# Logical dependencies between classes: how to find them and how to use them ?

Adelina Diana Stana
student - early pre-doctoral track
Department of Computer and Information Technology,
Politehnica University of Timisoara, Romania

Ioana Şora and Vladimir Creţu
supervisors
Department of Computer and Information Technology,
Politehnica University of Timisoara, Romania

***Abstract*—Emerging software engineering approaches support the idea that general methods and tools for dependency management should take into account not only structural dependencies but also logical dependencies. In this work we try to identify a set of factors that can be used to filter the recordings of class co-changes such that true logical dependencies are identified. Also we analyze the impact of filtering factors over the relationships between the sets of logical and structural dependencies and their intersection and differences. We present preliminary results obtained through an experimental study on a set of open source software projects with their historical evolution, and outline additional factors which need to be investigated in future work.**

## I. Introduction

The current trend recommends [1], [2], [3] that general dependency management methods and tools should also include logical dependencies besides the structural dependencies that are extracted by analyzing the source code. Gall [4] identified as logical coupling between two modules the fact that these modules repeatedly change together during the historical evolution of the software system.

The concepts of logical coupling and logical dependencies were first used in different analysis tasks, all related to changes: for software change impact analysis [5], for identifying the potential ripple effects caused by software changes during software maintenance and evolution [6], [1], [7], [8] or for their link to deffects [9], [10]. But, 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.

## II. Research questions

We investigated through three questions [11], [12] the factors that could be used to filter logical dependencies:

***Question 1***. How does the maximum number of source files accepted to change in a commit influence the logical dependencies of the system ?

*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 accepted in a commit: [2], [13], [14].

***Question 2***. Considering changes in comments only as valid can lead to additional logical dependencies?

*Motivation*: We consider that there is probably no logical dependency between two classes that change in the same time only by comments changes. 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 3***. 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. 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.

## III. 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 of the tool is presented in Figure 1.

### A. *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 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.

We use an external tool called srcML [15], [16] 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#.

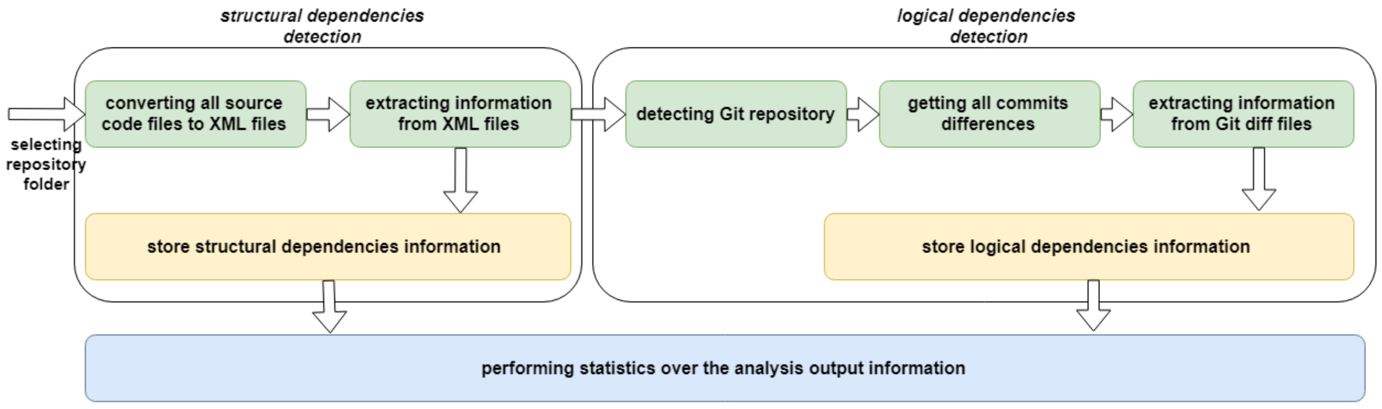
structural dependencies detection
logical dependencies detection
selecting repository folder
converting all source code files to XML files
extracting information from XML files
detecting Git repository
getting all commits differences
extracting information from Git diff files
store structural dependencies information
store logical dependencies information
performing statistics over the analysis output information

Fig. 1. Processing phases

## B. *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 [17], [2].

The tool looks through the main branch of the project and gets all the existing commits. For each commit a diff against the parent commit is 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 a occurrence counter for the logical link.

## IV. Experimental results

We have analysed a set of open-source projects found on GitHub[1] in order to extract the structural and logical dependencies between classes [18]. Table I enumerates all the systems studied.

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 in comments only count as logical dependencies:

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

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).

[1]http://github.com/

TABLE I
Summary of open source projects studied.

| 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 |

- 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.

In table II, we have on columns the values used for the $cs$, while on rows we have the values for the $occ$. The table contains median values obtained for experiments done under all combinations of the two threshold values. In all the subtables the upper right corner corresponds to the most relaxed conditions, while the lower left corner corresponds to the most restrictive filtering conditions.

## V. Discussion

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

***Question 1***. How does the maximum number of source files accepted to change in a commit influence the logical dependencies of the system ?

Based on the results presented in table II, the number of changed files taken into consideration has an important influence over the percentages. Tables III and IV present the

TABLE II
MEDIAN VALUES OBTAINED FOR EXPERIMENTS DONE

| | $cs \leq 5$ | $cs \leq 10$ | $cs \leq 20$ | $cs < \infty$ |
|---|---|---|---|---|
| A. Percentage of SD that are also LD, case with comments | | | | |
| $occ \geq 1$ | 13.77 | 23.27 | 36.04 | 66.48 |
| $occ \geq 2$ | 4.11 | 9.90 | 15.79 | 42.05 |
| $occ \geq 3$ | 1.92 | 5.82 | 8.41 | 21.15 |
| $occ \geq 4$ | 0.85 | 2.68 | 5.80 | 13.46 |
| B. Percentage of SD that are also LD, case without comments | | | | |
| $occ \geq 1$ | 13.77 | 22.15 | 34.38 | 63.96 |
| $occ \geq 2$ | 3.67 | 9.17 | 12.53 | 34.62 |
| $occ \geq 3$ | 1.82 | 4.03 | 6.83 | 17.60 |
| $occ \geq 4$ | 0.67 | 2.34 | 4.47 | 11.60 |
| C. Percentage of LD that are also SD, case with comments | | | | |
| $occ \geq 1$ | 9.40 | 7.81 | 5.53 | 0.89 |
| $occ \geq 2$ | 20.97 | 16.50 | 7.41 | 1.97 |
| $occ \geq 3$ | 18.82 | 18.53 | 13.73 | 2.10 |
| $occ \geq 4$ | 27.39 | 21.59 | 18.60 | 2.88 |
| D. Percentage of LD that are also SD, case without comments | | | | |
| $occ \geq 1$ | 8.94 | 8.50 | 6.08 | 0.89 |
| $occ \geq 2$ | 23.00 | 16.21 | 7.25 | 1.96 |
| $occ \geq 3$ | 18.78 | 18.67 | 14.47 | 2.09 |
| $occ \geq 4$ | 25.29 | 19.69 | 16.76 | 4.55 |
| E. Ratio of number of LD to number of SD, case with comments | | | | |
| $occ \geq 1$ | 1.13 | 3.13 | 5.54 | 72.96 |
| $occ \geq 2$ | 0.31 | 1.04 | 1.70 | 26.13 |
| $occ \geq 3$ | 0.11 | 0.44 | 0.80 | 11.84 |
| $occ \geq 4$ | 0.04 | 0.13 | 0.31 | 5.22 |
| F. Ratio of number of LD to number of SD, case without comments | | | | |
| $occ \geq 1$ | 0.94 | 2.77 | 4.63 | 66.07 |
| $occ \geq 2$ | 0.27 | 0.83 | 1.46 | 21.10 |
| $occ \geq 3$ | 0.08 | 0.35 | 0.63 | 9.29 |
| $occ \geq 4$ | 0.03 | 0.10 | 0.24 | 3.33 |

detailed situation of the number of logical dependencies, under the conditions of a varying threshold for the number of files accepted in a commit, without any filtering according to the number of occurrences, for all test systems. If no threshold is set for the number of files in a commit then the number of logical dependencies outnumbers the structural dependencies by a factor of up to 72 (Figure 2, case a.).

***Question 2***. Considering changes in comments only 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 subtables A. and B., subtables C. and D., and subtables E. and F. from table II. We observe that the differences are not significant, the decrease represents 4%-10% of the value with comments, thus we can say that eliminating changes due only to comments can slightly improve the quality of finding logical dependencies.

***Question 3***. How many occurrences of a logical dependency are needed to consider it a *valid* logical dependency ?

Based on the results presented in subtables E. and F. of table II, we observe that we can obtain a number of logical dependencies which is between one quarter and one third of the number of the structural dependencies either by setting a commit size threshold of 5 files combined with an occurrence threshold of 2, or a commit size threshold of 10 files combined with an occurrence threshold of 3, or a commit size threshold of 20 files combined with an occurrence threshold of 4. It is clear that increasing the threshold value for number of occurrences can be done only together with increasing the threshold value for maximum number of committed files. A more complex filtering condition can aggregate the 3 simple filtering cases, in the form: $((cs \leq 5) and (occ \geq 2)) or ((cs \leq 10) and (occ \geq 3)) or ((cs \leq 20) and (occ \geq 4))$. The filtering cases are presented in Figure 2 cases b. and c. respectively d.

TABLE III
NUMBER OF LOGICAL DEPENDENCIES, FOR DIFFERENT THRESHOLD VALUES FOR $cs$, WHEN $occ \geq 1$, CASE WITH COMMENTS

| ID | SD | LD $cs \leq 5$ | LD $cs \leq 10$ | LD $cs \leq 20$ | LD $cs < \infty$ |
|---|---|---|---|---|---|
| 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 |
| Median for LD/SD | | 1.13 | 3.13 | 5.54 | 72.96 |

TABLE IV
NUMBER OF LOGICAL DEPENDENCIES, FOR DIFFERENT THRESHOLD VALUES FOR $cs$, WHEN $occ \geq 1$, CASE WITHOUT COMMENTS

| ID | SD | LD $cs \leq 5$ | LD $cs \leq 10$ | LD $cs \leq 20$ | LD $cs < \infty$ |
|---|---|---|---|---|---|
| 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 |
| Median for LD/SD | | 0.94 | 2.77 | 4.63 | 66.07 |

## VI. LESSONS LEARNED AND FUTURE WORK

An issue which has not been investigated enough is whether the reduced set of logical dependencies obtained after filtering

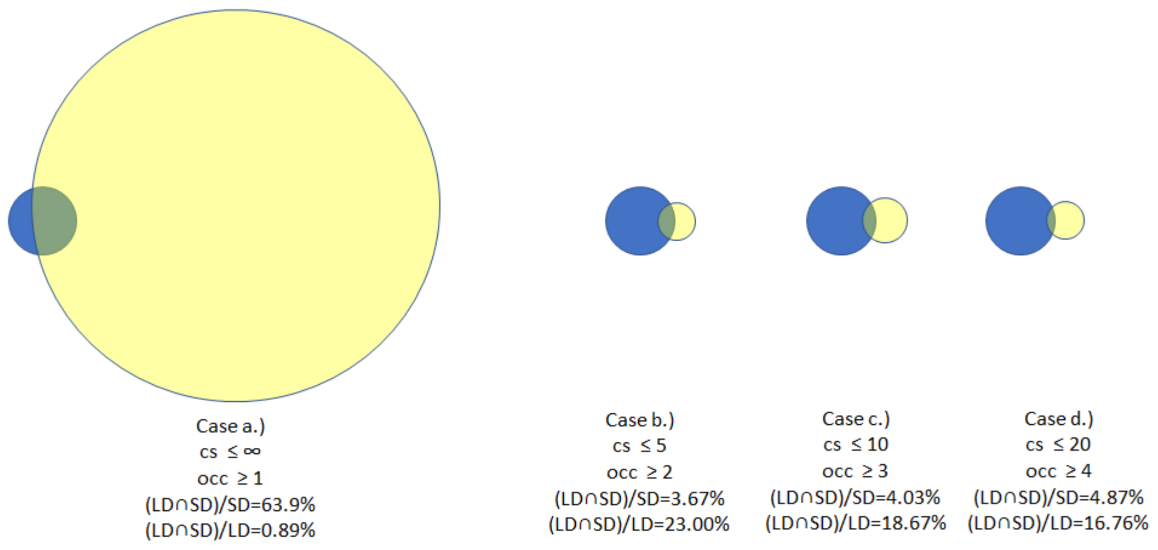

Fig. 2. Intersections of logical and structural dependencies, in different cases defined by different combinations of filtering thresholds.

contains indeed true logical dependencies. This could be done by manual inspection of the classes in order to validate the logical dependencies by the opinion of a human expert. Unfortunately doing such manual validation 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.

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 [19], [20] through clustering [21] or finding of key classes [22], and evaluating the positive impact on their results.

As we could see in table II C. and D., only a small amount of logical dependencies are between classes that also present structural dependencies. In our experiments, even after filtering, around 80% of the logical dependencies are between classes without structural dependencies. Although this big percentage is supported also by experiments of related works [2], [1], 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. 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 occurrences of co-changes: if co-changes 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.

## VII. Conclusion

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

Our experiments showed that the most important factors studied until now, 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. Increasing the threshold value for the number of occurrences can be done only together with increasing the threshold value for the maximum number of files allowed in a commit.

We have identified also new factors such as the history of structural dependencies and the trend of co-change occurrences which will be investigated in future work.The validation of our method for identifying logical dependencies will come by measuring the impact of using them to enhance dependency models used in applications of architectural reconstruction.

## References

[1] 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

[2] 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

[3] 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

[4] 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

[5] X. Ren, B. G. Ryder, M. Stoerzer, and F. Tip, “Chianti: a change impact analysis tool for java programs,” in *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005.*, May 2005, pp. 664–665.

[6] 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

[7] 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

[8] 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.

[9] 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.

[10] 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

[11] A. D. Stana, “An analysis of the relationship between structural and logical dependencies in software systems,” Master’s thesis, Politehnica University Timisoara, Romania, June 2018.

[12] A. D. Stana and I. Şora, “Identifying logical dependencies from co-changing classes,” in *Submitted to The 7th International Workshop on Software Mining (SoftwareMining) at ASE 2018*, 2018.

[13] 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

[14] 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

[15] 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

[16] 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

[17] 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.

[18] 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, Oct 2016. [Online]. Available: https://doi.org/10.1007/s10664-015-9393-5

[19] 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

[20] 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.

[21] I. Şora, G. Glodean, and M. Gligor, “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*, May 2010, pp. 259–264.

[22] I. Şora, “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*. Springer International Publishing, 2015, pp. 122–140.