stored this data in a .CSV file. An example of the co-evolution data is provided in Table 2, detailing an excerpt of the Java classes that co-evolve in the *UrSQL* project in its 4^{th} revision. The first column shows the project name, the third and fourth columns show classes that were co-changed, through association rules.

Table 2: Co-evolution data for Project *UrSQL* (excerpt)

Project Name	Rev	class A	class B
UrSQL	4	UDO	Filio
UrSQL	4	UDO	Main
UrSQL	4	UDO	UrSQLController
UrSQL	4	UDO	UrSQLEntity
UrSQL	4	UDO	UrSQLEntry
UrSQL	4	Filio	UDO
UrSQL	4	Filio	Main
UrSQL	4	Filio	UrSQLController

Using the *arules* [6] library in the **R** [7] environment for association rule mining [30], we were able to compute the Confidence metric for each pair of classes with an established logical dependency. Similar to prior research, the support and confidence thresholds have been set to 0.01 and 0.1 respectively [37]. This is because increasing the support and confidence increases precision but lowers recall (i.e., only a small number of association rules are identified when the minimum confidence value is higher than 0.01). The number of identified co-evolving class pairs reduces based on increase in confidence and such pruning looses important information [78]. Oliva and Gerosa classified confidence metrics as: [0.00-0.33] low logical coupling, [0.33-0.66] medium logical coupling and [0.66-1.00] high logical coupling and identified that highly logically coupled classes suffered slightest influence from structural coupling. In addition, the *arules* library in the R statistical environment has been used with a high precision and minimal false positives in prior research across different disciplines [29, 31, 37] when mining frequent item sets from data.

[6] https://cran.r-project.org/web/packages/arules/index.html
[7] https://www.r-project.org/

2.5.2 Semantic Coupling

In a previous study [2] described in Section 1, we compared two sentence similarity techniques (based on N-Gram [8] and Disco Word synonym[9] categories methods) against a corpus or text document cosine similarity based technique (VSM)[10] [11] for computing semantic similarity between Java classes. The study was conducted using two software projects and identified by means of Chi-squared independence tests that measuring the semantic similarity between classes using (only) their identifiers is similar to using the class corpora. This is because using identifiers was more efficient in terms of recall, and computation time [2]. The study also identified that English based word similarity techniques such as WordNet do not perform well on software terminologies (e.g., export ↔ dump). The steps taken to compute the semantic similarity between Java classes using the three techniques is explained in detail in [2].

In addition, the N-Gram technique is based on the edit distance and shared sub-strings of length n between sentences [38]. An example is the semantic similarity between two class 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 based on prior information retrieval research [38, 43] that shows that n=4 increases the precision when analyzing words or terms in various languages. The N-Gram technique also performs better on text from other languages [2] apart from English [38, 43] compared to English based text similarity methods like WordNet. The Disco technique although with low precision on non-English words has been compared to other text similarity techniques and proven to perform well when its outputs were manually checked. According to Despotakis *et al.* [17] "although the precision for Disco was low, it did provide additional valuable concepts, which were approved by both experts. We also manually checked the outputs of the semantic similarity measurements to minimize the effects of false positives."

In this study, we extend the previous study with a larger sample of OO software projects written in the Java programming language. The statistical methods applied in investigating the association between the corpus and identifier-based techniques are described in subsections 2.7.1 and 2.7.2. We also compare the techniques with three semantic dissimilarity thresholds ($\boldsymbol{t}$ = **0.1, 0.2, and 0.5**) based on what has been used previously in the literature (0.195 [38]; between 0.1 and 0.2 [63], 0.2 [19, 59]; and 0.5 [10, 11]).

[8] A Java implementation of the N-Gram distance algorithm is available at https://github.com/tdebatty/java-string-similarity#n-gram

[9] The Disco sentence similarity measures the semantic similarity between sentences according to the synonyms of their words. A Java implementation of the tool is publicly available at https://sourceforge.net/projects/semantics/?source=directory

[10] We have developed a tool that uses the VSM method to automate the corpus based technique. It can be downloaded at: https://github.com/najienka/SemanticSimilarityJava

[11] Two out of the studied projects have also been added to the online repository for replication.

2.6 Evaluating the intersection of sets (RQ2)

Once pair-wise semantic and logical dependencies were identified and the associated coupling values were calculated, we then built a spreadsheet *per* project based on the data with the following columns; LHS (antecedent), RHS (consequent), semantic similarity, and confidence. Using a Shell script, we could parse the data and identify the proportion of semantic dependencies that involved non-logical dependencies (i.e., $A - B$ from the sets in Figure 3), the proportion of logical dependencies that involved non-semantic dependencies (i.e., $B - A$ from Figure 3) as well as the intersection set of pairs of classes that are both semantically and logically related (i.e., $A \cap B$ from Figure 3).

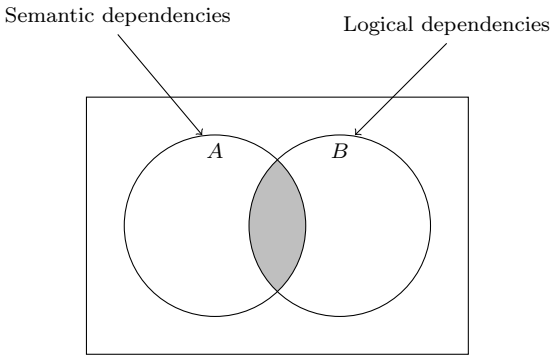

Fig. 3: Intersection of semantically and logically coupled classes

2.7 Statistical tests

Both RQ1 and RQ2 are linked to formal statistical testing. Below the two tests (Chi-Squared for RQ1 and Spearman for RQ2) are illustrated in the context of the analysed systems.

2.7.1 Chi-Squared (X^2) Test (RQ1)

To answer RQ1, 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 3 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 dissimilarity thresholds $t = 0.1$, 0.2, and 0.5.

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)

	$\geqslant 0.1$	< 0.1
$\geqslant 0.1$	3	0
< 0.1	0	3

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

	$\geqslant 0.1$	< 0.1
$\geqslant 0.1$	3	0
< 0.1	0	3

Table 3: 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 3) 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 $t = 0.1$.

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 method (chisq.test) in **R**[12]. 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.

[12] http://courses.statistics.com/software/R/Rchisq.htm

2.7.2 Spearman's Rank Correlation ρ (RQ1 and RQ2)

This section describes the computation of Spearman's rank correlation statistical tests for RQ1 and RQ2.

To further answer RQ1, using the semantic dissimilarity thresholds (0.1, 0.2, and 0.5) described in Section 2.5.2, we will in addition to the Chi-square test compute the linear correlation between the corpora based semantic similarity measurement technique and the identifier based techniques to verify whether the semantic coupling metrics reported by the different techniques for the same pairs of classes co-vary.

The intersection of dependency sets (from 2.6) is used to evaluate the relationship between the coupling types. All the values of the logical coupling strength (i.e., the *confidence* metrics) and all the values of the semantic coupling strength, are pulled together, *per* pair of classes, *per* project and along their string of revisions. Given a project, we created two vectors [13], one with the values of 'semantic similarity' between classes; the other with all the values of co-change confidence between classes.

Each observation in both vectors contain the semantic coupling between two classes and the confidence of their co-evolution or logical coupling metric. Notwithstanding, the semantic coupling metric is a symmetric metric whereby the semantic coupling is the same in both directions (for example a pair of classes A and B in a software project Y, A → B will have the same semantic coupling metric as B → A). The logical coupling metric however is not symmetric. The association rule A → B measures the strength of the following observation: "when A is modified, there will always be a change in B" [77]. Therefore, A → B and B → A are not treated as the same association rule (the confidence metric could be different but the semantic coupling metrics is the same) and are considered as different observations in the created vectors.

Computing a linear correlation between the strengths of the semantic and logical coupling of classes will help to further answer questions such as: "What is the strength of the relationship between semantic and logical coupling of classes?", "are classes with a high degree of semantic coupling likely to co-evolve frequently?". Various correlation coefficients have been considered including Pearson, Kendall and Spearman. However, for Pearson's to be valid the data has to follow a normal distribution [49,75] (the mean, median and mode have to be the same) while Kendall tau is used in small sample sizes and where there are multiple values with the same score [20] and interpreted based on the probability of concordant and discordant observations. Finally, p-values derived from Kendall tau are more accurate with smaller sample sizes.

The null hypothesis H_0 to be tested for RQ2 is as follows:

- H_0: *No linear relationship between the strengths of logical and semantic dependencies.*

[13] By the term *vectors* we refer to the distribution of values for the logical and the semantic coupling for the analyzed pairs of classes

The correlation between the two vectors is evaluated using the Spearman's rank correlation coefficient [75]. The Spearman's metric (non-parametric) was chosen because it is unlikely that either the semantic or logical coupling values will have a normal distribution [49]. Additionally, some classes might not be changed in all the revisions in which they are semantically coupled. Nevertheless the vectors will be of the same size or contain the same number of observations with the confidence metric in one and the semantic coupling metric in the other.

We adapt the categorization of correlation coefficients by Marcus and Poshyvanyk [42] as follows: ($[0 - 0.1]$ to be insignificant, $[0.1 - 0.3]$ low, $[0.3 - 0.5]$ moderate, $[0.5 - 0.7]$ large, $[0.7 - 0.9]$ very large, and $[0.9 - 1]$ almost perfect) if the rank correlation coefficient proves to be statistically significant.

We reject the null hypothesis for all the projects studied at the 95% confidence level. In other words, if the rank correlation coefficient proves to be statistically significant at the $\alpha = 0.05$ level, we will reject the null hypothesis and fail to reject the alternative hypothesis H_1: *There is a linear relationship between the logical coupling and semantic coupling of OO software classes.* The results derived for all projects are described in Section 3.2. The $\alpha = 0.05$ level was chosen as suggested in Yu's study [75]. One of the threats to the statistical validity to their study was the selection of the significance level. In that study, they chose $\alpha = 0.1$ which might have resulted in a type I error - mistakenly rejecting a null hypothesis. To reduce this threat, they planned in future research to decrease the α value to 0.05 for more accuracy (which we have done herein).

3 Results

Following the methodology outlined above, this section presents the results of the three analyses, as performed on the selected projects. The aim is to answer the research questions outlined in Table 1.

3.1 RQ1. *Can semantic coupling between classes be captured via class identifiers, rather than with source code corpora?*

A Chi-squared test of independence was carried out to investigate the independence of the semantic coupling metrics measured using:

1. A corpora based technique (VSM) and
2. A couple of identifier based techniques (N-Gram and Disco word synonym category-based)

The measurement was done at the class level of granularity as mentioned in Section 6.1 and based on the results derived from the statistical test we could either reject or fail to reject the null hypothesis (H_0) presented in Table

1 for RQ1 : *There is no association between the semantic similarity measures of the corpora and identifier based techniques.*

Figure 4 shows the distribution of the p-values *per* studied OO software project derived from the Chi-squared test of independence. The box-plot also shows that we considered three semantic dissimilarity thresholds ($t = 0.1$, 0.2, 0.5) based on previous studies [10,11,19,38,59,63] on text similarity, whereby any class pairs with a measure below the threshold were not considered as semantically coupled.

Fig. 4: RQ1- Chi-square association test results for class corpora (VSM) vs identifier (N-Gram, Disco word synonym category) based semantic similarity techniques (box-plot distribution of p-values for threshold $t = 0.1$, 0.2 and 0.5)

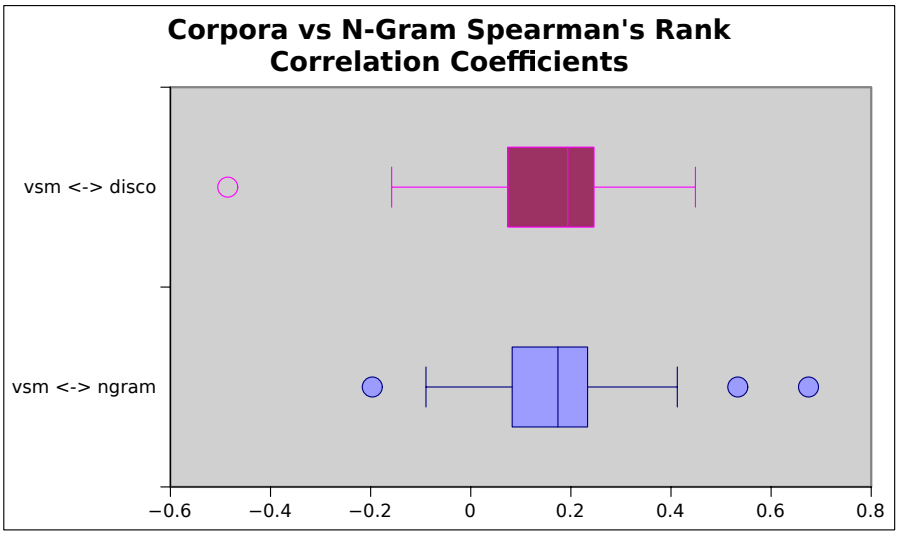

Fig. 5: RQ1- Spearman's rank correlation results for class corpora (VSM) vs identifier (N-Gram, Disco word synonym category) based semantic similarity techniques (box-plot distribution of Spearman's rank correlation coefficients)

We reject the null hypothesis at the 95% confidence level with only a 5% error margin. In other words, we consider results as statistically significant where the p-value is below or at $\alpha = 0.05$ level.

Figure 4 shows that when the threshold is set to 0.1, not all the p-values fall below 0.05. Therefore, we cannot reject the null hypothesis. When the threshold is set to 0.2, a similar condition for 0.1 also applies for the VSM $\leftrightarrow$ N-Gram tests with many outliers above the 0.05 mark. However, when the

threshold is set to 0.5 all the p-values are less than or equal to 0.05 for the VSM ↔ N-Gram tests. Yet the VSM ↔ Disco tests revealed only two outliers while the rest of the p-values are less than or below 0.05.

In addition to Chi-square independence test, we have used Spearman's rank correlation to further *verify* the linear correlation between the metrics derived from the corpus based technique against the identifier based techniques. Spearman's correlation results showed generally a statistically significant and weak correlation in at least half of the projects and between a moderate to large positive correlation (0.3 - 0.8) in another half of the projects with some outliers (negative correlation coefficients) as shown in Figure 5. However, these negative correlation coefficients are statistically insignificant; the p-values are greater than 0.05 meaning the negative correlation was identified by chance.

Based on the Spearman's correlation coefficient results, the semantic coupling metrics derived from IR techniques based on class identifiers and class corpora do not covary. However, the use of thresholds when testing for their independence shows a significant association at a semantic dissimilarity threshold of 0.5. This is expected as using a higher dissimilarity threshold of 0.5 for the semantic coupling means that only a subset of all pairs of classes will be reported as semantically coupled. Therefore, the Chi-square independence tests only reveal a significant association between identifier and corpora-based IR methods for a subset of classes – classes that are *highly semantically related* (semantic coupling $\geqslant 0.5$).

Based on the overall results, we cannot reject the null hypothesis and fail to reject the alternative hypothesis H_1 for these tests - ***There is an association between the semantic similarity measures of the corpora and identifier based techniques***, as semantic coupling metrics that exploit class identifiers [34] capture different information with respect to semantic coupling metrics using the entire class corpus.

3.1.1 Summary on RQ1 and its results

Similarly to the results derived in our pilot study [2], the Chi-square independence test results indicate the association between the semantic coupling metrics derived when measuring the semantic similarity between OO software classes based on their identifiers and the whole source code corpus. However, this association only applies to classes which as *highly* semantically related (semantic coupling $\geqslant 0.5$). This is an important result for further studies that wish to consider only highly semantically coupled classes, also considering the efficiency of both approaches (corpora and identifiers) in terms of computation time. The fifth column in Table 4 in Appendix A shows that time was saved when we computed the semantic similarity between classes using their identifiers in all but the first project. For example, the *semanticdiscoverytoolkit* project highlighted in Table 4. Extracting the corpus is time consuming as well as computing the semantic coupling metrics compared to using the identifiers alone. Especially for 'large' projects with hundreds of thousands of lines of source code, these results are essential for both researchers and practitioners.

On the other hand, the Spearman's correlation coefficient results confirm that the semantic coupling metrics derived from identifier and corpora-based IR techniques do not covary. Therefore, to recap the identifier-based metrics and corpora-based cannot be used interchangeably apart from when considering highly semantically linked classes (semantic coupling $\geqslant 0.5$) .

Notwithstanding, there are cases or software activities for which one sentence similarity measurement technique will be more useful compared to the other two. For example, in a scenario whereby two class identifiers are similar but these classes do not have related comments, the corpora based method will return a low similarity while identifier based methods will return a high semantic similarity metric. For example, considering the class identifiers *Geocoder-Geometry* and *GeocoderIT* in the *geocoder-java* OSS project. The following metrics are returned by the corpora (VSM), N-gram and Disco techniques respectively: 0.0, 0.5 and 0.7. To make use of identifier based techniques, class identifiers are split into short sentences or phrases before adopting the identifier based techniques. The N-gram technique returns a metric closest to that returned by the corpora based technique in comparison to Disco because Disco relies on the English dictionary and will not scale well on non-English terms. For example, considering the class identifiers *Data* and *AnzeigeSpielfield* in the *4-connect* OSS project. The following metrics are returned by the corpora, n-gram and disco techniques respectively: 0.1, 0.1 and -1.0. At 0.5. The Disco technique compares words based on the similarities of their synonyms while the N-Gram technique is based on the edit distance and shared sub-strings of length n between sentences and has been widely used in the literature on text analysis [38]. We have used n-grams of size 4 in this because research in the area of text mining [38,43] has shown that n=4 maximizes precision when analyzing words or terms in several languages including English, French, German, Italian and Swedish. To add to that, long lengths of n increase lexicon size, will not represent short words well and processing N-grams sizes larger than 10 is very slow [38].

While identifier based techniques are more efficient when measuring semantic coupling between classes in terms of computation time, the corpus based technique is useful when recovering traceability links between source code and design documents [41, 71]. Identifier based techniques are unable to extract the meaning or semantics of the documentation and source code to produce similarity measures that can be used to identify traceability links. This is because the identifier of a design document will be too vague and will likely be unrelated to a number of class identifiers. But when parts of the documents are parsed and compared with the terms embedded within the comments and source code of classes, then parts of design documents can be linked to classes in an OO software. Traceability is particularly useful when a developer is trying to comprehend someone else's code and following any provided documentation as is usually required during maintenance and evolution. This is usually done manually and can be time consuming (especially with large systems consisting of millions of lines of code) without tools that can automatically recover traceability links between source code and documentation.

3.2 RQ2. *Is there a «linear» relationship between semantic and logical coupling?*

In order to answer RQ2, it is necessary for us to compute the Spearman's rank correlation (ρ) between the strengths of the logical and semantic coupling between related class pairs in the studied projects. The strength of the logical coupling is measured in terms of the confidence metrics of identified association rules or frequently co-changed class pairs. Similarly to the confidence metric for logical coupling, the semantic coupling metric range between 0 and 1.

The measurement of how loosely or closely two classes are semantically coupled is based on the corpora-based method (VSM), having identified that identifier-based metrics and corpora-based metrics do not share a linear relationship or covary in Section 3.1. Answering RQ2 will shed more light on whether or not the strengths of the semantic and logical coupling of OO software classes covary. For instance, if they do covary, such statistical results will enable the prediction of the co-change frequency of class pairs based on the strength of their semantic coupling. To recap, the linear relationship between both semantic and logical coupling metrics is investigated using the Spearman's rank correlation coefficient in this section and the results are now presented.

The charts in Figure 6 show the correlation results including the p-values obtained. While Figure 7 further gives a clearer picture of the distribution of the p-values. Similarly to the correlation tests explained in Section 3.1, we reject the null hypothesis at the 95% confidence level with only a 5% error margin for the Spearman's rank correlation.

3.2.1 Summary on RQ2 and its results

Figure 6 shows that there is no substantial evidence to infer a particular type of correlation (+ve or -ve) exists between semantic and logical coupling strengths. There is a positive correlation in some projects, while a negative correlation in others. The p-values in 7 further show that the correlations might have been identified by chance and are not statistically significant. This is because most of the p-values are above 0.05 except for only a very few out of the sample of studied projects.

Therefore, due to the lack of any considerable evidence to suggest that there is a correlation between semantic and logical coupling strengths or related OO software classes, we fail to reject the null hypothesis (H_0) for RQ2 presented in Table 1: ***No linear relationship between the strengths of logical and semantic dependencies***.

In summary, to answer RQ2 we have computed the linear correlation between the strengths of the semantic and logical coupling class pairs. We have used the semantic similarity of class identifiers and the confidence of their co-evolution. The results indicate that these coupling strengths do not covary, so they should be considered independent. A pair of classes with a higher

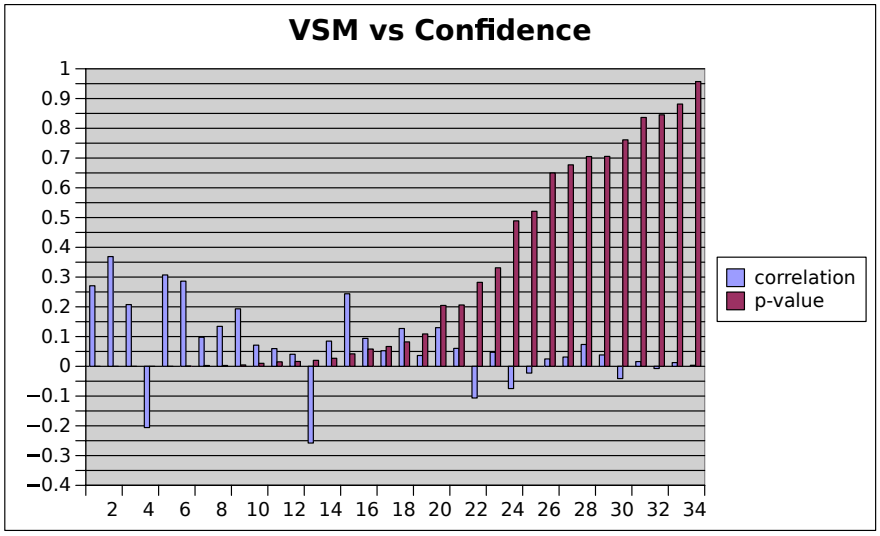

Fig. 6: RQ2- Correlation between VSM based semantic similarity measures and confidence

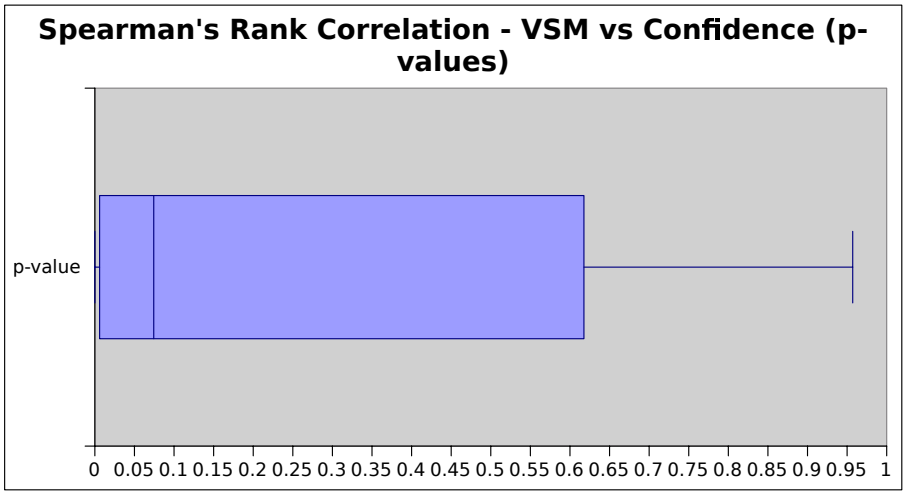

Fig. 7: RQ2- Correlation between VSM based semantic similarity measures and confidence (box-plot distribution of p-values)

co-evolution frequency are not necessarily bound to be linked by a semantic link.

This overall observation has two effects:

1. inferring the co-evolution degree or frequency of class pairs based on the strength of their semantic coupling and vice versa will produce a lot of *false positives*.
2. Using only semantic coupling information to predict co-evolution will produce a prediction model with *low precision*.

Previous research by Abdeen *et al.* [1] has shown that combining semantic and structural coupling information when predicting change impact sets outperforms using either of them individually. However, semantic coupling metrics produced better recall values compared to structural coupling metrics. Research has also shown that the lack of a linear correlation does not imply a lack of causation [50]. In RQ3, we investigate the possibility of a causal relationship between the semantic and logical coupling of classes.

3.3 RQ3. *Is there a «directional» relationship between semantic and logical coupling?*

With the aim of contributing to the interplay between semantic coupling and logical coupling we went a step further to empirically investigate the presence or absence of a (bi-)*directional* relationship between these types of software dependencies. In Section 3.1, we identified that identifier and corpora-based semantic coupling metrics do not covary. Consequentially, similarly to Section 3.2 in this section the semantic coupling metric is calculated using the corpora technique (VSM).

In order to answer RQ3, it is imperative to gain an understanding of the overlapping or intersection of the logical and semantic class dependencies *per* project. The intersection set *per* project is defined as *the proportion of class pairs linked both logically and semantically*. Classes linked logically have either been co-changed once or more while classes linked semantically share are all class pairs excluding those without any semantic similarity whatsoever. The intersection set of class pairs is represented by the shaded area in Figure 3. Depending on the size of the two sets, the Venn diagram could be far from symmetric.

Equations 1 and 2 are at the core of RQ3. Two formulas are presented: the Co-changed Semantic Dependencies (CSD, measured in percentages) and Semantic Logical Dependencies (SLD, also a percentage). These two formulas are used as a measure of the class dependencies that belong to the intersection set (both logically and semantically related classes). The CSD(%) represents co-changed semantic dependencies, these are class pairs that share a semantic and modification relationship (frequently co-changed). The SLD(%) represents classes that are logically or change related and also share a semantic relationship. Some classes might only share either a semantic relationship only or a logical relationship only and these classes do not belong to the intersection set.

$$CSD(\%) = \frac{Semantic \cap Logical}{Semantic} \tag{1}$$

$$SLD(\%) = \frac{Semantic \cap Logical}{Logical} \quad (2)$$

Figures 8 shows two summary plots with the CSD and SLD proportion extracted from the studied sample of OO software projects:

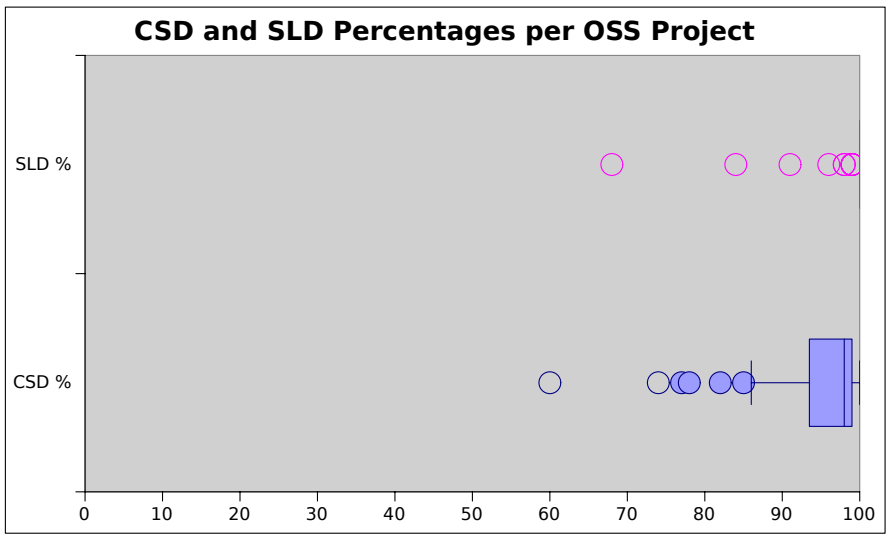

Fig. 8: CSD and SLD Percentages per OSS Project (KEY: CSD = Co-changed Semantic Dependencies; SLD = Semantic Logical Dependencies)

While the proportion of co-changed semantic dependencies (CSD) is high ($\geqslant 70\%$), the proportion of semantic logical dependencies (SLD) tends to also be high. Table 5 in Appendix B reveal for each project the number of distinct semantic dependencies in the third field, the number of distinct logical dependencies in the fourth field, the number of dependencies in the intersection set – pairs of classes that co-change and are semantically related, the percentage of semantic dependencies in the intersection set shown in the sixth field (see equation 1), while the last field shows the percentage of logical dependencies in the intersection set (see equation 2).

Table 5 is sorted by the project IDs and names for readability. The table shows that there is a directional connection between co-change and semantic coupling. When classes contain terms with similar meanings (i.e., the classes are semantically related) they require modifications at the same time. This also holds in the opposite direction.

From Table 5 in Appendix B, we know for instance, that the project with ID=56 has a proportion of 60% of its semantic class dependencies including logical dependencies (as shown in the 6^{th} column). On the other hand, all of the pairs that co-changed include semantic dependencies, in the same project.

This is a recurring pattern: in all of the projects as shown in Table 5, we have evidence to indicate that very often, semantically related classes involve logically related classes. In 17 of these projects, **all** the semantic dependencies are reflected into logical dependencies. In both Venn diagrams (left and right) in Figure 9, the smaller circle represents the set of semantic dependencies while the larger circle represents the set of logical dependencies. Using the Venn diagram on the left (weighted) in Figure 9, all the semantically coupled pairs of classes in the *alleywayreinvented* project (project ID = 12) need also co-changes. On the flip side, not all the pairs that co-change are semantically coupled.

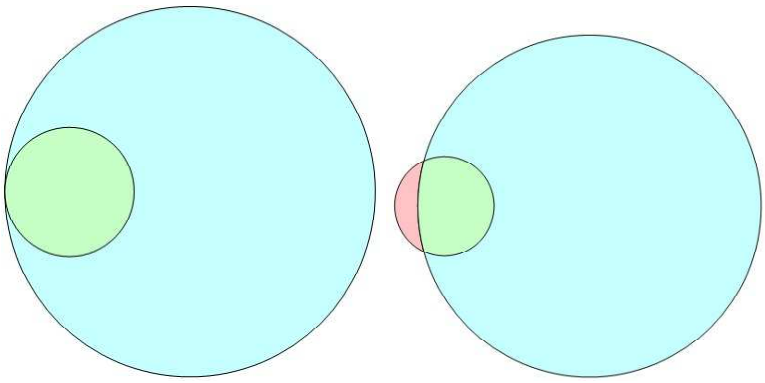

Fig. 9: Venn Diagrams (weighted) showing the two sets of coupling in two scenarios: project ID=12 (left) and project ID=69 (right)

The second most common scenario identified in the results is illustrated using the Venn diagram in Figure 9 (right), showing the *guitarjava* project (project ID = 69). A subset of pairs of semantically coupled classes do not need co-change, while the majority of the others still do. Again, in this project most of its other co-changes are not conducive of semantic links.

3.3.1 Summary on RQ3 and its results

The results mentioned above are illustrated with two box-plots each in Figure 8. The figure shows the distribution of class pairs belonging to the intersection set (classes with both semantic and logical dependencies; see equations 1 and 2). The results indicate a bi-directional relationship between semantic relationships and co-change, as in Figure 8 where both distributions are relatively high in the overall sample of studied OSS projects. Therefore, we reject the null hypothesis (H_0) for RQ3 presented in Table 1 and fail to reject the alternative hypothesis: ***There is a directional relationship between the semantic and logical dependencies among OO software classes***.

In summary, after identifying the lack of a linear relationship between two identifier based techniques in relation to a corpora based information retrieval (IR) technique in semantic coupling measurement (in Section 3.1), we went further to investigate whether there is a linear relationship between the strengths of semantic and logical dependencies of OO software classes. Results presented in Section 3.2 revealed the absence of a linear relationship between the strengths or degrees of the two software dependency types (semantic and logical) at the file or class level. Lastly, as motivated by previous research by Yu [75] and Figure 2 where it has been shown that structural coupling of classes leads to their co-change, we wanted to identify where there was a bi-directional relationship between semantic and logical dependencies. Other results in Section 3.3 revealed the presence of a bi-directional link from semantic to change dependency (semantic $\leftrightarrow$ logical coupling).

4 Discussion

In Section 3, we presented the results of a three-fold empirical study on the interplay between semantic and logical coupling among classes in OO systems. Previous studies have shown that a number of coupling measures, related to aggregation and invocation coupling, are related to a higher probability of common changes. This indicates that these coupling measures should be good indicators of ripple effects and are used as such in a decision model for ranking classes according to their probability to contain ripple effects associated with given change requests [9, 62, 70]. According to Briand *et al.* [9], it is also clear that a substantial number of ripple effects are not covered by the selected highly coupled classes. Thus, such models can be used to focus dependency analysis and help reduce the impact analysis effort. Nevertheless, other important dependencies are clearly not measured or accounted for, and may not be measurable from code alone.

The main findings from the analysis carried out in this study include:

RQ1 **Identifiers vs Corpora** – Firstly, identifier-based techniques (N-gram and Disco) yield **similar** results to analysing the whole corpora of software classes **only** for highly semantically related classes (semantic coupling $\geqslant$ 0.5) and as such cannot always be used interchangeably when computing semantic coupling. Secondly, N-gram and Disco are much more computationally **efficient** than corpora-based techniques, time-wise. Finally, the N-gram technique is more efficient than the Disco technique, precision-wise: the latter is heavily dependent on the English dictionary, as it considers words with similar English synonyms as semantically related. This study has shown that over 50% of the software projects analyzed do not contain classes with only English identifiers, therefore the Disco technique will produce a lot of false negatives.

RQ2 **Strengths of coupling** – There is no linear correlation between the degree or strengths of the semantic similarity between classes and the frequency

of their co-change. Statistical results prove that not all highly semantically related class pairs will require frequent co-changes.

RQ3 **Direction/Causality of Coupling** – There is a large overlapping between semantic and logical (change) class dependencies. If two classes are semantically coupled, there is a high chance that they will co-evolve in the future. However, from RQ2 we have shown that the degree of these dependency types do not show a linear correlation.

The last result is particularly important: for example, two (semantically similar) class pairs $A \leftrightarrow \hat{A}$ and $B \leftrightarrow \hat{B}$ could share a semantic similarity of 0.7, but not the same degree of co-change: the pair $A \leftrightarrow \hat{A}$ could change much more often than $B \leftrightarrow \hat{B}$. Even so, what we have shown is that it is highly likely that the pairs $A \leftrightarrow \hat{A}$ and $B \leftrightarrow \hat{B}$ will co-change at least once or more.

In addition, Spearman's rank correlation coefficient only assesses linear relationships but some relationships can be curvillinear [3]. Earlier research has shown that lack of correlation does not imply lack of causation [32,66,72].

Other researchers have emphasized the need to study the interplay between semantic and logical coupling in OO software as well as the interplay between structural and semantic coupling [46, 47]. It is noteworthy that this study has presented three novel results in (Sections 3.1, 3.2 and 3.3) the software dependency and maintenance domain. These results will be useful and can guide software developers when building software maintenance tools for change impact analysis (CIA).

Studies on the relationship and interplay between structural and logical coupling have shown that a majority of co-evolving classes are not structurally linked. According to Geipel and Schweitzer [25], this indirectly means that any model that tries to infer structural coupling from logical coupling or co-evolution will produce a lot of false positives. On the other hand, using the structural coupling information between pairs of classes to infer their future co-change is a more realistic objective [46]. Differently from previous studies, this study has shown that over 70% of semantically related classes will usually co-change and the same proportion of change related classes will usually share a semantic relationship. Reflecting back to Section 1, these results are backed by the argument by Bavota *et al.* [6]: "the peculiarity of the semantic coupling measure allows it to better estimate the mental model of developers than the other coupling measures. This is because, in several cases, the interactions between classes are encapsulated in the source code vocabulary". However we cannot firmly assert that using the semantic coupling metrics between classes to infer the strength of their co-change is a realistic objective as our empirical study did not show a linear relationship between the strengths of semantic and logical coupling. But we believe that using a combination of structural and semantic information to predict co-change patterns [1] might be a more feasible objective.

5 Threats to Validity

In this section we present the threats to validity of this study, dividing them in *external*, *internal* and *construct* threats.

External validity This paper presents the results of an empirical analysis that should be applicable to all open-source projects. We cannot generalise our findings on any other sample of projects, or from any other repository but the lessons learned from this study can be instructive and transferred to similar studies carried out by others. Nonetheless, in order to make the findings from our study more generalisable and representative of open-source projects, we have carried out our analysis on a random sample of projects, with different sizes.

Internal validity Our selection of the semantic dissimilarity threshold when investigating the association between the corpora-based technique and the identifier-based IR techniques for semantic coupling measurement is based on dissimilarity thresholds used in previous text mining and information retrieval studies. Therefore, to prevent any form of bias during the Chi-squared independence tests we have used three different values (t = 0.1, 0.2 and 0.5) and compared results. This is because different thresholds will reveal different results as shown in Section 3.1.

For measuring logical coupling, we have used the `arules` package in the R statistical environment [30]. We set the confidence threshold to 0.01 and this might have affected the results. While this is a low threshold, it results in a higher recall [16] (i.e., identified a larger set of frequently co-changing classes). We further conducted a manual check on the returned association rules in the smaller projects to ensure that class pairs returned by the `arules` tool actually co-changed and to validate its accuracy. We also adopted 2x2 contingency tables when investigating the association between the identifier and corpora-based IR techniques using the Chi-squared independence test. This test results in false positives when one or more cells have no observations but this was not the case in our data set as each project had at least one class pair in each contingency table cell.

For parsing the corpora of classes and computing semantic coupling, we have adopted the vector space model (VSM) IR technique and we acknowledge that this can have an impact on our results. We also acknowledge that other text document comparison techniques exist, one of which is the latent semantic analysis (LSI); an extension of VSM [5] used in other domains apart form software engineering.

LSI uses an approach called singular vector decomposition (SVD) to reduce text documents (dimensionality or noise reduction to reflect semantic associations between words - latent semantic space) [14] by representing synonyms with topics in a latent semantic space before computing document similarity.

[14] https://matpalm.com/lsa_via_svd/intro.html

```
1 ...
2  public class setzeStein {
3 ...
4     dbConnector DBConnect = new dbConnector();
5 ...
6 // DBConnect.insertMove(data.getAktSpieler(), eingabespalte);
7 ...
```

Listing 1: SetzeStein.java

As such, the dimensionality of a corpus is the number of distinct topics represented in it. Dimentionality reduction allows LSI to index or compare text documents based on topics/concepts instead of similar words. This means that LSI requires the use of fine tuned models.

However, in the context of semantic coupling the reduction of words in documents by grouping them into topics is time consuming as well as prone to low accuracy especially in cases where software teams or comments are multi-lingual (i.e., software built by developers who speak different languages and write comments in their native language). An example of this scenario is the two classes in Listings 1 and 2 both from the same software project (*4-connect*). Line 7 of Listing 2 contains English words while other comments in the class contain non-English words. The identifier of the class in Listing 1 is also not an English word. In such a case, it becomes imperative to translate words from one language to another before building a textual model for LSI to rely upon.

Prior research has demonstrated that text similarity is based on the notion that the meaning of a sentence is made up of not only the meanings of its individual words, but also the structural way the words are combined [48]. Measuring the similarity between non-English documents based on models trained with the Wikipedia corpus for example will yield a low accuracy in the software domain. In Section 3.1.1, we have demonstrated using the Disco word synonym technique that text similarity methods based on the English dictionary does not perform well in the software domain.

In a different study [5] on software traceability link recovery, VSM outperformed LSI. But in an earlier study on the same topic the authors preferred the use of LSI over VSM. According to Marcus *et al.* [41] "our main assumption is that developers use the same natural language (e.g., English, Romanian, etc.) in writing internal documentation and external documentation". However, our examples and results have shown that the reverse is the case. A feasible research topic for future work will be to investigate or build techniques for the measurement of semantic coupling between *multi-lingual* OO software classes. Lastly, the performance of LSI depends on the contents of the documents used to build the model. As such, LSI is also not scalable when new documents, not analyzed during model building are parsed using pre-built models, as the concepts in such documents is not captured in the model which also has to be re-built.

```
...
//Benutzt Methode insert um den Array players in Tabelle
    tbl_player zu speichern
insert("tbl_player", players);
}
public void insertMove(String Player, int Spalte) throws
Exception {
//fullt Array moves
String[] moves = new String[]{
    String.valueOf("(SELECT NEXT VALUE FOR seq_move FROM
        tbl_id)"),
    String.valueOf(Spalte),
    String.valueOf(Player)
    //String.valueOf(move.getSet()),
};
// Benutzt Methode insert um den Array moves in Tabelle
    tbl_move zu speichern
insert("tbl_move", moves);
...
```

Listing 2: DbConnector.java

Construct validity The scope of our sample of projects was limited to open-source software projects written in the Java programming language (object-oriented), thus we encourage investigating projects written in other programming languages and non-object-oriented software projects. The study was also conducted at the class level of granularity. This is because overall,the measurement of semantic coupling is more affected by the difference in granularity than logical or evolutionary coupling. A previous study has shown that for the semantic dependencies, going from the coarse granularity of classes to the finer granularity of methods results in the reduction of the sizes of the documents [34]. The documents are reduced in terms (and frequency). That is, a corpus for a class is typically much "bigger" than a corpus for a method [34]. For logical coupling some commits do not contain changes made to methods while some do not contain changes made to classes, so there is not way to map changes made to classes and methods. This informs the choice of the class level of granularity.

6 Related Work

Structural and logical (evolutionary) dependencies are at the core of software engineering. However, the study of semantic coupling is still evolving and relatively new compared to the number of studies undertaken on the structural and logical coupling of software classes. In the following section, we summarise the main results of related work on both aspects separately and jointly.

6.1 Semantic Coupling

Poshyvanyk and Marcus [52] defined a coupling metric for classes based on textual information extracted from source code identifiers and comments. Their conceptual coupling metric, CoCC (Conceptual Coupling of Classes), captures a new dimension of coupling not addressed by structural or dynamic measures. More recently, Ujhazi *et al.* [64] 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 [22]. Kuhn *et al.* [39] proposed the use of IR techniques to exploit linguistic information found in source code, such as identifiers (i.e., class or method) names and comments. Revelle *et al.* [57] define new feature coupling metrics based on structural and textual source code information.

Kagdi *et al.* [34] in their study on integrating conceptual and logical coupling metrics for change impact analysis suggest that measurement of conceptual metrics is better employed at the class level than at the method level. A corpus for a class is typically much "bigger" than a corpus for a method [34]. This informs our choice of conducting this study at the class level of granularity. In most of these studies, the semantic similarity between software classes using the LSI or VSM approach was adopted.

6.2 Logical Coupling

In comparison to the broad research on structural coupling, the study of logical coupling, evolutionary or change dependencies [47, 75, 78] only began a few years ago because of the advances in data mining techniques [75] used to extract co-evolution data. However, despite its short history, there have been several interesting studies published with promising results. Xia [73] argued that the most widely used design metrics for the inter-module relation were based on information flow rather than the coupling or cohesion criteria, and proposed a metric to compute coupling complexity of modules of a system. Ying *et al.* [74] proposed an approach for predicting source code changes by mining the change history of software systems. Zimmermann *et al.* [78] applied data mining to version histories to guide programmers on related changes using the idea that "Programmers who changed these functions also changed...." [78]. Given a set of existing changes, the mined association rules 1) suggest and predict likely further changes, 2) show up item coupling that is undetectable by program analysis, and 3) can prevent errors due to incomplete changes.

6.3 The Link Between Semantic and Logical Coupling

Recent studies [21,25,46,47] have shown that it is possible that structural and logical coupling are caused by other types of relationships (e.g., conceptual dependencies); some logically coupled classes were without structural coupling links between them and *vice versa*.

Kagdi *et al.* [34,35] demonstrated that finer granularity decreased the accuracy of all approaches; however, it does not prevent the combination of the two from outperforming the standalone techniques. That is, the gain acquired by combining conceptual and evolutionary coupling exists regardless of the granularity (file-level and method-level) considered in the study. Additionally, they did not analyze the relationship between the coupling types and their information retrieval technique (LSI) or take into consideration "common English words and programming language keywords". Since this study has shown that not all OO software contain only English words, this could have had an effect on the accuracy of their findings.

Related work shows that only a few studies have combined semantic and logical coupling to support software engineering activities and none have studied their correlation and interplay. Our study fills this gap by examining the interplay between semantic and logical coupling in OO software systems. Bavota *et al.* [6] investigated how class coupling as measured by dynamic, logical, structural and semantic coupling aligned with developer perception of coupling. They concluded that coupling was an important quality attribute of a software system which could not be captured by structural information such as method calls. More sophisticated approaches, and different source of information need to be analyzed to provide a better evaluation of developer perception of coupling. To this end, semantic coupling seems to reflect a developer's mental model when identifying interaction between entities.

Poshyvanyk *et al.* [53] propose new semantic coupling metrics based on the degree to which the identifiers and comments from different classes relate to each other at the class and method level of granularity. They suggest that their metrics capture new dimensions in software dependency measurement, compared with existing structural dependency metrics. For example, indicating change ripple effects better, compared to existing structural coupling measures and the new metrics can be used to rank classes in the course of impact analysis in large OO systems.

7 Conclusions and Future Work

We have presented two methods of measuring the semantic coupling of software classes using only their identifiers. We further compared each of these methods to measuring the semantic coupling of classes using their corpora. Results showed that using only the class identifiers is a more efficient approach but not in all cases (only when considering highly semantically similar classes). As such, identifier and corpora-based IR methods for computing semantic coupling cannot be used interchangeably in all studies. For projects with hundreds of thousands of lines of code, the extraction of text from all the classes to build their corpora can be time consuming and complex. Hence, the results derived from this study on class identifier-based semantic coupling metrics have some significance in the software dependency and maintenance domain and can be further explored.

On the interplay between semantic and logical dependencies, in 79 object-oriented and open-source software projects we could not detect a linear relationship between the strengths of semantic and logical dependencies. However, we identified a bi-directional link between semantic to logical dependencies. In other words, over 70% of classes that are semantically related will usually co-evolve and classes that are change related will usually share some degree of semantic coupling. Based on empirical results derived from a significant number of software projects, we conclude that identifying more efficient methods of semantic coupling computation as well as a directional relationship between semantic and change dependencies can help to improve CIA techniques that integrate semantic coupling information.

This will speed up the process of revealing 'hidden dependencies' not captured by source code dependencies. Our results can also guide software developers and researchers in developing future generations of tools for supporting program comprehension. Future work will involve comparing the change impact set identified when adopting identifier-based methods for semantic coupling measurement to class corpora-based methods based on precision and recall. Other future work will focus on the interplay between structural and semantic coupling as well as the measurement of semantic coupling between classes in OO software projects built by multi-lingual developers with comments written in several natural languages (e.g., English, French, German, etc.).

References

1. Abdeen, H., Bali, K., Sahraoui, H., Dufour, B.: Learning dependency-based change impact predictors using independent change histories. Information and Software Technology **67**, 220–235 (2015)
2. Ajienka, N., Capiluppi, A.: Semantic coupling between classes: Corpora or identifiers? In: Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, p. 40. ACM (2016)
3. Barnett, M.L., Salomon, R.M.: Beyond dichotomy: The curvilinear relationship between social responsibility and financial performance. Strategic Management Journal **27**(11), 1101–1122 (2006)
4. Bavota, G., De Lucia, A., Marcus, A., Oliveto, R.: A two-step technique for extract class refactoring. In: Proceedings of the IEEE/ACM international conference on Automated software engineering, pp. 151–154. ACM (2010)
5. Bavota, G., De Lucia, A., Oliveto, R., Panichella, A., Ricci, F., Tortora, G.: The role of artefact corpus in lsi-based traceability recovery. In: Traceability in Emerging Forms of Software Engineering (TEFSE), 2013 International Workshop on, pp. 83–89. IEEE (2013)
6. Bavota, G., Dit, B., Oliveto, R., Di Penta, M., Poshyvanyk, D., De Lucia, A.: An empirical study on the developers' perception of software coupling. In: Proceedings of the 2013 International Conference on Software Engineering, pp. 692–701. IEEE Press (2013)
7. Bavota, G., Gethers, M., Oliveto, R., Poshyvanyk, D., Lucia, A.d.: Improving software modularization via automated analysis of latent topics and dependencies. ACM Transactions on Software Engineering and Methodology (TOSEM) **23**(1), 4 (2014)
8. Bavota, G., Oliveto, R., Gethers, M., Poshyvanyk, D., De Lucia, A.: Methodbook: Recommending move method refactorings via relational topic models. Software Engineering, IEEE Transactions on **40**(7), 671–694 (2014)

9. Briand, L.C., Wust, J., Lounis, H.: Using coupling measurement for impact analysis in object-oriented systems. In: Software Maintenance, 1999.(ICSM'99) Proceedings. IEEE International Conference on, pp. 475–482. IEEE (1999)
10. Corley, C., Mihalcea, R.: Measuring the semantic similarity of texts. In: Proceedings of the ACL workshop on empirical modeling of semantic equivalence and entailment, pp. 13–18. Association for Computational Linguistics (2005)
11. Coster, W., Kauchak, D.: Learning to simplify sentences using wikipedia. In: Proceedings of the workshop on monolingual text-to-text generation, pp. 1–9. Association for Computational Linguistics (2011)
12. Crowston, K., Wei, K., Li, Q., Eseryel, U.Y., Howison, J.: Coordination of free/libre and open source software development (2005)
13. Cruz, D., Wieland, T., Ziegler, A.: Evaluation criteria for free/open source software products based on project analysis. Software Process: Improvement and Practice **11**(2), 107–122 (2006)
14. D'Ambros, M., Lanza, M., Lungu, M.: The evolution radar: Visualizing integrated logical coupling information. In: Proceedings of the 2006 international workshop on Mining software repositories, pp. 26–32. ACM (2006)
15. D'Ambros, M., Lanza, M., Robbes, R.: On the relationship between change coupling and software defects. In: Reverse Engineering, 2009. WCRE'09. 16th Working Conference on, pp. 135–144. IEEE (2009)
16. Dasseni, E., Verykios, V.S., Elmagarmid, A.K., Bertino, E.: Hiding association rules by using confidence and support. In: International Workshop on Information Hiding, pp. 369–383. Springer (2001)
17. Despotakis, D., Thakker, D., Lau, L., Dimitrova, V.: Capturing the semantics of individual viewpoints on social signals in interpersonal communication. Semantic Web Journal, Special Issue on Personal and Social Semantic Web ((Under review)) (2011)
18. Didele, V.: Statistical causality (2005)
19. Erkan, G., Radev, D.R.: Lexrank: Graph-based lexical centrality as salience in text summarization. Journal of Artificial Intelligence Research **22**, 457–479 (2004)
20. Field, A.P.: Discovering statistics using SPSS: and sex and drugs and rock 'n' roll, 3rd edn. SAGE, London;Los Angeles; (2009)
21. Fluri, B., Gall, H.C., Pinzger, M.: Fine-grained analysis of change couplings. In: Source Code Analysis and Manipulation, 2005. Fifth IEEE International Workshop on, pp. 66–74. IEEE (2005)
22. Fluri, B., Würsch, M., Gall, H.C.: 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, pp. 70–79. IEEE (2007)
23. Gall, H., Hajek, K., Jazayeri, M.: Detection of logical coupling based on product release history. In: Software Maintenance, 1998. Proceedings., International Conference on, pp. 190–198. IEEE (1998)
24. Gall, H., Jazayeri, M., Krajewski, J.: Cvs release history data for detecting logical couplings. In: Software Evolution, 2003. Proceedings. Sixth International Workshop on Principles of, pp. 13–23. IEEE (2003)
25. Geipel, M.M., Schweitzer, F.: The link between dependency and cochange: empirical evidence. Software Engineering, IEEE Transactions on **38**(6), 1432–1444 (2012)
26. Gethers, M., Aryani, A., Poshyvanyk, D.: Combining conceptual and domain-based couplings to detect database and code dependencies. In: Source Code Analysis and Manipulation (SCAM), 2012 IEEE 12th International Working Conference on, pp. 144–153. IEEE (2012)
27. Gousios, G., Pinzger, M., Deursen, A.v.: An exploratory study of the pull-based software development model. In: Proceedings of the 36th International Conference on Software Engineering, pp. 345–355. ACM (2014)
28. Haefliger, S., Von Krogh, G., Spaeth, S.: Code reuse in open source software. Management Science **54**(1), 180–193 (2008)
29. Hahsler, M., Gruen, B., Hornik, K., Hahsler, M.M.: The arules package. methods **15**, 1 (2006)
30. Hahsler, M., Grün, B., Hornik, K.: Introduction to arules–mining association rules and frequent item sets. SIGKDD Explor **2**(4) (2007)

31. Hahsler, M., Hornik, K.: Building on the arules infrastructure for analyzing transaction data with r. In: Advances in Data Analysis, pp. 449–456. Springer (2007)
32. Howard, G.S., Maxwell, S.E.: Correlation between student satisfaction and grades: A case of mistaken causation? Journal of Educational Psychology **72**(6), 810 (1980)
33. Kagdi, H., Collard, M.L., Maletic, J.I.: A survey and taxonomy of approaches for mining software repositories in the context of software evolution. Journal of Software Maintenance and Evolution: Research and Practice **19**(2), 77–131 (2007)
34. Kagdi, H., Gethers, M., Poshyvanyk, D.: Integrating conceptual and logical couplings for change impact analysis in software. Empirical Software Engineering **18**(5), 933–969 (2013)
35. Kagdi, H., Gethers, M., Poshyvanyk, D., Collard, M.L.: Blending conceptual and evolutionary couplings to support change impact analysis in source code. In: Reverse Engineering (WCRE), 2010 17th Working Conference on, pp. 119–128. IEEE (2010)
36. Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D.M., Damian, D.: An in-depth study of the promises and perils of mining github. Empirical Software Engineering **21**(5), 2035–2071 (2016)
37. Kenett, R., Salini, S.: Relative linkage disequilibrium: a new measure for association rules. In: Industrial Conference on Data Mining, pp. 189–199. Springer (2008)
38. Kešelj, V., Peng, F., Cercone, N., Thomas, C.: N-gram-based author profiles for authorship attribution. In: Proceedings of the conference pacific association for computational linguistics, PACLING, vol. 3, pp. 255–264 (2003)
39. Kuhn, A., Ducasse, S., Gírba, T.: Semantic clustering: Identifying topics in source code. Information and Software Technology **49**(3), 230–243 (2007)
40. Lozano, A., Noguera, C., Jonckers, V.: Explaining why methods change together. In: SCAM, pp. 185–194 (2014)
41. Marcus, A., Maletic, J.I., Sergeyev, A.: Recovery of traceability links between software documentation and source code. International Journal of Software Engineering and Knowledge Engineering **15**(05), 811–836 (2005)
42. Marcus, A., Poshyvanyk, D.: The conceptual cohesion of classes. In: 21st IEEE International Conference on Software Maintenance (ICSM'05), pp. 133–142. IEEE (2005)
43. Mcnamee, P., Mayfield, J.: Character n-gram tokenization for european language text retrieval. Information retrieval **7**(1-2), 73–97 (2004)
44. Midha, V., Palvia, P.: Factors affecting the success of open source software. Journal of Systems and Software **85**(4), 895–905 (2012)
45. Oliva, G., Gerosa, M.A.: Ivar: A conceptual framework for dependency management. In: IX Workshop de Manutenção de Software Moderna (WMWSM 2012) (2012)
46. Oliva, G.A., Gerosa, M.: 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)
47. Oliva, G.A., Gerosa, M.A.: On the interplay between structural and logical dependencies in open-source software. In: Software Engineering (SBES), 2011 25th Brazilian Symposium on, pp. 144–153. IEEE (2011)
48. Oliva, J., Serrano, J.I., del Castillo, M.D., Iglesias, Á.: Symss: A syntax-based measure for short-text semantic similarity. Data & Knowledge Engineering **70**(4), 390–405 (2011)
49. Pagano, R.R.: Understanding statistics in the behavioral sciences, 6th edn. Wadsworth-Thomson Learning, Australia;United Kingdom; (2001)
50. Perdicoúlis, A.: Correlation and causality
51. Petrenko, M., Rajlich, V.: Variable granularity for improving precision of impact analysis. In: Program Comprehension, 2009. ICPC'09. IEEE 17th International Conference on, pp. 10–19. IEEE (2009)
52. Poshyvanyk, D., Marcus, A.: The conceptual coupling metrics for object-oriented systems. In: Software Maintenance, 2006. ICSM'06. 22nd IEEE International Conference on, pp. 469–478. IEEE (2006)
53. Poshyvanyk, D., Marcus, A., Ferenc, R., Gyimóthy, T.: Using information retrieval based coupling measures for impact analysis. Empirical software engineering **14**(1), 5–32 (2009)
54. Prasad, L., Bhadauria, S.S.: How to realization architectural testing model using measurement metrics (2009)

55. Qusef, A., Bavota, G., Oliveto, R., De Lucia, A., Binkley, D.: Scotch: Test-to-code traceability using slicing and conceptual coupling. In: Software Maintenance (ICSM), 2011 27th IEEE International Conference on, pp. 63–72. IEEE (2011)
56. Rainer, A., Gale, S.: Evaluating the quality and quantity of data on open source software projects. In: Procs 1st Int Conf on Open Source Software (2005)
57. Revelle, M., Gethers, M., Poshyvanyk, D.: Using structural and textual information to capture feature coupling in object-oriented software. Empirical software engineering **16**(6), 773–811 (2011)
58. Samoladas, I., Angelis, L., Stamelos, I.: Survival analysis on the duration of open source projects. Information and Software Technology **52**(9), 902–922 (2010)
59. Sarikaya, R., Gravano, A., Gao, Y.: Rapid language model development using external resources for new spoken dialog domains. In: ICASSP (1), pp. 573–576 (2005)
60. Stake, R.E.: The art of case study research. Sage (1995)
61. Subramanyam, R., Krishnan, M.S.: Empirical analysis of CK metrics for object-oriented design complexity: Implications for software defects. Software Engineering, IEEE Transactions on **29**(4), 297–310 (2003)
62. Sun, X., Li, B., Leung, H., Li, B., Zhu, J.: Static change impact analysis techniques: A comparative study. Journal of Systems and Software **109**, 137–149 (2015)
63. Tan, C.L., Sung, S.Y., Yu, Z., Xu, Y.: Text retrieval from document images based on n-gram algorithm. In: PRICAI Workshop on Text and Web Mining, pp. 1–12. Citeseer (2000)
64. Újházi, B., Ferenc, R., Poshyvanyk, D., Gyimóthy, T.: New conceptual coupling and cohesion metrics for object-oriented systems. In: Source Code Analysis and Manipulation (SCAM), 2010 10th IEEE Working Conference on, pp. 33–42. IEEE (2010)
65. Vanciu, R., Rajlich, V.: Hidden dependencies in software systems. In: Software Maintenance (ICSM), 2010 IEEE International Conference on, pp. 1–10. IEEE (2010)
66. Verhulst, B., Eaves, L.J., Hatemi, P.K.: Correlation not causation: The relationship between personality traits and political ideologies. American journal of political science **56**(1), 34–51 (2012)
67. Wiese, I., Kuroda, R., Ré, R., Bulhões, R., Oliva, G., Gerosa, M.: Do historical metrics and developers communication aid to predict change couplings? Latin America Transactions, IEEE (Revista IEEE America Latina) **13**(6), 1979–1988 (2015)
68. Wiese, I.S., Kuroda, R.T., Re, R., Oliva, G.A., Gerosa, M.A.: An empirical study of the relation between strong change coupling and defects using history and social metrics in the apache aries project. In: IFIP International Conference on Open Source Systems, pp. 3–12. Springer (2015)
69. Wiese, I.S., Ré, R., Steinmacher, I., Kuroda, R.T., Oliva, G.A., Treude, C., Gerosa, M.A.: Using contextual information to predict co-changes. Journal of Systems and Software (2016)
70. Wilkie, F.G., Kitchenham, B.A.: Coupling measures and change ripples in c++ application software. Journal of Systems and Software **52**(2), 157–164 (2000)
71. Witte, R., Li, Q., Zhang, Y., Rilling, J.: Text mining and software engineering: an integrated source code and document analysis approach. Iet Software **2**(1), 3–16 (2008)
72. Wright, B.R.E., Caspi, A., Moffitt, T.E., Miech, R.A., Silva, P.A.: Reconsidering the relationship between ses and delinquency: Causation but not correlation. Criminology **37**(1), 175–194 (1999)
73. Xia, F.: Module coupling: A design metric. In: Software Engineering Conference, 1996. Proceedings., 1996 Asia-Pacific, pp. 44–54. IEEE (1996)
74. Ying, A.T., Murphy, G.C., Ng, R., Chu-Carroll, M.C.: Predicting source code changes by mining change history. Software Engineering, IEEE Transactions on **30**(9), 574–586 (2004)
75. Yu, L.: Understanding component co-evolution with a study on linux. Empirical Software Engineering **12**(2), 123–141 (2007)
76. Yu, Z., Rajlich, V.: Hidden dependencies in program comprehension and change propagation. In: Program Comprehension, 2001. IWPC 2001. Proceedings. 9th International Workshop on, pp. 293–299. IEEE (2001)
77. Zimmermann, T., Diehl, S., Zeller, A.: How history justifies system architecture (or not). In: Software Evolution, 2003. Proceedings. Sixth International Workshop on Principles of, pp. 73–83. IEEE (2003)

Table 4: RQ1- Characteristics of the software systems analyzed for semantic coupling measurement comparison

Project	Classes	Class Pairs	LOC (with comments)	Time to Analyze Corpora (mins)	Time to Analyze Identifiers (mins)	Δ mins
4-connect	10	45	1,160	0.003	0.005	<1%
alexo-chess	119	7,021	22,986	1.01	0.07	143%
alto	315	49,455	101,379	20	1	19%
audao	152	11,476	20,347	1.1	0.1	10%
bitlyj	22	231	1,255	0.002	0.002	0%
bluecove	390	75,855	75,237	18	1	17%
daedalum	68	2,278	10,172	0.2	0.01	19%
dbmigrate	7	21	1,337	0.003	0.00005	598%
echo-nest-java-api	36	630	6,903	0.1	0.005	19%
fdelimitedtextutilities	11	55	1,769	0.003	0.001	2%
geocoder-java	27	351	1,732	0.006	0.003	1%
google-voice-java	56	1,540	10,078	0.3	0.02	14%
gp-net-radius	25	300	2,469	0.01	0.002	4%
guitarjava	87	21	18,331	0.5	0.03	16%
jangod	127	8,001	10,789	0.4	0.02	19%
java-chess-web	111	6,105	7,983	0.2	0.04	4%
java-weather-api	35	595	2,041	0.01	0.004	2%
jbal	109	5,886	21,285	2	0.04	49%
jbandwidthlog	13	78	2,472	0.01	0.001	9%
jiopi	22	231	2,260	0.003	0.001	2%
jmemcache	14	91	1,035	0.002	0.001	1%
kryo	52	1,326	6,356	0.1	0.01	9%
migrator-postgresql	29	406	2,282	0.01	0.002	4%
monome-pages	158	12,403	64,942	9	0.08	112%
powermock	673	226,128	73,985	21	3	6%
prettyfaces	229	26,106	26,104	2	0.08	24%
projet-qcm-java	53	1,378	4,661	0.04	0.01	3%
ps3mediaserver	189	17,766	39,816	6	0.05	119%
restfb	75	2775	16,041	0.8	0.06	12%
scikit	109	5,886	18,224	1	0.03	32%
semanticdiscoverytoolkit	1,421	1,008,910	268,564	695	7.2	98%
seoma	280	39,060	37,007	2.4	0.3	7%
sjava-logging	19	171	1,514	0.002	0.001	1%
tabuvrp–study	28	378	2,524	0.01	0.002	4%
usemon	1,090	593,505	219,546	980	4	244%

78. Zimmermann, T., Zeller, A., Weissgerber, P., Diehl, S.: Mining version histories to guide software changes. Software Engineering, IEEE Transactions on **31**(6), 429–445 (2005)

Appendix A Summary of the software systems analyzed for semantic coupling measurement comparison

Appendix B Summary of the Intersection of Semantic and Logical Dependencies

Table 5: RQ3- Intersection of Semantic and Logical Dependencies in the studied 79 OSS Projects. (KEY: Sem. Dep. = Semantic Dependencies; Log. Dep. = Logical Dependencies; CSD = Co-changed Semantic Dependencies; SLD = Semantic Logical Dependencies)

ID	Project	Sem. Dep.	Log. Dep.	Int. Set	CSD (%)	SLD (%)
1	2dtetris	213	166	166	78	100
2	4-connect	55	80	54	98	68
7	ahs-scheduling	144	118	118	82	100
8	aima-java	190694	190432	189812	100	100
10	alexo-chess	9759	9603	9603	98	100
11	algmusic	3867	3812	3812	99	100
12	alleywayreinvented	668	680	668	100	98
13	alto	77600	78481	77505	100	99
14	amock	3508	2969	2969	85	100
18	apjava	202	196	196	97	100
20	appletbomberman	1307	1255	1255	96	100
22	ascrblr	1429	1396	1396	98	100
24	audao	6957	6838	6838	98	100
26	bitlyj	1068	1036	1036	97	100
28	bluecove	63358	63404	63212	100	100
30	castanea	681	624	624	92	100
31	catchnthrow	180	164	164	91	100
41	daedalum	4855	4854	4852	100	100
45	dbmigrate	29	26	26	90	100
51	echo-nest-java-api	1132	1116	1116	99	100
56	fdelimitedtextutilities	57	34	34	60	100
60	fyllgen	14316	14318	14298	100	100
64	geocoder-java	441	379	379	86	100
65	google-voice-java	767	724	694	90	96
66	gorobot	89362	88731	88627	99	100
67	gp-net-radius	537	522	522	97	100
68	guavatools	6923	6899	6879	99	100
69	guitarjava	3412	3681	3351	98	91
71	hobbylinkchecker	35890	35923	35887	100	100
79	jangod	15126	15220	15030	99	99
81	jaque	1228	1065	1065	87	100
84	java-chess-web	2902	2596	2590	89	100
86	java-weather-api	231	220	216	94	98
88	javacoder	104	104	104	100	100
92	javastepbystep	1945	1795	1795	92	100
96	jbal	12903	12986	12884	100	99
97	jbandwidthlog	468	468	468	100	100
99	jease	40346	39842	39842	99	100
103	jeudi-tech-spring	343	310	310	90	100
107	jiopi	553	532	532	96	100
109	jmemcache	97	94	94	97	100
112	jnoob	426	417	417	98	100
113	jothelo	137	148	124	91	84

Continued on next page

Table 5 – *Continued from previous page*

ID	Project	Sem. Dep.	Log. Dep.	Int. Set	CSD (%)	SLD (%)
115	jprg2-assg	336	332	332	99	100
118	jroguedps	6394	6255	6253	98	100
119	jsbe	70	70	70	100	100
122	jtowerdefense	2212	2191	2191	99	100
123	jugile-util	3175	3088	3082	97	100
124	jutf8search	152	152	150	99	99
127	kryo	5465	5372	5370	98	100
130	lemyriapode	10732	10520	10496	98	100
136	migrator-postgresql	478	476	476	100	100
140	mobs	703	672	672	96	100
141	mocrap	100	74	74	74	100
142	monome-pages	10462	10362	10354	99	100
148	ngamejava	1246	1196	1196	96	100
149	object-procedural-bridge	27983	27343	27309	98	100
152	onslaught	5747	5739	5739	100	100
157	p2ploan	10476	10041	10041	96	100
164	powerjava	168	150	148	88	99
165	powermock	105828	105733	105291	99	100
166	prettyfaces	12968	12987	12949	100	100
168	product-center	7708	7220	7220	94	100
169	project-armageddon	88	68	68	77	100
170	projet-qcm-java	937	868	868	93	100
172	ps3mediaserver	29497	29313	29303	99	100
179	restfb	4139	4045	4035	97	100
180	robust-coupe	1833	1648	1648	90	100
183	scikit	11489	10958	10956	95	100
184	semanticdiscoverytoolkit	179777	177962	177928	99	100
185	semweb4j	69401	68309	68009	98	100
186	seoma	17166	16929	16929	99	100
188	simplenamingservice	1662	1593	1593	96	100
189	sjava-logging	408	408	408	100	100
195	squabble	4687	4578	4578	98	100
197	subitizer	188	176	176	94	100
201	tabulasoftmed	58497	58420	58218	100	100
202	tabuvrp–study	491	442	442	90	100
211	usemon	529180	529590	528932	100	100