Managing Hidden Dependencies in OO Software: a Study Based on Open Source Projects

Nemitari Ajienka
Computer Science Department
Brunel University London
Email: nemitari.ajienka@brunel.ac.uk

Andrea Capiluppi
Computer Science Department
Brunel University London
Email: andrea.capiluppi@brunel.ac.uk

Steve Counsell
Computer Science Department
Brunel University London
Email: steve.counsell@brunel.ac.uk

***Abstract*—Dependency-based software change impact analysis is the domain concerned with estimating the sets of artifacts impacted by a change to a related artifact. Research has shown that analysing the various class dependency types independently will never completely reveal the impact sets. Therefore, dependency types are combined to improve the precision of estimated when compared to impact sets.**

Software classes can be linked in different ways; for instance semantically, if their meaning is somewhat related or, structurally, if one class depends on the services of other classes. 'Hidden' dependencies arise when two classes, linked structurally, do not share the same semantic namespace or when semantically dependent classes do not share a structural link.

With the goal of revealing hidden dependencies during change impact analysis, we empirically investigated the relationship between structural and semantic class dependencies in object-oriented software systems.

Results show that (i) semantic and structural links are significantly associated, (ii) the *strengths* of those links do not play a significant role and, (iii) a significant number of dependencies are *hidden*.

We propose two refactoring techniques to deal with hidden dependencies, based on existing design patterns. We plan to investigate them further to assert whether either has the potential for reducing refactoring and testing effort.

I. Introduction

The evolution of software systems is an inevitable process which has to be managed effectively to enhance software quality. Change impact analysis (CIA) [1] is a technique that identifies *impact sets*, i.e., the set of classes that require correction as a result of a change made to a class or artifact [2]. These sets are also known as *ripple effects* and are typically non-local, i.e., changes propagate to different parts of a system. Over the years, software engineering researchers have proposed tools and techniques to predict where those impact sets will arise. Most approaches rely on the software dependency information derived from static and dynamic analysis [1], [3], [4]. Others have focused on the root *causes* of these ripple effects and on predicting co-change of classes [5].

Various coupling measures have been proposed over the years, for example, *dynamic* coupling (i.e., call relationships between classes during program execution), *structural* coupling (i.e., structural relationships between classes, such as the number of method calls between them) [2], *logical* coupling (i.e., the use of historical data to identify classes that always co-change) [6] and *semantic* coupling (i.e., the degree to which identifiers and comments from classes relate to each other) [7]. While structural and semantic dependencies play major roles in software evolution, their relationship has not been empirically investigated in a large-scale empirical study.

Semantic coupling is always symmetrical: if a class A is coupled (i.e., semantically similar) to B, B is also coupled to A (A $\leftrightarrow$ B). Structural coupling is asymmetrical: a structural link from A to B (A $\rightarrow$ B) does not imply a structural link from B to A (B $\rightarrow$ A).

Ideally, semantically coupled classes should be structurally coupled (in any direction). Not semantically coupled classes should also be not-structurally coupled classes. Previous research has demonstrated that this is not the case [3], [4], [8]. While previous studies have focused on combining structural and semantic coupling information with minimal benefits, this study argues that combining them because they are not consistent is an issue in itself. This is because computing multiple coupling types to achieve one software maintenance task can be time consuming especially in large systems composed of millions of lines of code and this forms the core **motivation** for this study. Using Table I, we visualize this motivation, classifying class pairs of OO systems into four groups, based on structural and semantic dependencies.

TABLE I: Established, hidden and weak dependencies

Structural vs Semantic Dependencies		
	semantic	*not semantic*
structural	E	S
not structural	W	x

When pairs of classes are linked both structurally and semantically, we posit that a dependency is established (denoted 'E'). If a structural link is present, but not a semantic one, there could be a strong (denoted 'S') missing dependency. On the contrary, if there is a semantic link, but not the structural one, a weak (denoted 'W') missing dependency is detected. When neither a semantic nor structural link is detected, no dependency (denoted 'x') is established. If the structural and semantic class dependency types are established, the prediction of ripple effects can be run more precisely. On the other hand, when dependencies are hidden or weak, developers will detect a smaller number of dependencies capable of propagating further change. Based on these premises, our work has the

following goals, to be later articulated in formal research questions (Table II) and contributions:

G_1: to investigate whether there is a **statistically significant relationship** between the strengths (or weights) of semantic and structural class dependencies.

G_2: to investigate the **relationship** between structural and semantic coupling, by identifying (i) the proportion of structural links that involve non-semantically related classes (i.e., the *hidden* dependencies [9] from Table I) and (ii) the proportion of established semantic dependencies that involve non-structurally related classes (i.e., the *weak* dependencies);

G_3: to propose **practical approaches** to solve the strong (S) and weak (W) dependencies using established design patterns.

The paper is articulated as follows: Section II provides a brief background on software dependencies. Section III describes the data extraction and analysis with worked examples. The results are presented in Section IV. The implication of these results in software engineering (SE) are discussed in Section V. Section VI explores prior related work and Section VII deals with the threats to validity; finally, Section VIII concludes and points to further work.

II. Background

In this empirical study, structural and semantic dependencies form the basis of our measurements. According to Oliva and Gerosa [11], "*a dependency implies that the semantics of a client is not complete without its suppliers*". Below, we introduce the main concepts around 'structural', 'semantic' and 'hidden' dependencies.

A. Structural Dependencies

Geipel and Schweitzer [5] state that there is a *directed dependency* between classes *X* and *Y* if *X* depends on *Y* in such a way that *X* is not operational without class *Y* [11]. In the case of Java classes, this means that *X* will not compile in the absence of *Y*. Furthermore, the relationships: "class *X* depends on class *Y*" and "class *Y* depend on class *X*" have different effects on software evolution. If *X* depends on *Y*, changes made to *Y* can lead to changes to *X*, but not the other way round. We adapt Yu's [10] representation of directional coupling: a single directional solid arrow from class *X* to class *Y* denotes that class *Y* is directionally coupled to class *X*. This is depicted as ***X*→*Y***, because a change to class *X* can affect class *Y*.

Structural coupling is derived from the number of referring variables and functions of other artifacts. There are several types of relationships among source code entities. The constructs of most programming languages (e.g., C, C++, and Java) can motivate various types of relationships [12]. A method calls another method, a class extends another class, or a class aggregates objects of another class - all of these call relationships create a directional dependency between two classes. These static structural code dependencies are usually adopted when investigating structural dependencies. Similar to Yu [10], in this study, we have adopted the number of references from one class to another as a measure of the strength of their structural coupling.

B. Semantic Dependencies

Semantic coupling captures the degree to which the identifiers and comments from different classes are similar to each other [7]. Thus, it is limited to the underlying meanings of unstructured text in the source code of software entities and how these meanings relate to each other [13]. This relationship can also be quantified, as described in [2].

The benefits of the application of semantic technologies to software maintenance have been emphasized in prior research [14]. These benefits include software comprehension and traceability recovery (connecting parts of software documentation and source code using information retrieval (IR) techniques). According to Poshyvanyk and Marcus [7], semantic coupling metrics can be sued to "augment existing existing coupling metrics in tasks such as change impact analysis as existing measures do not capture all the ripple effects of changes in software. They also have direct application in reverse engineering tasks like re-modularization". Unlike structural coupling, semantic coupling between class identifiers for example, can be computed independent of programming languages [2].

In a pilot study [15], we established that metrics derived from computing the semantic coupling between OO software classes using *only* their identifiers (i.e., the class names) reflect the metrics derived when the whole source code corpus is analyzed. Using the identifiers alone is also more efficient in terms of computation time, especially in cases where the software system under analysis contains thousands of lines of code. Depending on the study, the unstructured text in the source code will obviously provide important information [16], and the identifiers alone will not suffice. It is important to note that unlike structural coupling, the semantic similarity between any two text documents is symmetric: the similarity between m_i and m_j is the same as the similarity between m_j and m_i. Therefore, the values of the semantic similarity between two classes or methods are the same [3] irrespective of their order (`addShape()`, `removeShape()` and `removeShape()`, `addShape()`). Thus, we shall represent semantic coupling with a bi-directional arrow between any class pair (i.e., X ↔ Y).

C. Hidden Dependencies

Comments and identifiers inside source code encode semantic information. Software dependencies between classes can be detected observing the similarities in the semantic information of two or more classes [8]. Such similarities are not detected when using traditional structural dependency analysis [9]. In this study, we refer to *hidden* dependencies as class dependencies that show only structural or semantic links (i.e., the union of the 'S' and 'W' sets from Table I). Hidden dependencies make both software comprehension and maintenance hard [17]. They play an important role because they spread changes among classes and they can be hard to detect. Therefore, developers will miss a significant number of them by relying on source code information alone during change impact analysis for example [18].

TABLE II: Research Goals and Research Questions

Goals	Research Questions	Rationale	Testable Null Hypothesis H_0
$\mathbf{G}_1$	[$\mathbf{RQ}_1$] *Is there a significant linear relationship between the **strengths** of structural and semantic coupling?*	It has been shown that *structural* dependencies exhibit a linear relationship with *logical* dependencies [10]. This RQ deals with the strength of those links: does a **stronger** *structural* link imply a **higher** *semantic* similarity?	[$\mathbf{H}_{0.1}$] No linear relationship between the strengths of structural and semantic class dependencies
$\mathbf{G}_2$	[$\mathbf{RQ}_2$] *Is there a dependency between structural and semantic dependencies in OO software?*	The influence of *semantic* and *structural* coupling upon each other has not been studied, despite the fact that an analysis of only semantic dependencies during CIA will not reveal some structural dependencies and vice-versa [3], [4], [8].	[$\mathbf{H}_{0.2}$] Structural and semantic class dependencies are independent

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. For example, a CIA method based on evolution history is based on past operations and already existing change dependencies [19]. Therefore, it could lead to incorrect or incomplete results notably when new artifacts are introduced in the software. Proof of this is contained in the study by Kagdi *et al.* [3] on CIA. When comparing the estimated and actual change impact sets, the logical and semantic dependency metrics estimated different and incomplete sets of classes that might get impacted by given change requests. The accuracy of each technique was reduced at the method level of granularity and this informs our choice of the class level of granularity. Notably, a union of the identified impact sets from both techniques outperformed their intersection. This is inefficient, as it involves computing dependencies twice (logical and semantic) during CIA.

III. Research Methodology

A. Dataset

Leveraging the FlossMole project [20], we used its latest available data dump[1]. We extracted a random subset of 79 non-trivial Java projects of different domains and sizes (in terms of number of classes) based on certain selection criteria such as number of revisions and developer activity[2]. The repository of each project was downloaded and stored, with its metadata (list of revisions for each class, and for the whole project, name of developers, date and time of changes), using the CVSAnalY set of tools[3,4]. These revisions do not contain files without the .java extension. In summary, the median of the number of revisions *per* project lies between 50 and 100. This is similar for the number of `.java` classes *per* project. As the upper outliers, the *Semantic discovery toolkit*[5]) project has some 1,500 classes, while *Ps3 media server*[6] underwent around 800 revisions.

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

[2]Prior research [21] 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, and improve generalizeabiliy of the study.

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

[4]Installation steps can be found at: https://sites.google.com/site/arnamoyswebsite/Welcome/updates-news/howtoinstallandruncvsanaly2inubuntu1110

[5]https://github.com/git2020agile/semanticdiscoverytoolkit

[6]https://github.com/zrevai/ps3mediaserver

B. Identifying and Measuring Dependencies

a) Structural Dependencies: In this study, the structural coupling of classes (and its strength) is measured by the number of *references* from the *caller* class to the *called* class. Two classes are structurally related if the number of structural references from the caller to called class is > 1. We have extracted the structural coupling metrics from the latest source code snapshot of each project with the SciTools UNDERSTAND command line tool [22]. Geipel and Schweitzer [5] analyzed the link between structural dependency and co-change. In computing correlation between structural coupling and co-change metrics they only took into consideration the latest code snapshot when they extracted structural dependencies. They inferred that structural dependencies between two classes i and j are somewhat stable from the creation of the younger class until the removal of either i or j.

b) Semantic Dependencies: The computation of the semantic similarity between OO software classes in this study is based on class identifiers (e.g., class names). In a previous study, we compared two sentence similarity measurement methods (the N-Gram[7] and DISCO Word synonym[8] categories methods [15]) against a corpus or document cosine similarity based technique (VSM[9,10]) for computing semantic similarity between Java classes. The study was conducted using two OSS projects and three semantic dissimilarity thresholds (t=0.25, 0.5 and 0.75). We identified that measuring the semantic similarity between classes using (only) their identifiers is similar to using the class corpora (at t=0.5). We also identified that the N-Gram identifier-based technique is the better technique for comparing English, as well as non-English terms. The Disco word synonym-based technique is suitable for only English terms. Thus, in this study we adopt t=0.5 as the semantic dissimilarity threshold while using the N-Gram technique alone to extract the semantic coupling metrics from the latest source code snapshot of each studied software project.

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

[8]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

[9]We have implemented the VSM method for automating the corpus based technique. It can be downloaded at: https://github.com/najienka/SemanticSimilarityJava

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

C. Data Analysis

In this section and for replicability, we describe the empirical investigations carried out to answer our research questions outlined in Table II. Once pair-wise semantic and structural dependencies were identified, we built contingency tables *per* project. Using a Shell script, we could parse the data and identify the proportion of class pairs that belonged in each cell. All possible class pairs belong to each cell if they meet the following cell descriptions.

- E - proportion of class pairs with BOTH structural and semantic dependencies;
- W - proportion of class pairs with ONLY semantic dependencies;
- S - proportion of class pairs with ONLY structural dependencies;
- x - proportion of unrelated class pairs.

As examples, we present the proportion of dependencies in each cell in the contingency table (Table III) for two software systems in the sample – *4-Connect* and *GuitarJava*. As a smaller system, *4-Connect* doesn't show any established dependencies. 18 class pairs are linked by structural links, but not a semantic link and are therefore classified as hidden. The majority of class pairs do not have either a structural or semantic link. As a larger system, *GuitarJava* shows 55 established (E) links, 212 strong (S) links and 65 weak (W) links. In Table III (middle and bottom), these projects are analysed on their S and W cells.

TABLE III: Contingency Tables: generic (top) and populated (bottom)

Generic Contingency Table

	Semantic Dep (VSM)	
Structural Dep	E	S
	W	x

Structural vs Semantic Dependencies in 4-Connect

	semantic	*not semantic*
structural	0	18
not structural	0	1

Structural vs Semantic Dependencies in GuitarJava

	semantic	*not semantic*
structural	55	212
not structural	65	18

1) Worked Example 1: Table III (middle) shows that when the classes of *4-Connect* are coupled, they are linked by hidden dependencies. As an example, we consider the structural dependency `DbConnector.java` → `SetzeStein.java`. These two classes are only structurally coupled in the last two revisions, while the project had a total of 42 revisions as at the time of data extraction. The pairs of classes were first linked (3 operational calls from `SetzeStein.java` to `DbConnector.java`) in revision 41. Conceptually, the identifier names are not related and their semantic coupling metric as measured by the N-Gram technique is 0. Analysing the two corpora with VSM gives a similarity metric of 0.03.

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

Listing 1: SetzeStein.java

```
1  ...
2  //Benutzt Methode insert um den Array players
       in Tabelle tbl_player zu speichern
3  insert("tbl_player", players);
4  }
5  public void insertMove(String Player, int
       Spalte) throws
6  Exception {
7  //fullt Array moves
8  String[] moves = new String[]{
9      String.valueOf("(SELECT␣NEXT␣VALUE␣FOR␣
           seq_move␣FROM␣tbl_id)"),
10     String.valueOf(Spalte),
11     String.valueOf(Player)
12     //String.valueOf(move.getSet()),
13 };
14 // Benutzt Methode insert um den Array moves in
       Tabelle tbl_move zu speichern
15 insert("tbl_move", moves);
16 ...
```

Listing 2: DbConnector.java

A further investigation into the latest snapshot of the source code of both classes as shown in Listings 1 and 2 reveals the absence of sufficient comments to describe the method `insertMove()` in `DbConnector.java` (line 5 in Listing 2) being called or referenced by `SetzeStein.java`. Both classes ought to have related comments with similar terms to describe the function. Thus, using the vector space model approach [15] to compute their semantic coupling metric with their corpora also echoes the metrics derived by using the identifier-based technique. Line 6 of Listing 1 contains the call to the function `insertMove()`, that was also changed to a comment, to imply that it is not in use. This also means that the created instance of `DbConnector.java` in line 4 of Listing 1 is also not in use.

2) Worked Example 2: As an example from the S cell, the *GuitarJava* project contains the semantic dependency `Audiodevicefactory.java` ↔ `AudioDeviceBase.java`. Based on the class identifiers, the two classes are semantically related. Taking a further look into the source code snippets of both classes as shown in Listings 3 and 4, we see that similar words and phrases (e.g, `audio device` and `audiodevice`), are used throughout their code and comments (highlighted in green color). However, the two classes are not structurally linked: no structural link ties these two classes, apart from semantic

```
1  public abstract class AudioDeviceFactory
2  {
3      /**
4      * Creates a new <code>AudioDevice</code>
5      * @return a new instance of a specific
           class of <code>AudioDevice</code>.
6      *@throws JavaLayerException if an instance
           of
7      * AudioDevice could not be created
8      *
9      */
10     public abstract AudioDevice
           createAudioDevice() throws
           JavaLayerException;
11     /**
12     * Creates an instance of an AudioDevice
           implementation.
13 ...
```

Listing 3: Audiodevicefactory.java

```
1  /**
2   * The <code>AudioDeviceBase</code> class
        provides a simple thread-safe
3   * implementation of the <code>AudioDevice</
        code> interface.
4   */
5  public abstract class AudioDeviceBase
       implements AudioDevice
6  {
7  ...
8  /**
9  * Opens this audio device.
10 * @param decoder The decoder that will provide
       audio data
11 * to this audio device.
12 */
13 ...
```

Listing 4: AudioDeviceBase.java

similarities in their corpora and identifiers.

D. Is there a significant linear relationship between the strengths of structural and semantic coupling? – RQ1

The **RQ1** only applies to the E cell in Table I and it tests for a linear relationship between the strengths of structural and semantic links between pairs of classes. The Spearman's rank correlation [10] was run for all the projects in the sample: given a project, we created two vectors, one with the values of 'number of references' between pairs of classes; the other with all the values of their pair-wise semantic similarity. This is to enable us investigate whether the strengths of semantic and structural coupling have a statistical effect on each other and the extent to which they co-vary. We reject the null hypothesis for all the projects studied at the 99% confidence level, so $\alpha = 0.01$ as the rejection threshold. The $\alpha = 0.01$ level minimizes the threat of making a type I error - mistakenly rejecting a null hypothesis. The null hypothesis $H_{0.1}$ as stated in Table II to be tested is as follows:

- $H_{0.1}$: *No linear relationship between the strengths of structural and semantic class dependencies.*

The R studio data analysis tool[11] was used to compute correlation. The Spearman's metric (non-parametric) was chosen because it is unlikely that the structural (or semantic) coupling values will have a normal distribution across all the class dependencies *per* project. The value of the correlation coefficient lies in the range $[-1; 1]$, where -1 indicates a strong negative correlation and 1 indicates a strong positive correlation. We adapt the categorisation for correlation coefficients used in [23] ($[0-0.1]$ 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 at the $\alpha = 0.01$ level.

E. Is there a dependency between structural and semantic dependencies in OO software? – RQ2

To answer **RQ2**, we adopted the populated contingency tables for each project to understand the dependency between the two categorical variables (structural and semantic class dependencies) per project. We then went further to statistically test for the strength of this association using the Fisher's Exact Test of Independence, that performs better than the Chi-square statistical X^2 on small sample sizes [12,13], and in case of missing values from any of the cells. The test asserts the independence of two categorical variables, with a null hypothesis $H_{0.2}$ of *no association between them*. Similarly to the Spearman's Rank Correlation tests, we reject the null hypothesis if $\alpha > 0.01$. Section IV will outline and discuss the results for the overall studied sample of 79 OSS projects.

IV. Results

A. Spearman's Rank Correlation (ρ) - RQ1

This section presents the correlation results using the Spearman's rank test. The Spearman's rank correlation measures the strength of a linear relationship between two variables. In this case, the two variables are the strengths of the structural and semantic dependencies between classes in our studied sample of OO software projects. For the strength of semantic links, we adopted the N-Gram results between pairs of classes. Due to space constraints, correlation results for a randomly selected ten projects (out of our sample of projects) are listed in Table IV. The three columns contain the project names, correlation coefficients and the p-values, respectively. Highlighted are the results with significant p-values ($<=$ 0.01). However, the correlation values for these are low. This is further presented in the box-plot in Figure 1 where the median correlation coefficient is 0 and the median p-value is just below 0.4. There are two outliers in Figure 1 represented by two circles with correlation coefficients higher than 0.3. The first is the *jmemcache* project with a correlation coefficient of 0.5, however the p-value is 0.3. The second is the *robostcoupe* project with a significant and large positive correlation coefficient of 0.8 and a p-value of 0.00000016.

[11] https://www.rstudio.com/
[12] http://www.biostathandbook.com/fishers.html
[13] https://www.r-bloggers.com/contingency-tables--fishers-exact-test/

TABLE IV: Spearman's rank correlation (structural vs semantic (n-gram) dependency strengths (10 projects))

Project Name	Correlation Coefficient	p-value
2dtetris	0.1	0.56574
aima-java	0.2	0.00001
alexo-chess	-0.4	0.00003
alto	0.1	0.04559
castanea	-0.2	0.31923
daedalum	0.1	0.63592
dbmigrate	-0.1	0.85998
echo-nest-java-api	-0.3	0.34961
fyllgen	-0.1	0.45667
google-voice-java	0.3	0.04806

Among the negative correlations, there is one significant outlier. Shown in Table IV, the *alexo-chess* project exhibits a significant (p-value < 0.01) and negative moderate correlation coefficient (-0.4). A deeper look into the coupling pattern between the classes in this project shows that classes with a high number of structural links between them exhibit a low semantic couping. These results demonstrate that we cannot infer a strong semantic coupling between classes, just because they demonstrate a strong structural relationship. In addition, *we cannot reject the null hypothesis* due to significant correlation results observed in only a small subset of the studied sample.

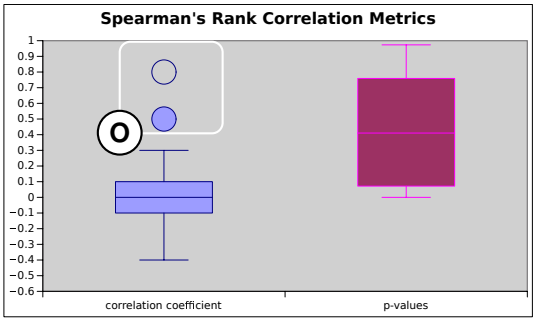

Fig. 1: Spearman's Rank Correlation Results for Overall Sample – Showing two outliers with strong positive correlation > 0.4, one insignificant with p-value=0.3 (O)

B. The dependency between semantic and structural class dependencies - RQ2

For each project in the sample, we could categorise all pairs of classes using 2x2 contingency Tables based on the type(s) of dependencies between them (structural and/or semantic). Table V presents the number of class pairs belonging to each cell of the contingency Table, but only for a randomly selected ten projects in the studied sample, due to space constraints. The first column shows the project names. The second to sixth column shows the number of class pairs belonging to the cells of the generic contingency table and the corresponding p-value derived from the Fisher's exact independence test *per* project.

[14]Not a Number. NaN is usually the product of some arithmetic operations, such as 0/0

TABLE V: E, S, W and x sets of class pairs (10 projects)

Project Name	E	S	W	x	p (F)	p (C)
2dtetris	16	64	5	0	0	0.0005
4-connect	0	18	0	1	1	Nan[14]
ahs-scheduling	6	52	1	3	0.4	1
aima-java	547	2,875	2,627	193	0	0
alto	258	1,328	1,809	28	0	0
bluecove	142	559	1,338	423	0	0
castanea	18	122	22	6	0	0
daedalum	35	211	98	78	0	0
dbmigrate	9	4	2	0	1	1
echo-nest-java-api	10	103	15	21	0.00002	0

In addition to Table 5, Figure 2 shows the distribution of the percentage of class pairs in each contingency table cell in the overall studied sample of 79 OO software projects. Figure 2 shows that a majority of the class pairs belong to cells S and W of the contingency tables with cell S having a median of around 58% of class pairs. These are the class pairs that need reworking to minimize the number of hidden dependencies during change impact analysis. Cell E has a median of around 10%. Meaning the intersection of class dependencies detected by an analysis of structural and semantic coupling *per* project in our sample is low.

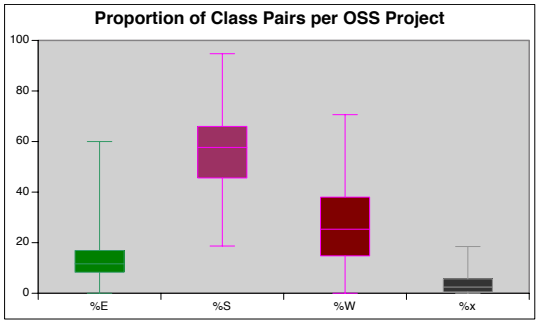

Fig. 2: Proportion of Class Pairs belonging to each cell in Contingency Table III per Project in Studied Sample

To further test for the independence between structural and semantic class dependencies, we applied the Fisher's test on the contingency tables derived for each project. This is to test for a significant relationship or association between structural and semantic class dependencies in OO software. The test asserts that the two variables being investigated are categorical and the data has been collected by random sampling. We categorized structural and semantic dependencies in Section III-C and the projects in our sample were randomly selected. Using the R studio data analysis tool[15], we computed the tests for each project in the sample. We set the p-value to 0.01 in order to improve the generalizeability of the results derived from our random sample of OO projects on a larger sample and to minimize the threat of making a Type I error.

Out of a total of 79 projects, the 72 with a relative distribution of class pairs in the four cells of the contingency tables produced p-values < 0.01. However, there were 7 outliers, all

[15]https://www.rstudio.com/

with either a small number of class pairs or none in some cells including the *4-connect*, *ahs-scheduling* and *dbmigrate* OSS projects in Table V, and their p-values are all > 0.05 with maximum = 1. A small number or absence of class pairs in some cells of the contingency Table do skew [24] the Fisher's statistical test results. We derived similar results (p-values < 0.01) for a significant majority of the projects when we used the Chi-square test for independence to confirm whether the low p-values derived from the Fisher's test were due to overflow errors (the influence of larger values in some cells). Overall Chi-square independence test results are presented in the seventh column of Table V.

Based on our results there is significant evidence to reject the null hypothesis: *Structural and semantic dependencies are independent*. In addition, the mean of the p-values derived from the Fisher's exact independence test is 0.05 which means that overall there is a 5% chance that we have mistakenly rejected the null hypothesis. However, looking at Figure 2 which summarizes the proportion of links belonging to each contingency table cell in the overall sample, it is evident that cells S and W have a larger proportion of class dependencies. These dependencies need further reworking to minimize unnoticeable coupling during impact analysis. Only 10% of the links in the studied sample on average will be noticeable by an analysis of both structural AND semantic coupling. Around 58% of the coupling links will be noticed by an analysis of ONLY structural coupling. This is a high proportion of unnoticed dependencies and propagators of ripple effects during CIA. Finally, around 30% of the links will be noticed by semantic analysis ONLY. These results support the earlier argument in Section III-C regarding the need to pay attention to the class pairs in cells S and W of the contingency tables. Classes with structural links should be semantically linked to maximise the number of dependencies uncovered by the structural analysis of the source code and the semantic information embedded in the class identifiers and comments.

V. Discussion

Spearman's Rank correlation ρ results for RQ1 as outlined in Section IV-A, have not provided a strong evidence for rejecting the null hypothesis $H_{0.1}$.

> *No linear relationship between the strength of structural and semantic links between pairs of classes. A strong structural link does not imply a strong semantic link.*

The analysis of the dependency between structural and semantic coupling (RQ2) is presented in Section IV-B. The results in Table V and Figure 2 show that only a small proportion of class pairs have both structural and semantic dependency links between them. This means that during CIA, an analysis of only the source code (or only the semantic information embedded in class identifiers and comments) will not reveal complete change impact sets. However, further results from the Fisher's exact test and confirmed by a Chi-square independence test have shown a significant association between structural and semantic class dependencies that needs improvement.

> *There is a small overlapping between semantic and structural class dependencies. However, structural and semantic dependencies are significantly associated.*

We propose two refactoring operations Subsections V-B and V-A to decrease unnoticeable class coupling during change impact analysis, improve impact set prediction and reduce testing efforts in OO software. Also, in V-C, we have outlined applications of our results in software engineering.

A. Solving the S dependencies: 'rename class' design pattern

In Section III-C1, we presented an example of class pairs belonging to the S cell (i.e., the hidden dependencies). These are class pairs with structural links but without semantic links. In the example presented, there are no comments embedded within the source code of the classes to describe the referenced function `insertMove()`. Moreover, the class identifiers are not semantically linked. Due to the lack of embedded semantic information within the source code or the class identifiers, an analysis of only semantic dependencies during CIA will not reveal the structural dependency between these class pairs, causing this dependency to be unnoticed. Previous research has emphasised the importance of the quality of identifiers and comments within source code because of the information they provide [23] for a number of software development and evolution tasks [25], [26]. According to Ujhazi *et al.*, improving the quality of the underlying textual information in source code can be done by *"applying advanced source code pre-processing techniques for splitting and expanding identifiers and comments in software"* [2].

Fisher's exact independence test results in Section IV-B have shown a significant association between structural and semantic class coupling. Therefore, with the goal of minimizing the number of hidden dependencies during CIA and strengthening this association, we propose two refactoring activities for OO software class pairs belonging to the S cell:

- the addition of meaningful comments in both the caller and called classes as well as the affected methods;
- the use of conceptually related identifiers by means of renaming related methods and classes, using the "Rename Class" design pattern [27].

B. Solving the W dependencies: 'extract class' design pattern

In Section III-C1, we described an example of class pairs belonging to the W category (i.e., a weak hidden dependency). Differently from those in cell S, these are class pairs with only semantic links. In the refactoring domain [28], a refactoring that helps in addressing the pairs of classes in the W subset is referred to as an *Extract Class* (EC) [29]. EC is a refactoring that analyzes the (structural and/or semantic) similarity of the methods in a class in order to identify chains of strongly related methods (i.e., *method chains*). Those identified method chains are further adapted to define new classes with higher cohesion than the original class. Using a shared definition, cohesion is the *"degree to which elements of a module belong together"* [30] and classes are a set of responsibilities [23]. Past findings in this context states that *"classes with unrelated*

methods often need to be restructured by distributing some of their responsibilities to new classes, thus reducing their complexity and improving their cohesion" [29]. Research has shown that EC is able to identify meaningful refactoring operations and new cohesive classes [16], [28], [29].

From the results of Fisher's test and previous research, we are proposing an improved EC approach based on the semantic similarity of methods belonging to class pairs in cell W (Table III (top)). This is geared towards minimizing the variance in class dependencies detected by structural and semantic coupling analysis during CIA. In contrast to the previous approach proposed by Bavota *et al.*, our approach also extracts chains of related methods from the class pairs, but based on the semantic similarity of the methods. It is noteworthy that this is ONLY done when the classes do not have a strong internal structure based on semantic cohesion [23] (e.g., Listing 4 and 3) which is also an important attribute of OO software classes. Kabaili *et al.* [31] state that *"some classes have multiple methods that share no variables but perform related functionalities; placing each method in a different class would be against good OO design"*. Figure 3 presents a pictorial view of our proposed EC refactoring workflow, partly adapted from the original (Figure 1 in [29]).

Using Figure 3 we propose that the newly created classes are a combination of already existing structurally coupled, classes and their closest semantically coupled extracted method chain from only classes with weak internal structures. This design decision was taken to prevent an increase in overall coupling and to preserve the software architecture, while increasing the semantic cohesion of classes. If new classes are created using only the extracted chain of semantically related methods, the hidden dependencies would remain undetected during CIA; those classes would be semantically cohesive, but without structural links to other classes. To avoid having new classes with a very low number of methods, Bavota *et al.* [29] merged each trivial chain with the most coupled non-trivial chain to obtain the final set of classes to be extracted from the original class. In our adapted approach, each method chain is merged with the most semantically coupled (already existing) class, which in turn must have structural links to one or more classes.

C. Applications in Software Engineering

In relation to our results in Section IV, establishing whether structural and semantic coupling have a direct influence on each other has several applications in software engineering:

1) **Improving efficiency and accuracy during impact analysis**: Previous researchers have combined pairs of coupling types in software maintenance tasks (which in itself can be inefficient when analyzing large systems) with minimal accuracy [3], [4]. Therefore, it is imperative to bridge the gap between structural and semantic coupling with the aim of improving impact analysis. In future work, the proposed refactoring techniques in Section V-A and V-B will be further explored.
2) **Concentrated and reduced testing efforts**: when changes are made to one class, other classes with strong structural or semantic coupling to that class should be tested [3], [4], [8]. This is to ensure that the changes in one class do not introduce regression faults in other classes. If the overlapping between structural and semantic dependencies is large, then these tests only have to be carried out once; only semantically coupled class pairs will need to be tested or ONLY structurally coupled pairs.
3) **Minimizing the model-code gap**: Prior software maintenance research has shown that complete and/or current architecture descriptions rarely exist [32]. Reconstructing software architectures can aid comprehension. Thus, a significant proportion of the architectural dependencies could be reconstructed by relying solely on static semantic dependency information [33].

VI. Previous Work

Compared to structural coupling research in OO software systems, the semantic coupling research field is still 'young' and evolving as tools and techniques become available and with the inclusion of more researchers in the domain. No empirical study has been carried out on the link between the semantic and structural coupling between classes.

A. Change Impact Set and Unnoticeable Dependencies in Object-Oriented Software

According to Poshyvanyk *et al.* [8], *"Existing models do not capture all ripple effects of changes in existing software"* [18]. They argued that semantic dependencies also *"propagate changes in software"*. Based on a case study on the source code of the Mozilla web browser, they compared the conceptual coupling metrics to nine structural coupling metrics and concluded that conceptual coupling metrics are better predictors for classes impacted by changes. However, their results are not supported by our results and those of Abdeen *et al.* [4] who have performed inter-system and intra-system change impact prediction using structural, and semantic dependencies. While our results have shown that not all class dependencies can be captured by semantic coupling, Abdeen *et al.* identified that using semantic coupling produces better recall values in the intra-system scenario. On the other hand, they identified that using structural dependencies or a combination of both types of dependencies outperformed the use of structural or semantic dependencies only as an addition of semantic coupling data provides extra information during the learning phase.

Based on their results, it remains unclear whether the observed phenomenon was a result of the presence of an 80:20 rule in the relationship between structural and semantic coupling. It could have been that only 20% of the structural dependencies might have led to co-change or could be accounted for by 80% or more of the semantic dependencies. Thus a combination of both dependency types would improve the model by a small degree. Our results provide a statistical backing for the results in [4]. Sharma and Suryanarayana [19] developed a Visual Studio extension for inter-granular static CIA. The tool supports inter-granular change impact queries

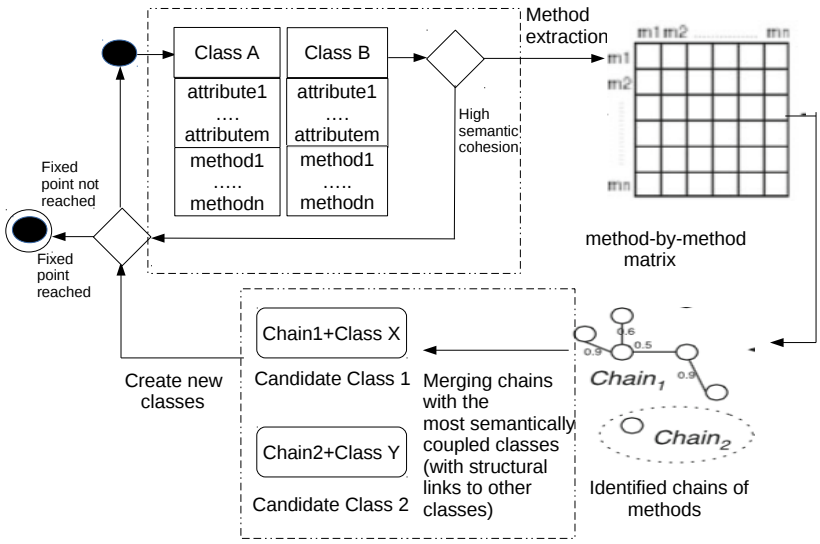

Fig. 3: Extract class refactoring for pairs of semantically related and non-structurally related classes in Cell W Table III (top)), partly adapted from [29] Fig. 1. (dashed components are our additions to the original model)

and *"hidden dependencies"* embedded in semantic coupling alongside structural coupling between source code artifacts across various levels of granularity. Results from four software systems, revealed a low precision of 55% when comparing the estimated change impact sets to the actual impact sets.

B. Coupling in Object-Oriented Software

Gethers and Poshyvanyk [13] proved that their proposed relational topic-based coupling (RTC) metric not only captures new dimensions of coupling between classes, which are not covered by the existing structural coupling metrics, but also can be used to effectively *support* impact analysis. These results counter those earlier presented by Poshyvanyk *et al.* [8] where they stated that conceptual coupling metrics better predict change impact sets.

Previous research has shown that there is a gap in the literature with regards to analysing the relationship between structural and semantic coupling at the class level of granularity in OO software [11]. In this study, we have identified that there is a wide gap between class dependencies captured by structural and semantic coupling. Related work has shown that using the metrics to support each other during impact set prediction is not very efficient in terms of accuracy. We have further proposed feasible and testable approaches to reduce this gap and minimize the variation in the dependencies captured by structural and semantic coupling analysis during CIA [18].

VII. Threats to Validity

In this section we identify the threats to validity of the results presented in Section IV. Firstly, we cannot generalise our findings on a different sample of OSS projects. Nonetheless, to make our findings more generalisable and representative of OSS projects, we have analyzed a large and random sample of projects, of different sizes in terms of number of classes and historical data (number of past revisions). The scope of our sample of projects is also limited to OO projects written in Java. We encourage investigating projects written in other programming languages, non-object-oriented software projects and commercial software.

Secondly, the semantic coupling measurement adopted is based on class identifiers and could have had an effect on the identified semantic dependencies. However, in a prior study, we identified that identifier-based metrics closely reflect class corpora-based semantic coupling metrics [15]. The semantic similarity between class identifiers has previously been combined with historical data in ranking classes that might be impacted by a given change request. Our approach follows a previous research study [3]. Lastly, the Fisher's exact test for independence will yield a false negative (no significant association) when there are no unrelated class pairs. However, this rarely occurred in our sample there are only 13 projects out of 79 without unrelated pairs of classes. Similarly, in a different sample, this observation will be rare as it is unlikely that every single class in an OO project will be both structurally and semantically linked as they will be used in different domains.

VIII. Conclusion and Further Work

In this study we have conducted an empirical analysis on OO software projects to understand the relationship between structural and semantic class dependencies. This is the first empirical study of this kind and was carried out on 79 Java OSS projects with the aim of managing hidden

class dependencies before, during and after change impact analysis. Based on our investigations, dependencies between class pairs fall in two major categories; the *ideal* (without hidden dependencies) and *non-ideal* categories (with hidden dependencies) with reference to the motivating contingency Table I. Our results show that: (1) Only a small fraction of class pairs belong to the ideal category. This implies that running a CIA with only a semantic or structural analysis will result in a large variation of estimated change impact sets; (2) There is no correlation between the strengths of structural and semantic links: a higher number of service connections between two classes does not imply a higher similarity in their semantics, and; (3) There is evidence of a significant association between structural and semantic dependencies: however, some classes linked by structural coupling are not usually linked by semantic similarities and these classes need reworking to minimize unnoticeable coupling during CIA.

We expanded the findings of our empirical study with two practical approaches outlined in Subsections V-B and V-A to solve the hidden dependencies. As future work, we plan to empirically analyze the refactoring approaches in the context of OSS projects of different domains and implement a refactoring plug-in for a Java IDE. This tool will be able to semi-automatically handle the proposed refactoring approaches. The future evaluation of these refactoring approaches will establish whether the prediction of the impact sets can be improved, as well as reduce testing efforts in OO software.

We also encourage the replication of this study using a different sample of software projects to validate the presented results.

REFERENCES

[1] B. Li, X. Sun, H. Leung, and S. Zhang, "A survey of code-based change impact analysis techniques," *Software Testing, Verification and Reliability*, vol. 23, no. 8, pp. 613–646, 2013.

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

[3] H. Kagdi, M. Gethers, and D. Poshyvanyk, "Integrating conceptual and logical couplings for change impact analysis in software," *Empirical Software Engineering*, vol. 18, no. 5, pp. 933–969, 2013.

[4] H. Abdeen, K. Bali, H. Sahraoui, and B. Dufour, "Learning dependency-based change impact predictors using independent change histories," *Information and Software Technology*, vol. 67, pp. 220–235, 2015.

[5] M. M. Geipel and F. Schweitzer, "The link between dependency and cochange: empirical evidence," *Software Engineering, IEEE Transactions on*, vol. 38, no. 6, pp. 1432–1444, 2012.

[6] H. Gall, K. Hajek, and M. Jazayeri, "Detection of logical coupling based on product release history," in *International Conference on Software Maintenance*. IEEE, 1998, pp. 190–198.

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

[8] D. Poshyvanyk, A. Marcus, R. Ferenc, and T. Gyimóthy, "Using information retrieval based coupling measures for impact analysis," *Empirical software engineering*, vol. 14, no. 1, pp. 5–32, 2009.

[9] R. Vanciu and V. Rajlich, "Hidden dependencies in software systems," in *Software Maintenance (ICSM), 2010 IEEE International Conference on*. IEEE, 2010, pp. 1–10.

[10] L. Yu, "Understanding component co-evolution with a study on linux," *Empirical Software Engineering*, vol. 12, no. 2, pp. 123–141, 2007.

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

[12] D. Flanagan, *Java in a Nutshell*. " O'Reilly Media, Inc.", 2005.

[13] M. Gethers and D. Poshyvanyk, "Using relational topic models to capture coupling among classes in object-oriented software systems," in *Software Maintenance (ICSM), 2010 IEEE International Conference on*. IEEE, 2010, pp. 1–10.

[14] J. Rilling, R. Witte, D. Gaševic, and J. Z. Pan, "Semantic technologies in system maintenance (stsm 2008)," in *Program Comprehension, 2008. ICPC 2008. The 16th IEEE International Conference on*. IEEE, 2008, pp. 279–282.

[15] N. Ajienka and A. Capiluppi, "Semantic coupling between classes: Corpora or identifiers?" in *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '16. Ciudad Real, Spain: ACM, 2016, p. 40.

[16] G. Bavota, A. De Lucia, and R. Oliveto, "Identifying extract class refactoring opportunities using structural and semantic cohesion measures," *Journal of Systems and Software*, vol. 84, no. 3, pp. 397–414, 2011.

[17] Z. Yu and V. Rajlich, "Hidden dependencies in program comprehension and change propagation," in *9th International Workshop on Program Comprehension (IWPC)*. IEEE, 2001, pp. 293–299.

[18] L. C. Briand, J. Wuest, and H. Lounis, "Using coupling measurement for impact analysis in object-oriented systems," in *IEEE International Conference on Software Maintenance, (ICSM)*, 1999, pp. 475–482.

[19] T. Sharma and G. Suryanarayana, "Augur: Incorporating hidden dependencies and variable granularity in change impact analysis."

[20] J. Howison, M. Conklin, and K. Crowston, "Flossmole: A collaborative repository for floss research data and analyses," *International Journal of Information Technology and Web Engineering (IJITWE)*, vol. 1, no. 3, pp. 17–26, 2006.

[21] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. M. German, and D. Damian, "An in-depth study of the promises and perils of mining github," *Empirical Software Engineering*, vol. 21, no. 5, pp. 2035–2071, 2016.

[22] J. Lindsay, J. Noble, and E. Tempero, "Does size matter?: a preliminary investigation of the consequences of powerlaws in software," in *Proceedings of the 2010 ICSE Workshop on Emerging Trends in Software Metrics*. ACM, 2010, pp. 16–23.

[23] A. Marcus and D. Poshyvanyk, "The conceptual cohesion of classes," in *21st IEEE International Conference on Software Maintenance (ICSM)*, 2005, pp. 133–142.

[24] K. Zhukov, "Exploring the role of technology in instrumental skill development of australian higher education music students," *Australian Journal of Music Education*, no. 2, p. 66, 2015.

[25] N. Anquetil and T. Lethbridge, "Assessing the relevance of identifier names in a legacy software system," in *Proceedings of the 1998 conference of the Centre for Advanced Studies on Collaborative Research*. IBM Press, 1998, p. 4.

[26] B. Caprile and P. Tonella, "Restructuring program identifier names." in *icsm*, 2000, pp. 97–107.

[27] E. Gamma, *Design patterns: elements of reusable object-oriented software*. Pearson Education India, 1995.

[28] M. Fowler and K. Beck, *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 1999.

[29] G. Bavota, A. De Lucia, A. Marcus, and R. Oliveto, "Automating extract class refactoring: an improved method and its evaluation," *Empirical Software Engineering*, vol. 19, no. 6, pp. 1617–1664, 2014.

[30] J. M. Bieman and B.-K. Kang, "Cohesion and reuse in an object-oriented system," in *ACM SIGSOFT Software Engineering Notes*, vol. 20, no. SI. ACM, 1995, pp. 259–262.

[31] H. Kabaili, R. K. Keller, and F. Lustman, "Cohesion as changeability indicator in object-oriented systems," in *Fifth European Conference on Software Maintenance and Re-engineering*. IEEE, 2001, pp. 39–46.

[32] A. Dragomir, M. F. Harun, and H. Lichter, "On bridging the gap between practice and vision for software architecture reconstruction and evolution: A toolbox perspective," in *Proceedings of the WICSA 2014 Companion Volume*. ACM, 2014, p. 10.

[33] A. Siddiqui, "Is software architecture still a shared hallucination?" *Full-scale Software Engineering*, p. 1, 2015.