A good title

Adelina Diana Stana and Ioana Şora
Department of Computer and Information Technology,
Politehnica University Timisoara, Romania

***Abstract*—bla some abstract**

I. Introduction

bla some intro

II. State of the art

State of the art has already established that "logical dependencies" exist and they are determined by the co-evolution of classes [1].

Their primary usage was to improve techniques for Software Change Impact Analysis, for identifying the potential ripple effects caused by software changes during software maintenance and evolution, or for their link to deffects [2]. They also have been used in tools that predict and recommend necessary changes [3].

The current trend recommends [4], [5] that dependency management methods and tools also include these kind of dependencies besides the structural ones. Applications based on dependency analysis, such as software architecture reconstruction, could also be improved by taking into account the hidden dependencies that exist beyond structural dependencies. However, a thorough survey [6] shows that historical information are rarely used. Another survey [7] mentions one explanation of the reduced use of historical information as the size of the extracted information. Below, we will analyze the state of the art results for determining logical dependencies from the point of view of their sizes.

There are researches that investigated quantitative aspects of logical dependencies and their interplay with structural dependencies. Oliva and Gerosa [4], [8] have found 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 [5] 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 [5] 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 could calculate the ratio between the number of logical dependencies and the number of structural dependencies for all the projects analyzed by them, and the average ratio resulted 12. This means that, using their method of detecting logical dependencies for a system, the number of logical dependencies outnumbers with one order of magnitude the number of structural dependencies.

Another kind of non-structural dependencies are the semantic or conceptual dependencies [9], [10]. 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 [11]. 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 arise the question whether it is a legitimate approach to accept all co-evolving classes as logical coupling.

Changes made to two components at the same commit do not necessarily indicate the co-evolution of the two. These changes could be completely unrelated. The study [12] 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 [3] 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 be able to use logical dependencies in architectural reconstruction analysis, logical dependencies must be filtered until they remain only a reduced but relevant set of true logical dependencies. In this work, we explore several ways of filtering logical dependencies.

III. Research questions

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 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. How the number of source files changed in a commit can influence the logical dependencies of the system and the overlapping rates with the structural dependencies.
Motivation: A commit that has as participants a big number of files 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 number of files in a commit. Cappiluppi and Ajienka, in their works [5], [11] only take into consideration commits with less then 10 source code files changed in building the logical dependencies. The research of Beck et al [13] only takes in consideration transactions with up to 25 files. The research [4] 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.

Question 2. Considering comments 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 final result ?
Motivation: Not all the commits that have source code files changed include code changes , some of them can be only comments changes. Regarding this aspect, we can consider that there is no logical dependency between two classes that change in the same time only by comments changes . Some studies have not taken this aspect into consideration, so we will analyse the impact of not considering/ considering comments as valid changes on the results.

Question 3. One occurrence of a logical dependency is enough to consider it as valid ? If we consider only logical dependencies with more then one occurrence as valid, the results are influenced in a significant way ?
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.
But 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.

In order to answer these research questions, we have built a tool that extracts structural and logical dependencies on different scenarios, the design and implementation of the tool is presented in section IV.

We have analyzed 19 open-source software systems of different sizes within the tool developed, the experimental results obtained being presented in section **??** and discussed in section VI.

IV. 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.*

A. Extracting structural dependencies

Even though in some of the cases if class A depends on class B, changes in class B can produce changes in class A, but not the other way around [12] . There are some cases in which if class A depends on class B, changes in class B can produce changes in class A and viceversa. So we will consider structural dependencies as bidirectional relationships, ”class A depends on class B” and ”class B depends on class A”. The choice of building bidirectional relationships is also motivated by the fact that we cannot establish for the moment the direction of the logical dependencies of the system. So in order to have a omogenity between the logical and structural dependencies analysis results, we will take both of the relationships types as bidirectional. In this step the entire source code folder is scanned and only source code files are extracted in order to convert them into XML files through calls to an external tool called srcML [14], [15]. All the information about classes, methods, calls to other classes are afterwards extracted from the XML files.

B. Extracting logical dependencies

The versioning system contains the long-term change history of every file. Each project change made by an individual at certain point of time is contained into a commit [16]. All the commits are stored in the versioning system cronologicaly and each commit has a parent. The parent commit is the baseline from which development began, the only exeption 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 [5]. 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.
Finally after all the differences files are stored , all the files are parsed and logical dependencies are build. In addition , the

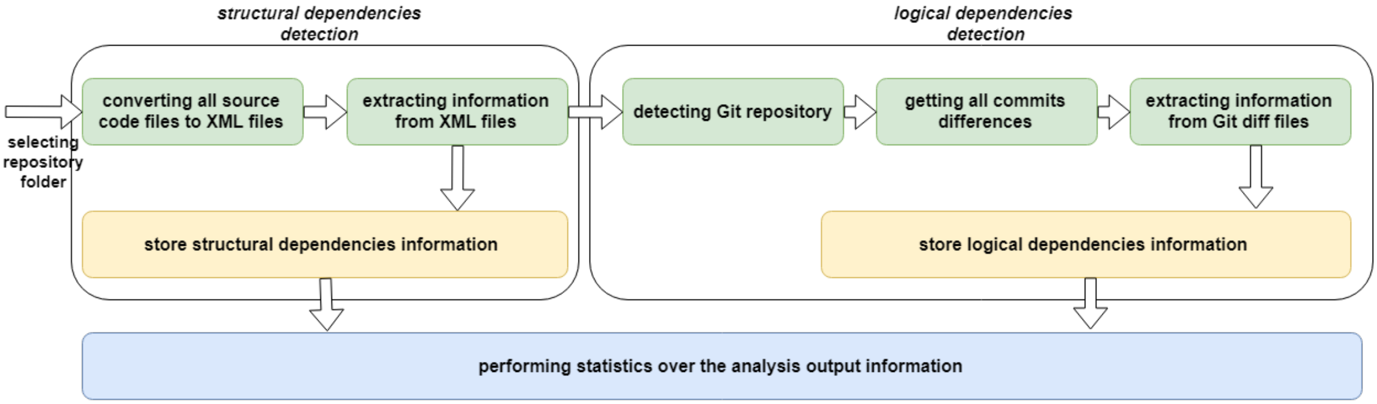

Fig. 1. Processing phases

number of files changed in a commit can influence the logical dependencies. A relatively big number of files changed can indicate a merge of all changes from another branch as a single commit. This can lead to a number of logical dependencies that are redundant since the files are not actually changing in the same time.The logical dependencies are splitted into three categories :

Category 1: *Dependencies found in commits with less than 5 source code files changed.*

Category 2: *Dependencies found in commits with more than 5 files changed but less than 20.*

Category 3: *Dependencies found in commits with more than 20 files.*

Also for each category two dependencies analysis will be made: ***A:*** *Considering comments as valid changes.* ***B:*** *Considering comments as redundant changes.* In the second case if class A and class B change together but the only change found is a comment change then between class A and B will not be set logical dependency.

V. Experimental results

In this study, we have made a set of statistical analysis on a set of open-source projects in order to extract the structural and logical dependencies between classes. Table I illustrates all the systems studied. The 1st column shows the projects IDs; 2nd column shows the project name; 3rd column shows the number of classes 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.

Table II, illustrates results for commits with less than 5 files changed. The 1st column shows the projects IDs; 2nd column shows the number of structural dependencies; 3rd column shows the number logical dependencies found with comments taken into consideration as change; 4th column shows the number of logical dependencies found in col. 3 that are also structural dependencies; 5th column shows logical dependencies found without taking into consideration comments as change; finally the 6th column shows the number of logical dependencies found in col.5 that are also structural dependencies.

ID	Project	Nr. of classes	Nr. of commits	Type
1	urSQL	39	89	java
2	prettyfaces	257	207	java
3	jbal	102	113	java
4	guavatools	209	85	java
5	monome-pages	196	280	java
6	kryo	289	743	java
7	slema	267	368	java
8	bluecove	386	1679	java
9	aima-java	818	1181	java
10	powermock	803	1512	java
11	restfb	713	1545	java
12	rxjava	2251	2468	java
13	metro-jax-ws	365	2222	java
14	mockito	1121	1572	java
15	grizzly	1170	3122	java
16	shipkit	222	1483	java
17	Tensorflow	1104	2386	cpp

TABLE I
SUMMARY OF OPEN SOURCE PROJECTS STUDIED.

ID	SD	LD+comments	Overlaps	LD-comments	Overlaps
1	52	59	15	49	12
2	264	21	5	19	5
3	106	27	2	27	2
4	138	89	19	84	19
5	250	239	40	217	38
6	566	1576	129	1488	126
7	358	223	37	200	34
8	447	687	61	619	58
9	1463	1063	101	963	86
10	466	1052	73	932	68
11	832	1529	297	1373	286
12	2557	1172	32	1107	31
13	154	488	10	417	10
14	541	2360	132	2246	131
15	2698	2620	335	2341	312
16	138	1519	64	1406	61
17	293	1569	46	1539	45

TABLE II
RESULTS FOR COMMITS WITH LESS THAN 5 FILES CHANGED

ID	SD	LD ¡=5	LD ¡=10	LD¡=20	No limit
1	52	59	145	288	415
2	264	21	21	76	76
3	106	27	57	231	5570
4	138	89	210	598	1023
5	250	239	824	1593	4635
6	566	1576	2548	4217	22437
7	358	223	1051	1756	6845
8	447	687	1421	2308	32612
9	1463	1063	2640	6257	156710
10	466	1052	2693	5696	42726
11	832	1529	2604	4184	32133
12	2557	1172	3575	9319	577118
13	154	488	940	1811	55837
14	541	2360	5871	9689	182276
15	2698	2620	6773	16058	218476
16	138	1519	3584	6233	22145
17	293	1569	3253	5667	32347
Overlapping Avg LD with SD		13,76	23,26	36,04	66,48

TABLE III
LOGICAL DEPENDENCIES FOR DIFFERENT TYPES OF THRESHOLDS, CASE WITH COMMENTS

ID	1 link	2 links	3 links
1	28,85	15,38	1,92
2	1,89	0,76	0,38
3	1,89	0,94	0,00
4	13,77	2,17	1,45
5	16,00	8,00	3,20
6	22,79	12,54	4,24
7	10,34	3,63	0,56
8	13,65	5,82	2,68
9	6,90	1,78	0,27
10	15,67	2,58	1,07
11	35,70	26,56	9,38
12	1,25	0,82	0,35
13	6,49	2,60	1,30
14	24,40	14,05	6,28
15	12,42	4,11	2,04
16	46,38	29,71	14,49
17	15,70	9,56	5,46
Avg	13,76	4,11	1,92

TABLE V
OVERLAPPING RESULTS IN PERCENTAGE FOR LOGICAL DEPENDENCIES EXTRACTED FROM COMMITS WITH LESS THAN 5 FILES CHANGED, SPLITTED BY NUMBERS OF OCCURRENCES

ID	SD	LD ¡=5	LD ¡=10	LD ¡=20	No limit
1	52	49	121	257	319
2	264	19	19	74	74
3	106	27	33	171	5553
4	138	84	194	566	991
5	250	217	712	1327	4004
6	566	1488	2307	3928	20396
7	358	200	918	1502	4751
8	447	619	1255	2066	31879
9	1463	963	2374	5632	149531
10	466	932	2399	4729	35846
11	832	1373	2305	3618	28401
12	2557	1107	3340	7948	333585
13	154	417	758	1407	51894
14	541	2246	5424	8504	148053
15	2698	2341	5716	12486	178262
16	138	1406	3161	5475	20215
17	293	1539	3195	5578	29720
Overlapping Avg LD with SD		13,76	22,14	34,38	63,95

TABLE IV
LOGICAL DEPENDENCIES FOR DIFFERENT TYPES OF THRESHOLDS, CASE WITHOUT COMMENTS

ID	1 link	2 links	3 links
1	76,92	46,15	21,15
2	1,89	0,76	0,38
3	85,85	84,91	83,96
4	70,29	15,22	10,87
5	69,60	58,40	44,80
6	65,55	42,05	30,57
7	66,48	32,68	18,99
8	64,88	43,40	20,58
9	75,32	58,58	41,90
10	57,73	30,69	19,53
11	81,49	64,66	37,98
12	14,74	8,80	5,91
13	33,12	9,74	7,79
14	67,28	45,84	26,80
15	52,85	33,80	22,65
16	80,43	71,01	55,07
17	24,57	18,43	17,06
Avg	66,48	42,04	21,15

TABLE VI
OVERLAPPING RESULTS IN PERCENTAGE FOR LOGICAL DEPENDENCIES EXTRACTED FROM ALL COMMITS, SPLITTED BY NUMBERS OF OCCURRENCES

VI. DISCUSSION

This section presents the results of the three analyses, performed on the selected projects (I). The purpose is to answer the three research questions outlined in section III.

Question 1. How the number of source files changed in a commit can influence the logical dependencies of the system and the overlapping rates with the structural dependencies.

Based on the results presented in tables III and IV, from section V, the number of changed files taken into consideration has an important influence over the overall rates of the logical and structural dependencies overlaps. If no threshold is set for the number of files then we can affirm that a significant number of structural dependencies are also logical. In table III we have obtained an overlap of structural and logical dependencies of 63,95% which is with 41,81% more than if consider only commits with less then 5 source code files changed per commit (Figure 2) and with 22,14% more if consider only commits with more then 5 and less then 20 source code files changed per commit.

Question 2. Considering comments 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 final result ?

Table VII illustrates the percentages rates extracted from tables III and IV from section V. As it was specified in the tables description , the percentages rates are the number of structural and logical dependencies overlaps, reported to the total number of structural dependencies. How it can be seen, the overlapping rates are also influenced by the comments

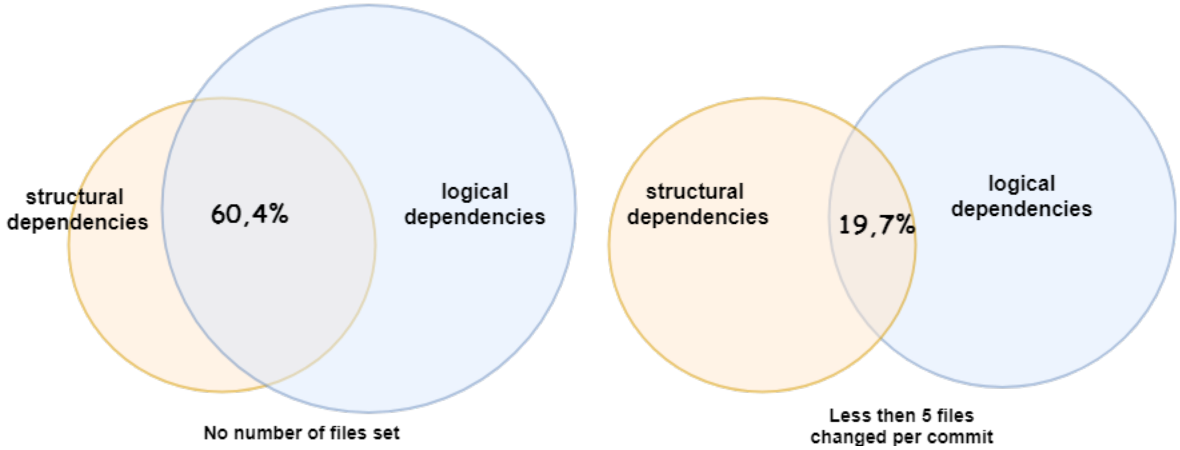

Fig. 2. Venn diagrams of the overlapping rates with comments taken into consideration as change

Category	With comments	Without comments
less 5	13,76%	13,76 %
more 5 less 20	23,26%	22,14%
more 20	36,04%	34,38%
total	66,48%	63,95%

TABLE VII

AVERAGE PERCENTAGES RATES WITH AND WITHOUT CONSIDERING COMMENTS AS VALID CHANGES.

Category	With comments	Without comments
less 5	9,09%	8,67%
more 5 less 20	15,87%	13,29%
more 20	18,8%	16,52%
total	39,1%	35,38%

TABLE VIII

OVERALL PERCENTAGES RATES BY FILTERING THE LOGICAL DEPENDENCIES OCCURRENCES.

filtering but in a small percentage. The rates are with aprox 2% lower if comments are not taken into consideration as a change. (Figure 3)

Question 3. One occurrence of a logical dependency is enough to consider it as valid ? If we consider only logical dependencies with more then one occurrence as valid, the results are influenced in a significant way ?

Table V and VI from chapter Experimental Results, illustrates results in percentages, reported to the structural dependencies, of the analysis for all the systems when logical dependencies where build with/ without comments taken into consideration as change and multiple occurrences of logical dependencies taken into consideration as valid dependencies. Based on the experimental results averages VIII , we can affirm that the results are with approx 50% lower after filtering VII. This indicates that a lot of logical dependencies are the result of a single commit in which the two elements of the dependency where changed together.

If we look at the average rates of projects 13, 15, 16, 18 we can observe that the difference is much smaller (15-20%), the only thing that those projects have in common is the size and the number of commits compared to the other ones from the list, the size and the number of commits of the projects mentioned above are relatively big.

This can lead to the conclusion that, if the project studied has a relatively small amount of commits (Project ID 8.), 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. (Figure 4)

As a conclusion, it results that large number of structural dependencies are doubled by logical or not, this number is particularly influenced by the number of files that participate in a commit that taken into consideration. It also results that taken or not comments as change, the final results are not influenced in a big percentage.

In this research work we have tried to identify methods to acquire the most relevant logical dependencies from the system so that can be used in the future for architectural reconstruction, that is currently based only on the information provided by structural dependencies.

For future work, we will investigate the cause for the large number of logical dependencies which are not overlapping with structural dependencies. As we can see in tables **??** and **??**, where the percentages are reported to the logical dependencies, only a small amount (aprox 10%) of logical dependencies are also structural .

In this work we have extracted structural dependencies from the last revision of the system but logical dependencies from all the revisions of the system. We will study also structural dependencies from all the revisions of the system since some logical dependencies may have been also structural on previos 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 will probably increase.

VII. THREATS TO VALIDITY

bla threats

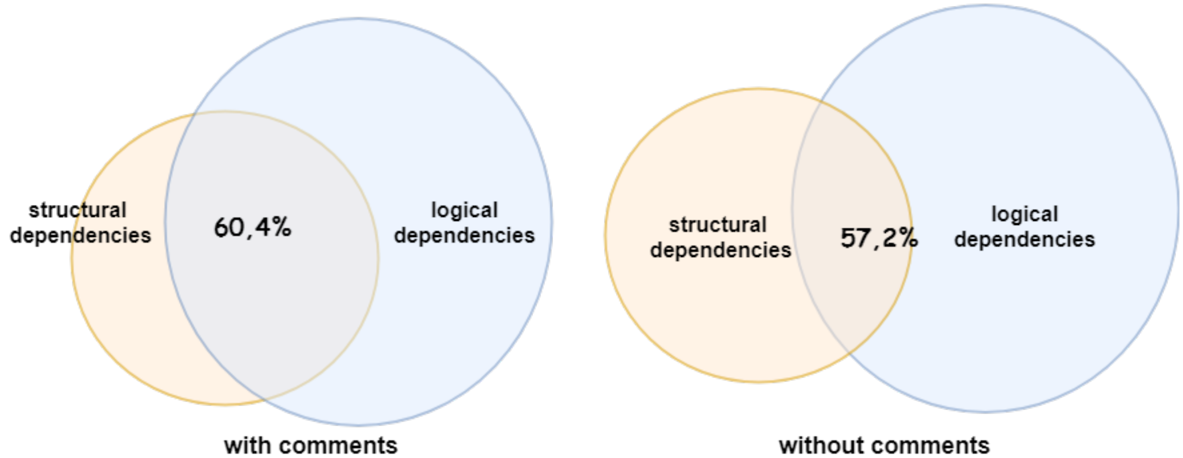

Fig. 3. Venn diagrams of the overlapping rates with comments and without comments taken into consideration.

VIII. Conclusion

bla final concl

References

[1] H. Gall, K. Hajek, and M. Jazayeri, "Detection of logical coupling based on product release history," in *Proceedings of the International Conference on Software Maintenance*, ser. ICSM '98. Washington, DC, USA: IEEE Computer Society, 1998, pp. 190–. [Online]. Available: http://dl.acm.org/citation.cfm?id=850947.853338

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

[3] T. Zimmermann, P. Weisgerber, S. Diehl, and A. Zeller, "Mining version histories to guide software changes," in *Proceedings of the 26th International Conference on Software Engineering*, ser. ICSE '04. Washington, DC, USA: IEEE Computer Society, 2004, pp. 563–572. [Online]. Available: http://dl.acm.org/citation.cfm?id=998675.999460

[4] G. A. Oliva and M. A. Gerosa, "On the interplay between structural and logical dependencies in open-source software," in *Proceedings of the 2011 25th Brazilian Symposium on Software Engineering*, ser. SBES '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 144–153. [Online]. Available: https://doi.org/10.1109/SBES.2011.39

[5] N. Ajienka and A. Capiluppi, "Understanding the interplay between the logical and structural coupling of software classes," *Journal of Systems and Software*, vol. 134, pp. 120–137, 2017. [Online]. Available: https://doi.org/10.1016/j.jss.2017.08.042

[6] S. Ducasse and D. Pollet, "Software architecture reconstruction: A process-oriented taxonomy," *IEEE Transactions on Software Engineering*, vol. 35, no. 4, pp. 573–591, July 2009.

[7] M. Shtern and V. Tzerpos, "Clustering methodologies for software engineering," *Adv. Soft. Eng.*, vol. 2012, pp. 1:1–1:1, Jan. 2012. [Online]. Available: http://dx.doi.org/10.1155/2012/792024

[8] G. A. Oliva and M. A. Gerosa, "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*, 2015, pp. 250–260. [Online]. Available: https://doi.org/10.1109/ISSRE.2015.7381818

[9] 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, Feb 2009. [Online]. Available: https://doi.org/10.1007/s10664-008-9088-2

[10] H. Kagdi, M. Gethers, D. Poshyvanyk, and M. L. Collard, "Blending conceptual and evolutionary couplings to support change impact analysis in source code," in *2010 17th Working Conference on Reverse Engineering*, Oct 2010, pp. 119–128.

[11] N. Ajienka, A. Capiluppi, and S. Counsell, "An empirical study on the interplay between semantic coupling and co-change of software classes," *Empirical Software Engineering*, vol. 23, no. 3, pp. 1791–1825, 2018. [Online]. Available: https://doi.org/10.1007/s10664-017-9569-2

[12] L. Yu, "Understanding component co-evolution with a study on linux," *Empirical Software Engineering*, vol. 12, no. 2, pp. 123–141, Apr 2007. [Online]. Available: https://doi.org/10.1007/s10664-006-9000-x

[13] F. Beck and S. Diehl, "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*, ser. ESEC/FSE '11. New York, NY, USA: ACM, 2011, pp. 354–364. [Online]. Available: http://doi.acm.org/10.1145/2025113.2025162

[14] M. L. Collard, H. H. Kagdi, and J. I. Maletic, "An XML-based lightweight C++ fact extractor," in *Proceedings of the 11th IEEE International Workshop on Program Comprehension*, ser. IWPC '03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 134–. [Online]. Available: http://dl.acm.org/citation.cfm?id=851042.857028

[15] M. L. Collard, M. J. Decker, and J. I. Maletic, "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*, ser. SCAM '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 173–184. [Online]. Available: https://doi.org/10.1109/SCAM.2011.19

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

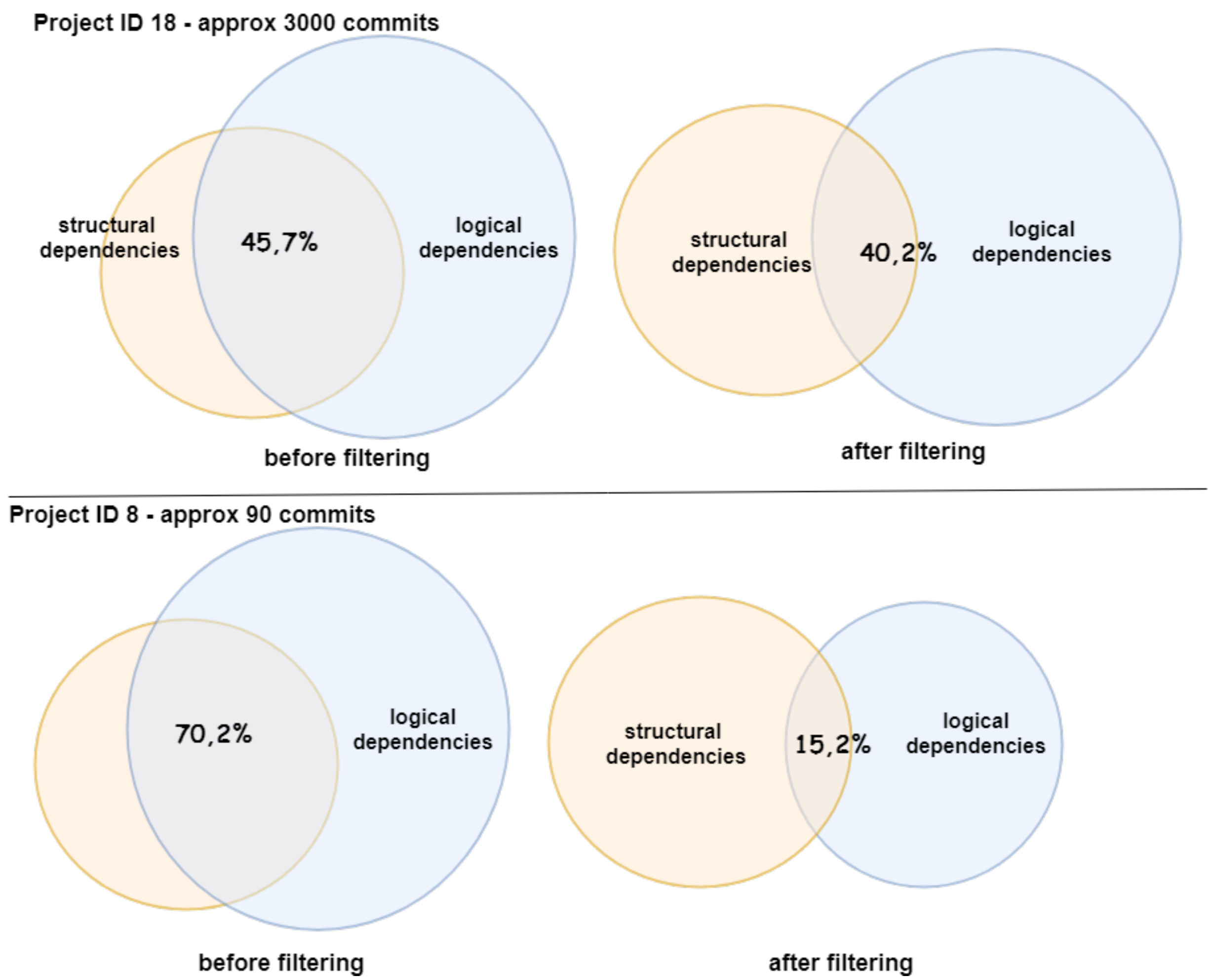

Fig. 4. Impact of logical dependencies occurrences filtering on projects with different sizes.