Identifying logical dependencies from co-changing classes

blind for review

Keywords: software evolution, logical dependencies, structural dependencies

Abstract: Emerging software engineering approaches support the idea that logical dependencies should be included next to structural dependencies in general methods and tools for dependency management. However, logical dependencies are still hard to identify, as not all co-changes during the system evolution represent true logical dependencies. Our work identifies a set of factors that can be used to filter the recordings of class co-changes in order to find logical dependencies. Also we analyze the quantitative relationships between the sets of logical and structural dependencies and their intersection and differences. We present results obtained through an experimental study on a set of open source software projects with their historical evolution. Identifying logical dependencies from co-changing classes will enhance dependency models used in various software analysis activities.

1 Introduction

Coupling reflects the degree of interdependence between different software modules, being a measure of how closely connected they are. Coupling should be low in order to ensure the testability, reusability, and evolvability properties of modules. The traditional approach on coupling was to quantify the structural dependencies or interactions between modules. Structural dependencies can be determined by analysis of the source code.

State of the art has already established that modules may present different kinds and degrees of interdependence, even if no structural dependencies can be found by analyzing the source code. Gall (Gall et al., 1998) identified as logical coupling between two modules the fact that these modules repeatedly change together during the historical evolution of the software system. This can be an indicator of a logical dependency between these modules.

The concepts of logical coupling and logical dependencies were first used in different analysis tasks, all related to changes: for software change impact analysis (Ren et al., 2005), for identifying the potential ripple effects caused by software changes during software maintenance and evolution (Oliva and Gerosa, 2015), (Oliva and Gerosa, 2011), (Poshyvanyk et al., 2009), (Kagdi et al., 2010) or for their link to deffects (Wiese et al., 2015), (Zimmermann et al., 2004).

The current trend recommends that general dependency management methods and tools should also include logical dependencies besides the structural dependencies (Oliva and Gerosa, 2011), (Ajienka and Capiluppi, 2017). Different applications based on dependency analysis could be improved if, beyond structural dependencies, they also take into account the hidden non-structural dependencies. For example, works which investigate different methods for architectural reconstruction (Şora et al., 2010), (Sora, 2013), (Şora, 2015), all of them based on the information provided by structural dependencies, could enrich their dependency models by taking into account also logical dependencies. However, a thorough survey (Ducasse and Pollet, 2009) shows that historical information has been rarely used in architectural reconstruction. Another survey (Shtern and Tzerpos, 2012) mentions one possible explanation why historical information have been rarely used in architectural reconstruction: the size of the extracted information. The problem is not only the size of the extraction process, which has to analyze many versions from the historical evolution of the system, but also the number of the logical dependencies and how they relate to the number of structural dependencies. Logical dependencies should integrate harmonious with structural dependencies in a unitary dependency model: valuable logical dependencies should not be omitted from the dependency model, but structural dependencies should not be engulfed by questionable logical dependencies. Thus, in order to add logical dependencies besides structural dependencies in dependency models, class co-changes must be filtered until they remain only a reduced but relevant set of true logical dependencies.

In the next section we analyze the state of the art results for determining logical dependencies from the point of view of their quantitative relationship with structural dependencies. Starting from this analysis, in Section 3 we identify a set of factors that can be used to filter the recordings of class co-changes such that logical dependencies are identified and formulate the research questions. In order to answer these research questions, we have built a tool that extracts structural and logical dependencies in different scenarios. The design and implementation of the tool is briefly presented in section 4. We have analyzed several open-source software systems of different sizes with our tool, obtaining the experimental results presented in Section 5. Section 6 discusses the experimental results and answers the research questions. Threats to validity and future work directions are identified in Section sec:validity.

2 State of the art

There are researches that investigated quantitative aspects of logical dependencies and their interplay with structural dependencies. Oliva and Gerosa (Oliva and Gerosa, 2011), (Oliva and Gerosa, 2015) have found first that the set of co-changed classes was much larger compared to the set of structurally coupled classes. They identified structural and logical dependencies from 150000 revisions from the Apache Software Foundation SVN repository. Also they concluded that in at least 91% of the cases, logical dependencies involve files that are not structurally related. This implies that not all of the change dependencies are related to structural dependencies and there could be other reasons for software artifacts to be change dependent.

Ajienka and Capiluppi also studied the interplay between logical and structural coupling of software classes. In (Ajienka and Capiluppi, 2017) they perform experiments on 79 open source systems: for each system, they determine the sets of structural dependencies, the set of logical dependencies and the intersections of these sets. They quantify the overlapping or intersection of these sets, coming to the conclusion that not all co-changed class pairs (classes with logical dependencies) are also linked by structural dependencies. One other interesting aspect which has not been investigated by the authors in (Ajienka and Capiluppi, 2017) is the total number of logical dependencies, reported to the total number of structural dependencies of a software systems. However, they provide the raw data of their measurements and we calculated the ratio between the number of logical dependencies and the number of structural dependencies for all the projects analyzed by them: the average ratio resulted 12. This means that, using their method of detecting logical dependencies for a system, the number of logical dependencies outnumbers by one order of magnitude the number of structural dependencies.

Another kind of non-structural dependencies are the semantic or conceptual dependencies (Poshyvanyk et al., 2009), (Kagdi et al., 2010). Semantic coupling is given by the degree to which the identifiers and comments from different classes are similar to each other. Semantic coupling could be an indicator for logical dependencies, as studied by Ajienka et al in (Ajienka et al., 2018). The experiments showed that a large number of co-evolving classes do not present semantic coupling, adding to the earlier research which showed that a large number of co-evolving classes do not present structural coupling. All these experimental findings rise the question whether it is a legitimate approach to accept all co-evolving classes as logical coupling.

Changes made to two components in the same commit do not necessarily indicate the co-evolution of the two. These changes could be completely unrelated. The study (Yu, 2007) acknowledges the fact that evolutionary coupling could also be determined accidentally by two components changing in the same commit (independent evolution, as it is called) and this will bring noise to the measurement of evolutionary coupling.

Zimmermann et al (Zimmermann et al., 2004) introduced data mining techniques to obtain association rules from version histories. The mined association rules have a probabilistic interpretation based on the amount of evidence in the transactions they are derived from. This amount of evidence is determined by two measures: support and confidence. They developed a tool to predict future or missing changes.

In order to add logical dependencies besides structural dependencies as inputs for methods and tools for dependency management and analysis, class co-changes must be filtered until they remain only a reduced but relevant set of true logical dependencies.

3 Research questions

In this work, we explore several ways of filtering logical dependencies. We identify following factors that could be used to filter logical dependencies: the maximum number of files in a commit accepted as logical dependencies, the minimum number of occurrences for a co-change to be considered a logical dependency, and accepting changes in comments as a source of logical dependencies.

We will address the following research questions:

Question 1. Which is the most frequent size for a commit transaction ?

Motivation: We calculate the size for a commit transaction as the total number of source code files that have changed. Even though the versioning systems best practices encourage developers to commit often which implies small size commit transactions, the size of the commit transaction relies also on the developers culture. We think that finding the most frequent size for a commit transaction could help into setting ranges for what is a normal size commit transaction for the systems. And also to set a target commit transaction group from which we can extract logical dependencies.

Question 2.Is it necessary to set a threshold on the size of commit transactions which are considered to generate valid logical dependencies ?

Motivation: A big commit transaction can indicate that a merge with another branch or a folder renaming has been made. In this case, a series of irrelevant logical dependencies can be introduced since not all the files are updated in the same time for a development reason. Different works have chosen fixed threshold values for the maximum number of files accepted in a commit. Cappiluppi and Ajienka, in their works (Ajienka and Capiluppi, 2017), (Ajienka et al., 2018) only take into consideration commits with less then 10 source code files changed in building the logical dependencies. The research of Beck et al (Beck and Diehl, 2011) only takes in consideration transactions with up to 25 files. The research (Oliva and Gerosa, 2011) provided also a quantitative analysis of the number of files per revision; Based on the analysis of 40,518 revisions, the mean value obtained for the number of files in a revision is 6 files. However, standard deviation value shows that the dispersion is high. Based on all these considerations, we will experiment with different values for the threshold value for the maximum number of files accepted in a commit.

Question 3. Considering changes which are only in comments as valid can lead to additional logical dependencies? How many logical dependencies are introduced by considering comment changes as valid changes and in what percentage can this influence the analysis?

Motivation: Not all the commits that have source code files changed include real code changes, some of them can be only comments changes. We consider that there is probably no logical dependency between two classes that change in the same time only by comments changes. It could be that someone is adding implementation documentation or copyright or ownership information. Some studies have not considered this aspect, so we will analyse the impact of considering/not considering changes in comments as valid logical dependencies.

Question 4. How many occurrences of a logical dependency are needed to consider it a *valid* logical dependency ?

Motivation: One occurrence of a logical dependency between two classes can be a valid logical dependency, but can also be a coincidence. Taking into consideration only logical dependencies with multiple occurrences as valid dependencies can lead to more accurate logical dependencies and more accurate results. On the other hand, if the project studied has a relatively small amount of commits, the probability to find multiple updates of the same classes in the same time can be small, so filtering after the number of occurrences can lead to filtering all the logical dependencies extracted. Giving the fact that we will study multiple projects of different sizes and number of commits, we will analyze also the impact of this filtering on different projects.

Question 5. How does filtering affect the overlap between structural and logical dependencies ?

Motivation: Traditional software engineering considers coupling as the cause for co-changes, thus logical and structural dependencies should present a very big overlap. However, in (Oliva and Gerosa, 2011) and (Ajienka and Capiluppi, 2017) has been experimentally determined that a very large number of logical dependencies are outside the intersection with structural dependencies. We will investigate the influence of different filtering degrees on the intersections between logical and structural dependencies.

4 Tool for measuring software dependencies

In order to build structural and logical dependencies we have developed a tool that takes as input the source code repository and builds the required software dependencies. The workflow can be delimited by three major steps as it follows (Figure 1):

Step 1: *Extracting structural dependencies.*
Step 2: *Extracting logical dependencies.*
Step 3: *Processing the information extracted.*

4.1 Extracting structural dependencies

A structural dependency between two classes A and B is given by the fact that A statically depends on B, meaning that A cannot be compiled without knowing

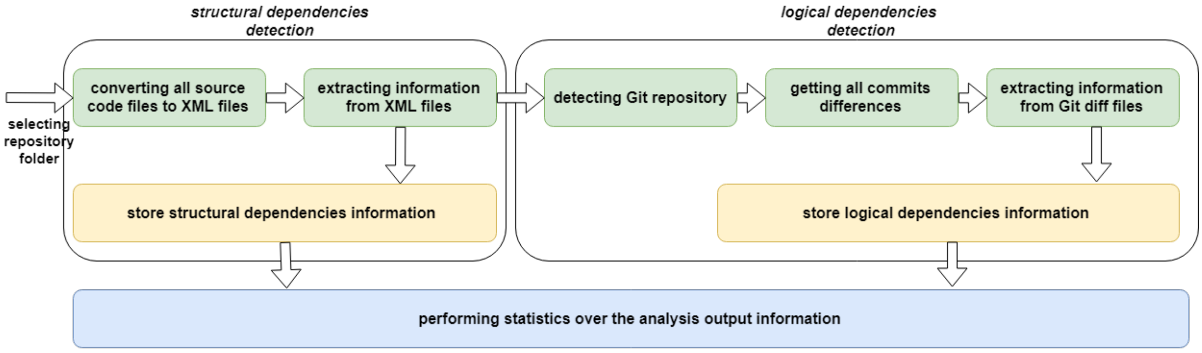

Figure 1: Processing phases

about B. In object oriented system, this dependency can be given by many types of relationships between the two classes: A extends B, A implements B, A has attributes of type B, A has methods which have type B in their signature, A uses local variables of type B, A calls methods of B.

We use an external tool called srcML (Collard et al., 2003), (Collard et al., 2011) to convert all source code files from the current release into XMl files. All the information about classes, methods, calls to other classes are afterwards extracted by our tool parsing the XML files and building a dependencies data structure. We have chosen to rely on srcML as a preprocessing tool because it reduces a significant number of syntactic differences from different programming languages and can make easyer the parsing of source code written in different programming languages such as Java, C++ and C#.

4.2 Extracting logical dependencies

The versioning system contains the long-term change history of every file. Each project change made by an individual at a certain point of time is contained into a commit (Collins-Sussman et al., 2010). All the commits are stored in the versioning system chronologically and each commit has a parent. The parent commit is the baseline from which development began, the only exception to this rule is the first commit which has no parent. We will take into consideration only *commits that have a parent* since the first commit can include source code files that are already in development (migration from one versioning system to another) and this can introduce redundant logical links (Ajienka and Capiluppi, 2017).

The tool looks through the main branch of the project and gets all the existing commits. For each commit a diff against the parent will be made and stored. Here we have the option to ignore commits that contain more files than a threshold value for commit size. Also, we have the option to check whether the differences are in actual code or if they affect only parts of source files that are only comments. Finally after all the difference files are stored, all the files are parsed and logical dependencies are build. For a group of files that are committed together, logical dependencies are added between all pairs formed by members of the group. Adding a logical dependency increases an occurrence counter for the logical link.

5 Experimental results

We have analysed a set of open-source projects found on GitHub[1] (Kalliamvakou et al., 2016) in order to extract the structural and logical dependencies between classes. Table 1 enumerates all the systems studied. The 1st column assigns the projects IDs; 2nd column shows the project name; 3rd column shows the number of entities(classes and interfaces) extracted; 4th column shows the number of most recent commits analysed from the active branch of each project and the 5th shows the language in which the project was developed.

For each system, we extracted its structural dependencies, its logical dependencies and determined the overlap between the two dependencies sets, in various experimental conditions.

One variable experimental condition is whether changes located in comments contribute towards logical dependencies. This condition distinguishes between two different cases:

- with comments: a change in source code files is counted towards a logical dependency, even if the change is inside comments in all files
- without comments: commits that changed source code files only by editing comments are ignored as logical dependencies

[1] http://github.com/

Table 1: Summary of open source projects studied.

ID	Project	Nr. of entites	Nr. of commits	Type
1	bluecove	586	894	java
2	aima-java	987	818	java
3	powermock	1084	893	java
4	restfb	783	1188	java
5	rxjava	2673	2468	java
6	metro-jax-ws	1103	2222	java
7	mockito	1409	1572	java
8	grizzly	1592	3122	java
9	shipkit	242	1483	java
10	OpenClinica	1653	3749	java
11	robolectric	2050	5029	java
12	aeron	541	5101	java
13	antlr4	1381	3449	java
14	mcidasv	805	3668	java
15	ShareX	919	2505	csharp
16	aspnetboilerplate	2353	1615	csharp
17	orleans	3485	3353	csharp
18	cli	767	2397	csharp
19	cake	2250	1853	csharp
20	Avalonia	1677	2445	csharp
21	EntityFramework	7107	2443	csharp
22	jellyfin	2179	4065	csharp
23	PowerShell	861	2033	csharp
24	WeiXinMPSDK	2029	2723	csharp
25	ArchiSteamFarm	117	2181	csharp
26	VisualStudio	1016	4417	csharp
27	CppSharp	259	3882	csharp

In all cases, we varied the following threshold values:

- commit size (cs): the maximum number of files allowed in a commit to be counted as logical dependency. The values for this threshold were 5, 10, 20 and no threshold (infinity).
- number of occurrences (occ): the minimum number of repeated occurrences for a co-change to be counted as logical dependency. The values for this threshold were 1, 2, 3 and 4.

The six tables below present the synthesis of our experiments. We have computed the following values:

- the mean ratio of the number of logical dependencies (LD) to the number of structural dependencies (SD)
- the mean percentage of structural dependencies that are also logical dependencies (calculated from the number of overlaps divided to the number of structural dependencies)
- the mean percentage of logical dependencies that are also structural dependencies (calculated from the number of overlaps divided to the number of logical dependencies)

In all the six tables, 2, 3, 4, 5, 6, 7 we have on columns the values used for the commit size cs, while on rows we have the values for the number of occurrences threshold occ. The tables contain median values obtained for experiments done under all combinations of the two threshold values, on all test systems. In all tables, the upper right corner corresponds to the most relaxed conditions, while the lower left corner corresponds to the most restrictive conditions.

Table 2: Ratio of number of LD to number of SD, case with comments

	$cs \leq 5$	$cs \leq 10$	$cs \leq 20$	$cs < \infty$
$occ \geq 1$	8.02	17.22	33.19	314.6
$occ \geq 2$	4.05	8.90	16.24	274.6
$occ \geq 3$	2.57	5.04	9.92	200.87
$occ \geq 4$	1.81	3.39	6.19	134.8

Table 3: Ratio of number of LD to number of SD, case without comments

	$cs \leq 5$	$cs \leq 10$	$cs \leq 20$	$cs < \infty$
$occ \geq 1$	7.85	16.33	29.78	306.54
$occ \geq 2$	3.93	7.78	15.65	246.08
$occ \geq 3$	2.42	4.91	8.19	115.56
$occ \geq 4$	1.64	3.19	5.47	51.77

Table 4: Percentage of SD that are also LD, case with comments

	$cs \leq 5$	$cs \leq 10$	$cs \leq 20$	$cs < \infty$
$occ \geq 1$	17.11	25.69	37.94	78.11
$occ \geq 2$	9.74	15.52	25.14	68.32
$occ \geq 3$	5.92	10.6	17.75	63.6
$occ \geq 4$	4.87	8.45	13.72	56.52

Table 5: Percentage of SD that are also LD, case without comments

	$cs \leq 5$	$cs \leq 10$	$cs \leq 20$	$cs < \infty$
$occ \geq 1$	16.56	24.77	37.33	75.98
$occ \geq 2$	9.03	15.2	23.85	61.66
$occ \geq 3$	5.92	10.6	15.81	49.03
$occ \geq 4$	4.59	7.41	11.85	37.53

Table 6: Percentage of LD that are also SD, case with comments

	$cs \leq 5$	$cs \leq 10$	$cs \leq 20$	$cs < \infty$
$occ \geq 1$	1.62	1.26	1.06	0.24
$occ \geq 2$	2.72	1.95	1.58	0.31
$occ \geq 3$	3.94	2.27	1.71	0.33
$occ \geq 4$	4.66	2.76	2.42	0.41

Table 7: Percentage of LD that are also SD, case without comments

	$cs \leq 5$	$cs \leq 10$	$cs \leq 20$	$cs < \infty$
$occ \geq 1$	1.62	1.47	1.15	0.28
$occ \geq 2$	2.72	2.1	1.7	0.31
$occ \geq 3$	3.63	2.38	2.3	0.35
$occ \geq 4$	4.66	2.62	2.34	0.53

6 Discussion

This section uses the experimental results to answer the research questions outlined in section 3.

Question 1. Which is the most frequent size for a commit transaction ?

Table 8 presents the size distribution for commit transactions in percentage relative to the total number of commits for each system presented in table 1.The average percentage for commit transactions with less than 5 source code files is 79.7%, on the opposite side are commit transactions with more than 20 source code files which have an average percentage of 5.16%. Based on the results we can say that the majority of the commit transactions have no more than 5 source code files.

Question 2. Is it necessary to set a threshold on the size of commit transactions which are considered to generate valid logical dependencies ? Based on the results from table 9 and table 8 we identify that a small amount of commits as in the case of commit transactions with more than 20 source code files can lead to a vast amount of logical dependencies. The commit transactions with less than 5 files, which are the most frequent types of commits, produce in average only 6.97% of the total logical denedencies extracted from the systems. TBD

Based on the results presented in tables 2 and 3, the number of changed files taken into consideration has an important influence over the ratio of the number of logical dependencies to the number of structural dependencies. If no threshold is set for the number of files in a commit (the cases in the last column in tables 2 and 3) then the number of logical dependencies outnumbers the structural dependencies with a factor of up to 314. The maximum factor is measured in the case when no filtering is done on the number of occurrences (first row). In this case, we can not talk about logical dependencies, but about classes that happened to once change in the same time, by various reasons. The number of pairs of classes that happen to once change in the same time is up to 300 times bigger than the number of pairs of classes presenting structural dependencies.

When filtering is done according to conditions on the number of occurrences, we observe in tables 2 and 3 that the values on the last column still do not fall below 51. This number is still too big to accept for logical dependencies. It is clear that it is necessary to

Table 8: The distribution of commit size in percentage relative to the total number of commits of the system

	$cs \leq 5$	$cs \leq 10$	$cs \leq 20$	$cs < \infty$
1	82.55	10.85	4.14	2.46
2	71.39	13.08	7.82	7.7
3	73.91	13.33	6.27	6.49
4	84.51	8.5	3.11	3.87
5	75.2	11.26	5.92	7.62
6	87.8	6.35	2.57	3.29
7	78.18	11.96	5.73	4.13
8	79.63	9.67	5.77	4.93
9	83.82	9.58	4.18	2.43
10	82.58	9.66	5.31	2.45
11	82.96	8.55	4.89	3.6
12	87.69	8.51	2.96	0.84
13	81.19	8.23	5.54	5.03
14	96.7	1.94	0.71	0.65
15	89.27	7.11	2.17	1.45
16	77.28	12.76	5.51	4.46
17	70.3	12.53	9.48	7.69
18	73.93	12.27	6.63	7.18
19	69.99	14.41	6.91	8.69
20	68.79	10.1	7.44	13.66
21	60.66	17.63	10.04	11.66
22	73.97	12.63	6.94	6.47
23	83.13	6.64	4.18	6.05
24	79.43	8.56	5.66	6.35
25	94.54	3.62	1.1	0.73
26	76.21	9.74	5.84	8.22
27	86.17	8.53	4.12	1.18
Avg	79.7	9.93	5.22	5.16

Table 9: Percentage of LD extracted from different commit size chunks relative to the total number of LD of the system

	$cs \leq 5$	$cs \leq 10$	$cs \leq 20$	$cs < \infty$
1	10.1	2.72	4.08	83.11
2	1.77	1.9	3.55	92.78
3	4.07	5.09	5.98	84.87
4	31.56	7.63	24.68	36.13
5	0.52	1.98	2.15	95.35
6	0.47	0.24	0.5	98.8
7	1.59	1.81	2.56	94.04
8	1.54	2.03	3.76	92.67
9	8.15	7.83	11.85	72.17
10	1.52	2.06	2.49	93.92
11	7.28	8.23	13.72	70.76
12	23.71	21.08	19.93	35.28
13	10.79	20.4	7.9	60.91
14	2.12	0.91	1.3	95.67
15	1.42	1.29	2.13	95.16
16	2.21	2.52	3.12	92.16
17	1.59	1.9	4.66	91.86
18	2.41	3.58	6.63	87.38
19	0.61	0.69	1.54	97.16
20	0.54	0.79	1.56	97.11
21	0.96	0.93	45.01	53.1
22	2.06	3.96	5.81	88.18
23	1.28	1.26	1.03	96.44
24	0.82	0.79	1.67	96.73
25	41.28	17.22	9.06	32.44
26	2.75	3.07	6.49	87.69
27	25.16	18.52	23.65	32.67
Avg	6.97	5.2	8.03	79.8

put a threshold on the number of files accepted in a commit in order to filter out noise.

If we refer to the overlap between structural and logical dependencies, we can see in tables 4 and 5 that the percentage of structural dependencies which are also logical dependencies is as well affected by setting a threshold on the number of files accepted in a commit. When setting a threshold, then we have less logical dependencies overall and one could say that this is what affects also the smaller number of structural dependencies that are also logical dependencies. However, we can see that the percentage of dependencies in the overlap decreases much slower than the total number of logical dependencies. For example, when setting the *cs* threshold at 10, we see in Table 2 that the total number of logical dependencies decreases approx 20 times compared with no threshold. In the same time, we can see in Table 4 that the overlap between the logical and structural dependencies decreases less, only approx 3 times. This confirms the fact that the logical dependencies filtered out were not true dependencies. It is clear that setting a threshold on the maximum number of files accepted in a commit is essential for the quality of finding true logical dependencies.

Question 3. Considering changes only in comments as valid can lead to additional logical dependencies? How many logical dependencies are introduced by considering comment changes as valid changes and in what percentage this can influence the analysis?

In order to assess the influence of comments, we compare pairwise tables 2 and 3, tables 4 and 5 and tables 6 and 7. We observe that, although there are some differences between pairs of measurements done in similar conditions with and without comments, the differences are not significant.

In the case of the ratio of the number of logical dependencies to the number of structural dependencies, from tables 2 and 3 we can see that the maximum difference is for the values from the position of the first row, last column. Without comments, the value of the ratio is 306.54, compared to the value with comments which is 314.6. The decrease represents less than 3% of the value with comments. In the case of the percentage of structural dependencies that are also logical dependencies, from tables 4 and 5, we can see that the maximum difference is also for the values from the first row, last column. Without comments, the overlap is 75.98, compared to the value with comments which is 78.11. The decrease represents also less than 3% of the value with comments. We notice that the differences between the two cases are very small.

Question 4. How many occurrences of a logical dependency are needed to consider it a *valid* logical dependency ?

If we look at consecutive rows in table 2 or in table 3, corresponding to increased threshold values for the number of occurrences, we can roughly say that increasing by 1 the occurrence threshold while maintaining the other conditions reduces with more than half the total number of logical dependencies.

In order to find the appropriate level of filtering out false logical dependencies, we assume as a rule of thumb that the number of logical dependencies should be not bigger that the number of structural dependencies. Choosing the most restrictive combination of thresholds (a commit size threshold of 5 files combined with an occurrence threshold of 4) leads to a number of logical dependencies which comes near to the number of structural dependencies.

Question 5. How does filtering affect the overlap between structural and logical dependencies ?

In order to present overlaps between structural and logical dependencies we will use Venn diagrams, used for this purpose also in (Oliva and Gerosa, 2011) and (Ajienka and Capiluppi, 2017). In Figure 2 we present the intersections of logical and structural dependencies, in a selection of relevant cases. In all cases, the left circle, which is of constant size, represents the set of structural dependencies. The right circle represents the logical dependencies and its area is proportional with the number of logical dependencies. All cases are cases without comments.

In Figure 2, case a.) which corresponds to no filtering, the number of logical dependencies is bigger than the number of structural dependencies by a factor of 314. In this case, we also have the biggest percentage of structural dependencies which are also co-changing as logical dependencies (78%). This case shows that classes with structural dependencies change together more often, but these changes cannot be yet considered logical dependencies.

In Figure 2, case b.) corresponding to the highest level of filtering, the number of logical dependencies is only slightly bigger than the number of structural dependencies by a factor of 1.8. In this case, the percentage of structural dependencies which are also logical is small, as well as the percentage of logical dependencies that are also structural. The percentage of logical dependencies that are also structural is 4.66% in this case. The percentage of logical dependencies that are not structural is 95.34%, and we consider that they deserve to be considered as additional dependencies. We consider that this is the optimal level of filtering.

The percentage of structural dependencies which are also logical is 4.87%, while the percentage of logical dependencies that are also structural is 4.66%. We can see that a percentage of 95.34% of logical dependencies do not correspond with structural dependencies, while 95.13% of the structural dependencies are not doubled by logical dependencies. These percentages are statistically similar with the values obtained in (Oliva and Gerosa, 2011): they measured LCOP (Logical Coupling Only Percentage) and SCOP (Structural Coupling Only Percentage) and obtained 95% for SCOP and 91% for LCOP.

In (Ajienka and Capiluppi, 2017) the authors measured CSD (Co-changes Structural Dependencies ratio) and the CLD (Coupled Logical Dependencies ratio) and obtained the values for CSD about 80% and CLD about 15%. Also in case of (Ajienka and Capiluppi, 2017) the total number of logical dependencies is 10 times bigger than the number of structural dependencies, while in our work, in the case considered of optimal filtering, the total number of logical dependencies can be approximated with the number of structural dependencies. The explanation for this difference lies in the different manner of determining the logical dependencies: while (Ajienka and Capiluppi, 2017) also uses a threshold of 10 for the maximum number of files in a commit, they count all co-changes toward logical dependencies, although they assign them different strengths. This would correspond to the filtering case determined by the combination: the commit size threshold is 10 files and the occurrence threshold is 1.

Tables 10 and 11 relate with table 4 and they detail information about overlaps between logical and structural dependencies. Table 10 gives details about the cases summarized in the first column, while table 11 gives details about the cases summarized in the last column of table 4.

Looking at the detailed tables, we can notice that the values for individual projects may present significant differences from the median. For smaller projects, such as projects with IDs 1, 3, 4, the filtering can lead to loosing logical dependencies.

7 Threats to validity

An issue which has not been investigated enough is whether the reduced set of logical dependencies obtained after filtering contains indeed true logical dependencies. This could be done by manual inspection of the classes in order to validate the reported logical dependencies by the opinion of a human expert. Unfortunately doing such manual validation for all case studies is an impossibly huge task. We have manually inspected the code and code changes from a few of the case studies and the results seem promising. For example, in the case of the project selma, when filtering with both thresholds on commit size and number of occurrences, the tool reduced the initial set of 4751 co-change links identified between classes when no filtering was done, to a number of just 25 logical dependencies. Out of these 25 logical dependencies, 5 are doubled by structural dependencies. From the 20 remaining logical dependencies identified by the tool, we determined by manual inspection that 18 are logical, while for 2 of them we could not see a logical reason for a dependency relationship. The

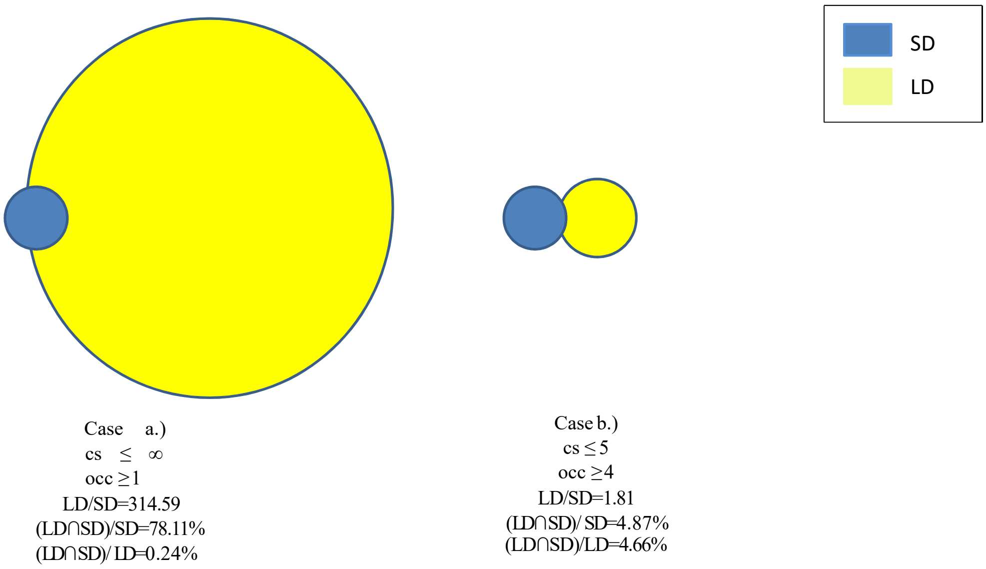

Figure 2: Intersections of logical and structural dependencies, in different cases defined by different combinations of filtering thresholds.

18 class pairs where we could confirm by manual inspection a logical link are: ('FieldItem', 'MapperMethodGenerator'), ('CollectionsMapperIT', 'MappingBuilder'), ('MapperMethodGenerator', 'MethodWrapper'), ('MapperMethodGenerator', 'MappingSourceNode'), ('MapperMethodGenerator', 'MappingBuilder'), ('MapperMethodGenerator', 'BeanWrapper'), ('MapperMethodGenerator', 'InOutType'), ('MapperMethodGenerator', 'FailingMappersIT'), ('MapperMethodGenerator', 'FailingMissingPropertyMapsMappersIT'), ('MapperMethodGenerator', 'MapperWrapper'), ('MapperMethodGenerator', 'MappingRegistry'), ('CustomMapperWrapper', 'FactoryWrapper'), ('CustomMapperWrapper', 'MappingBuilder'), ('CustomMapperWrapper', 'MapperClassGenerator'), ('MappingRegistry', 'MapperClassGenerator'), ('InOutType', 'MappingBuilder'), ('MappingBuilder', 'MappingSourceNode'), ('FactoryWrapper', 'MapperClassGenerator').

We consider that in the future, the validation of extracted logical dependencies will occur by using them to enhance dependency graphs for applications such as architectural reconstruction through clustering (Şora et al., 2010) or finding of key classes (Şora, 2015), and evaluating the positive impact on their results.

As we could see in tables 6 and 7, only a small amount of logical dependencies are between classes that also present structural dependencies. In our experiments, even after filtering, around 95% of the logical dependencies are between classes without structural dependencies. Although this big percentage is supported also by experiments of related works, we consider that future work must further investigate its cause. One possible cause could be that some of the co-changes were legit logical dependencies at some moment in the past, maybe even doubled by structural dependencies in previous revisions, but in the mean time the problem causing them may have been refactored and they should not be added to the dependency model of the current system.

In this work we have extracted logical dependencies from all the revisions of the system, and structural dependencies from the last revision of the system. In future work we will take into account also structural dependencies from all the revisions of the system, in order to filter out the old, out-of-date logical dependencies. Some logical dependencies may have been also structural in previous revisions of the system but not in the current one. If we take into consideration also structural dependencies from previous revisions then the overlapping rate between logical and structural dependencies could probably increase. Another way to investigate this problem could be to study the trend of occurrencies of co-changes: if co-changes

Table 10: Percentage of SD that are also LD, when $cs \leq 5$, for different threshold values for *occ*

ID	$occ \geq 1$	$occ \geq 2$	$occ \geq 3$	$occ \geq 4$
1	18,48	9,74	5,73	4,87
2	9,52	4,31	2,13	1,82
3	19,92	8,79	4,67	2,75
4	53,69	43,14	28,38	26,05
5	9,17	6,16	3,31	2,31
6	12,57	8,66	5,31	4,33
7	18,77	13,65	9,34	6,9
8	19,97	11,94	7,94	5,87
9	53,85	40,11	24,73	17,58
10	8,03	5,45	3,15	2,27
11	40,93	31,37	25,68	21,97
12	35,91	24,66	17,88	13,49
13	17,11	10,79	7,5	5,88
14	39,87	28,6	18,67	15,31
15	16,27	6,44	3,73	3,39
16	14,8	7,85	4,25	3,27
17	10,33	5,19	3,1	2,36
18	3,86	2,15	0,86	0
19	3,55	2,43	0,93	0,75
20	14,78	8,09	4,46	3,63
21	16,56	10,67	5,92	4,59
22	9,5	5,72	3,17	1,43
23	21,76	16,79	14,89	13,36
24	10,72	8,56	7,11	6,08
25	62,5	62,5	62,5	62,5
26	25,14	18,78	14,92	11,33
27	51,3	30,43	26,96	24,35
M	17,10	9,74	5,92	4,87

Table 11: Percentage of SD that are also LD, when $cs < \infty$, for different threshold values for *occ*

ID	$occ \geq 1$	$occ \geq 2$	$occ \geq 3$	$occ \geq 4$
1	77,51	54,44	46,7	31,38
2	82,44	55,71	32,06	20,94
3	71,02	50	32,28	24,18
4	98,52	90,93	87,55	86,18
5	91,55	81,06	72,8	65,17
6	91,06	84,78	75,7	65,78
7	87,73	68,24	49,63	35,66
8	97,57	96,42	77,85	65,26
9	90,66	87,91	80,77	73,63
10	29,1	22,96	17,6	13,29
11	72,68	64,48	56,17	49,67
12	70,07	59,22	47,73	37,35
13	56,56	46,9	40,26	33,94
14	97,73	95,04	94,2	85,95
15	97,63	95,93	94,24	64,41
16	46,03	39,33	28,21	20,61
17	72,64	51,02	37,69	30,3
18	78,11	75,54	67,81	53,65
19	93,08	91,4	88,22	80,56
20	71,13	65,97	63,6	59,83
21	77,09	54,96	41,43	32,33
22	52,71	51,48	35,14	31,36
23	98,09	90,08	86,26	69,08
24	77,63	75,88	65,57	63,81
25	100	100	100	100
26	71,82	67,4	67,4	65,19
27	80,87	68,7	59,13	56,52
M	78,11	68,23	63,59	56,52

between a pair of classes used to happen more often in the remote past than in the more recent past, it may be a sign that the problem causing the logical coupling has been removed in the mean time.

8 Conclusion

In this work we experimentally define methods to filter out the most relevant logical dependencies from co-changing classes.

Regarding whether co-changes which affect only comments contribute to logical dependencies, our study has found out that when ignoring commits which change only comments in source files, the quality of logical dependencies is slightly improved.

Our experiments show that the most important factors which affect the quality of logical dependencies are: the maximum number of files allowed in a commit to be counted as logical dependency, and the minimum number of repeated occurrences for a co-change to be counted as logical dependency.

REFERENCES

Ajienka, N. and Capiluppi, A. (2017). Understanding the interplay between the logical and structural coupling of software classes. *Journal of Systems and Software*, 134:120–137.

Ajienka, N., Capiluppi, A., and Counsell, S. (2018). An empirical study on the interplay between semantic coupling and co-change of software classes. *Empirical Software Engineering*, 23(3):1791–1825.

Beck, F. and Diehl, S. (2011). On the congruence of modularity and code coupling. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ESEC/FSE '11, pages 354–364, New York, NY, USA. ACM.

Collard, M. L., Decker, M. J., and Maletic, J. I. (2011). Lightweight transformation and fact extraction with the srcML toolkit. In *Proceedings of the 2011 IEEE 11th International Working Conference on Source Code Analysis and Manipulation*, SCAM '11, pages 173–184, Washington, DC, USA. IEEE Computer Society.

Collard, M. L., Kagdi, H. H., and Maletic, J. I. (2003). An XML-based lightweight C++ fact extractor. In *Proceedings of the 11th IEEE International Workshop on Program Comprehension*, IWPC '03, pages 134–, Washington, DC, USA. IEEE Computer Society.

Collins-Sussman, B., Fitzpatrick, B. W., and Pilato, C. M. (2010). *Version Control With Subversion for Subversion 1.6: The Official Guide And Reference Manual*. CreateSpace, Paramount, CA.

Şora, I. (2015). Helping program comprehension of large software systems by identifying their most important classes. In *Evaluation of Novel Approaches to Software Engineering - 10th International Conference, ENASE 2015, Barcelona, Spain, April 29-30, 2015, Revised Selected Papers*, pages 122–140. Springer International Publishing.

Şora, I., Glodean, G., and Gligor, M. (2010). Software architecture reconstruction: An approach based on combining graph clustering and partitioning. In *Computational Cybernetics and Technical Informatics (ICCC-CONTI), 2010 International Joint Conference on*, pages 259–264.

Ducasse, S. and Pollet, D. (2009). Software architecture reconstruction: A process-oriented taxonomy. *IEEE Transactions on Software Engineering*, 35(4):573–591.

Gall, H., Hajek, K., and Jazayeri, M. (1998). Detection of logical coupling based on product release history. In *Proceedings of the International Conference on Software Maintenance*, ICSM '98, pages 190–, Washington, DC, USA. IEEE Computer Society.

Kagdi, H., Gethers, M., Poshyvanyk, D., and Collard, M. L. (2010). Blending conceptual and evolutionary couplings to support change impact analysis in source code. In *2010 17th Working Conference on Reverse Engineering*, pages 119–128.

Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D. M., and Damian, D. (2016). An in-depth study of the promises and perils of mining github. *Empirical Software Engineering*, 21(5):2035–2071.

Oliva, G. A. and Gerosa, M. A. (2011). On the interplay between structural and logical dependencies in open-source software. In *Proceedings of the 2011 25th Brazilian Symposium on Software Engineering*, SBES '11, pages 144–153, Washington, DC, USA. IEEE Computer Society.

Oliva, G. A. and Gerosa, M. A. (2015). Experience report: How do structural dependencies influence change propagation? an empirical study. In *26th IEEE International Symposium on Software Reliability Engineering, ISSRE 2015, Gaithersbury, MD, USA, November 2-5, 2015*, pages 250–260.

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

Ren, X., Ryder, B. G., Stoerzer, M., and Tip, F. (2005). Chianti: a change impact analysis tool for java programs. In *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005.*, pages 664–665.

Shtern, M. and Tzerpos, V. (2012). Clustering methodologies for software engineering. *Adv. Soft. Eng.*, 2012:1:1–1:1.

Sora, I. (2013). Software architecture reconstruction through clustering: Finding the right similarity factors. In *Proceedings of the 1st International Workshop in Software Evolution and Modernization - Volume 1: SEM, (ENASE 2013)*, pages 45–54. INSTICC, SciTePress.

Wiese, I. S., Kuroda, R. T., Re, R., Oliva, G. A., and Gerosa, M. A. (2015). An empirical study of the relation between strong change coupling and defects using history and social metrics in the apache aries project. In Damiani, E., Frati, F., Riehle, D., and Wasserman, A. I., editors, *Open Source Systems: Adoption and Impact*, pages 3–12, Cham. Springer International Publishing.

Yu, L. (2007). Understanding component co-evolution with a study on linux. *Empirical Software Engineering*, 12(2):123–141.

Zimmermann, T., Weisgerber, P., Diehl, S., and Zeller, A. (2004). Mining version histories to guide software changes. In *Proceedings of the 26th International Conference on Software Engineering*, ICSE '04, pages 563–572, Washington, DC, USA. IEEE Computer Society.