The Journal of Systems and Software 134 (2017) 120–137

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Understanding the interplay between the logical and structural coupling of software classes

Nemitari Ajienka*, Andrea Capiluppi

Brunel University London, Kingston Lane, Uxbridge, Middlesex, UB8 3PH, UK

ARTICLE INFO

Article history:
Received 17 March 2017
Revised 14 August 2017
Accepted 22 August 2017
Available online 24 August 2017

Keywords:
Object-oriented (OO)
Open-source software (OSS)
References
Structural coupling
Co-changed structural dependencies (CSD)
Coupled logical dependencies (CLD)

ABSTRACT

During the lifetime of object-Oriented (OO) software systems, new classes are added to increase functionality, also increasing the inter-dependencies between classes. Logical coupling depicts the change dependencies between classes, while structural coupling measures source code dependencies induced via the system architecture. The relationship or dependency between logical and structural coupling have been debated in the past, but no large study has confirmed yet their interplay.

In this study, we have analysed 79 open-source software projects of different sizes to investigate the interplay between the two types of coupling. First, we quantified the overlapping or intersection of structural and logical class dependencies. Second, we statistically computed the correlation between the strengths of logical and structural dependencies. Third, we propose a simple technique to determine the *stability* of OO software systems, by clustering the pairs of classes as "stable" or "unstable", based on their co-change pattern.

The results from our statistical analysis show that although there is no strong evidence of a linear correlation between the strengths of the coupling types, there is substantial evidence to conclude that structurally coupled class pairs usually include logical dependencies. However, not all co-changed class pairs are also linked by structural dependencies. Finally, we identified that only a low proportion of structural coupling shows excessive instability in the studied OSS projects.

1. Introduction

Various software dependency measures have been proposed over the years. *Logical coupling* is a measure of the degree to which two or more classes change together or co-evolve, based on the historical data of modifications; while *structural coupling* is a measure of the structural or source code dependencies between software classes. For example, the number of method calls between object-oriented (OO) software classes.

Establishing that two software entities *co-evolve* (i.e., they are *logically coupled*) means that developers consider them as logically related: for example, a change in one entity causes a change to be made to another entity. This is also known as the cause → effect rule.

On the other hand, *structural coupling* is the degree of interdependence between software modules, and it indicates how closely connected two modules are at the source code level. Henderson-Sellers et al. (1996) state that strong coupling complicates a system since a module is harder to understand, change, or correct by itself, if it is highly interrelated with other modules. "*Software complexity can be reduced by designing systems with the weakest possible coupling between modules*" (Henderson-Sellers et al., 1996) because "*every time a supplier class changes, its clients are also likely to change*" (Oliva and Gerosa, 2011).

In earlier studies, co-evolution of OO software classes has been studied in relation to structural coupling (Oliva and Gerosa, 2011; Yu, 2007; Geipel and Schweitzer, 2012; Oliva and Gerosa, 2015) and software quality (Zimmermann et al., 2005; D'Ambros et al., 2009a). Some of these studies showed that most of the structurally coupled related entities in software projects do not co-evolve, and the other way round (Oliva and Gerosa, 2011; Geipel and Schweitzer, 2012; Oliva and Gerosa, 2015).

Fig. 1 illustrates what has been proposed in the past, and for a smaller subset of classes: analysing the direction of the relationship between co-evolution and structural coupling for 12 Linux kernel modules (Yu, 2007), Yu identified a linear and directional relationship between the co-evolution and structural coupling. According to that work, structural coupling does not bring about independent evolution: if software classes are evolved independently, there will be no correlation between structural cou-

* Corresponding author.
E-mail addresses: nemitari.ajienka@brunel.ac.uk (N. Ajienka), andrea.capiluppi@brunel.ac.uk (A. Capiluppi).

http://dx.doi.org/10.1016/j.jss.2017.08.042
0164-1212/

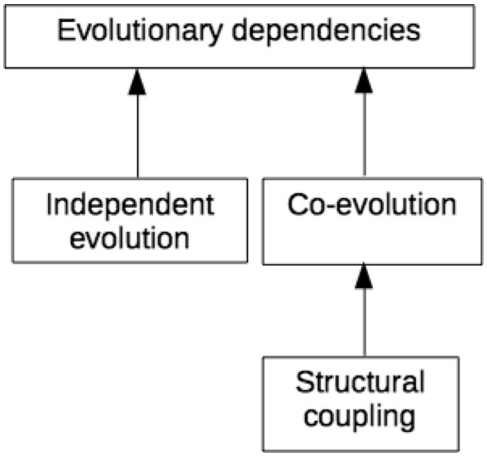

Fig. 1. The relationships among evolutionary dependencies, structural coupling and co-evolution Yu (2007) of Linux Kernel Modules.

pling and co-evolution data. In addition, according to Oliva and Gerosa (2015), controlling coupling levels in practice is still challenging. One of the reasons is that the extent to which changes propagate via structural dependencies is still not clear.

In this context and state of knowledge, this paper analyses a sample of 79 OSS projects (written in Java) in order to add evidence to the discussion on the causes of co-evolution of classes with a large sample of a variety of software projects and to work on the gaps identified in previous research (Yu, 2007; Oliva and Gerosa, 2011).

This work is articulated as follows: in Section 2 we briefly explain the types of software dependencies (coupling) under study. For the sake of replicability, in Section 3 we describe the steps taken to carry out this study, with a working example using a software project. Sections 4 and 5 highlight the findings of our study, followed by a discussion on the importance of these findings. In Section 6 we summarise the related work, and put ours into context. Section 7 highlights the threats to validity and finally, our conclusions and areas for further research are presented in Section 8.

2. Object-oriented software dependencies

A dependency is a semantic relationship that indicates that a client element may be affected by changes performed in a supplier element (Oliva and Gerosa, 2011). In the next Subsections, we introduce structural and logical dependencies and discuss how they can be operationalised in the context of OO programming.

2.1. Logical coupling

According to Wiese et al., "*change coupling is a phenomenon associated with recurrent co-changes found in the software evolution or change history*" (Wiese et al., 2015b). Therefore, the logical coupling of any two classes is based on their evolution history, and is a measure of the observation that the two classes always co-evolve or change together (Gall et al., 1998; 2003; D'Ambros et al., 2009b; Wiese et al., 2015a). They are commonly treated as association rules (Zimmermann et al., 2005), which means that when X_1 is changed, X_2 is also changed (Oliva and Gerosa, 2011). 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 (Oliva and Gerosa, 2011).

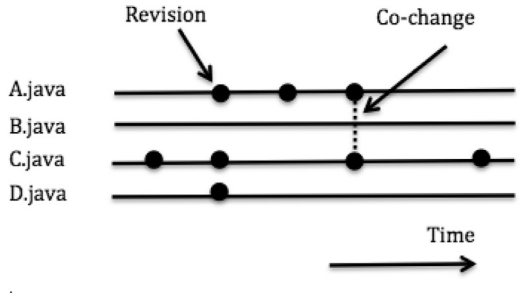

Fig. 2. Association rule example for confidence and support metrics.

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. (2013) state that logical coupling captures the extent to which software artifacts 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. (D'Ambros et al., 2006) it is useful to study logical coupling because it can reveal dependencies that are not revealed by analyzing only the source code (Yu, 2007). This sort of dependencies are the most troublesome and are prone to represent sources of bugs in software projects. Zimmermann et al. (2003) represents logical dependency using two metrics: support and confidence.

Operationalisation. Confidence and *support* are two well-known metrics used in association rule learning: 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 Fig. 2, class *A* was modified in 3 transactions (where 3 is the *"Transaction Count"* (Yu, 2007)). 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 $A \leftarrow C$.

In this paper, the degree or strength of the logical dependency between classes is evaluated using the **confidence** metric. By doing so, we evaluated the *significance* of the association rules between classes (Oliva and Gerosa, 2011), and across the lifespan of a software project (i.e., taking all versions of the software system into consideration).

As per its definition, the *confidence*[1] value of a dependency link 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 Fig. 2, 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.

Finally, 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 → effect rules can be different.

[1] Also called the support ratio (Yu, 2007). In this study we only adopt the confidence metric which is a measure of the degree to which a change in one class also leads to a change in another class.