1 Presentation of the research topic (max. 1 page)

The domain of the proposed thesis is Automated Software Engineering. The thesis will develop methods for the analysis of legacy software systems, focusing on using historical information describing the evolution of the systems extracted from the versioning systems. The methods for analysis will integrate techniques based on computational algorithms as well as data-mining. As proof-of-concept, tool prototypes will implement the proposed methods and validate them by extensive experimentation on several cases of real-life systems.

2 Current status of research within the proposed topic (max. 1 page)

The current trend recommends that general dependency management methods and tools should also include logical dependencies besides the structural dependencies [8], [1].

Structural dependencies are the result of the source code analysis of the system [2] while logical dependencies refer to those dependencies between entities that are not always visible through source code analysis and can be added during the development process. Logical dependencies can be easily extracted from the versioning system (e.g. Subversion , Git) commits.

The concepts of logical coupling and logical dependencies were first used in different analysis tasks, all related to changes: for software change impact analysis [11], for identifying the potential ripple effects caused by software changes during software maintenance and evolution [9], [8], [10], [7] or for their link to deffects [13], [14].

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 [5], [3], [4], 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 [6] shows

that historical information has been rarely used in architectural reconstruction.

Another survey [12] mentions one possible explanation why historical information have been rarely used in architectural reconstruction: the size of the extracted information. One problem is the size of the extraction process, which has to analyze many versions from the historical evolution of the system. Another problem is the big number of pairs of classes which record co-changes and how they relate to the number of pairs of classes with structural dependencies.

3 | Justification of research topic (max. 3 page)

The software architecture is important in order to understand and maintain the systems. The software architecture is created at the beginning together with the code and shall be updated once in a while to reflect the reality of the code. So reconstructing the architecture and verifying if still matches the reality is important [6].

As mentioned above, the main reason why historical information is rarely used in architectural reconstruction is the size of the extracted information.

Logical dependencies should integrate harmoniously with structural dependencies in an unitary dependency model: valid logical dependencies should not be omitted from the dependency model, but structural dependencies should not be engulfed by questionable logical dependencies generated by casual co-changes. 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 valid logical dependencies.

Other works have also tried to filter co-changes and study the overlapping between the filtered co-changes and structural dependencies[8], [1].

One of the most used co-changes filter is the commit size. Each change or group of changes in the source code are submitted in the versioning system as a commit. The commit size is the number of files changed in that particular commit.

Comparing the total amount of co-changes extracted without any filtering with the amount extracted with commit size filter we can say that the filter has indeed a big impact on the total number of extracted co-changes. This kind of filter can only shrink the number of co-changes extracted but cannot guarantee that the remaining ones have more relevancy and are more logical linked.

A logical dependency can be also a structural dependency and vice-versa so studying the overlapping between logical and structural dependencies while filtering is important since we want to introduce those logical dependencies among with structural dependencies in architectural reconstruction systems. If their overlapping is to high then extracting and filtering in order to introduce only a couple of new dependencies in the system will maybe not worth the effort. But current stud-

ies have shown a small percentage of overlapping between them with and without any kind of filtering [1].

Taking 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 concurrences 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.

Try not to give a fixed number as threshold for filters and try to deduce that number from the size of the system

4 | Research content and stages of research-implementation schedule

blblablu

5 | Necessary resources and available resources in UPT for the implementation of the research training

The topic will be developed within the Database and Artificial Intelligence lab from UPT. The topic continues the research directions from grant "Automated recovery of architectural information from source code - AReAS".

Bibliography

[1] Nemitari Ajienka and Andrea Capiluppi. Understanding the interplay between the logical and structural coupling of software classes. *Journal of Systems and Software*, 134:120–137, 2017.

[2] David Binkley. Source code analysis: A road map. pages 104–119, 06 2007.

[3] Ioana Şora. 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, 2013.

[4] Ioana 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*, pages 122–140. Springer International Publishing, 2015.

[5] Ioana Şora, Gabriel Glodean, and Mihai 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*, pages 259–264, May 2010.

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

[7] 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*, pages 119–128, Oct 2010.

[8] Gustavo Ansaldi Oliva and Marco Aurelio Gerosa. 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, 2011. IEEE Computer Society.

[9] Gustavo Ansaldi Oliva and Marco Aurélio 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, pages 250–260, 2015.

[10] Denys Poshyvanyk, Andrian Marcus, Rudolf Ferenc, and Tibor Gyimóthy. Using information retrieval based coupling measures for impact analysis. *Empirical Software Engineering*, 14(1):5–32, Feb 2009.

[11] Xiaoxia 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.*, pages 664–665, May 2005.

[12] Mark Shtern and Vassilios Tzerpos. Clustering methodologies for software engineering. *Adv. Soft. Eng.*, 2012:1:1–1:1, January 2012.

[13] Igor Scaliante Wiese, Rodrigo Takashi Kuroda, Reginaldo Re, Gustavo Ansaldi Oliva, and Marco Aurélio Gerosa. An empirical study of the relation between strong change coupling and defects using history and social metrics in the apache aries project. In Ernesto Damiani, Fulvio Frati, Dirk Riehle, and Anthony I. Wasserman, editors, *Open Source Systems: Adoption and Impact*, pages 3–12, Cham, 2015. Springer International Publishing.

[14] Thomas Zimmermann, Peter Weisgerber, Stephan Diehl, and Andreas Zeller. 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, 2004. IEEE Computer Society.