Universitatea Politehnica Timişoara

Lucrare de finalizare a activităţii la disciplina

Etică şi Integritate Academică în Cercetare Ştiinţifică şi Diseminarea Rezultatelor

Anul I – Doctorat

An universitar 2018/2019

Doctorand: Adelina Diana Stana

Conducător ştiinţific: prof.dr.ing Vladimir-Ioan Cretu

Departamentul: Calculatoare si Tehnologia Informatiei

Cuprins

IOSUD - University Politehnica Timişoara
Doctoral School of Engineering Studies

Annex 3.1 to the Contract of Doctoral Studies

Individual Curriculum[i]

The Ph.D. student: Stana Adelina Diana
Scientific supervisor: 1. prof.dr.ing Cretu Vladimir-Ioan
Advisory Committee: 2. conf.dr.ing Marinescu Cristina
3. conf.dr.ing Chirila Ciprian
4. conf.dr.ing Pescaru Dan

PhD Field: Automated Software Engineering
Proposed topic of the doctoral thesis: Methods and Tools for the Analysis of Legacy Software Systems

Content of the individual curriculum [ii]:
1. Presentation of the research topic (max. 1 page)
2. Current status of research within the proposed topic [iii] (max. 1 page)
3.Justification of research topic [iv] (max. 3 page)
4. Research content and stages of research-implementation schedule[v]
5. Necessary resources and available resources in UPT for the implementation of the research training [vi]
6. Bibliography[vii]

RESEARCH REPORT AND SCIENTIFIC PAPERS
SCHEDULED AS RESULTS OF THE RESEARCH TRAINING

Activities	Estimated deadlines of completion
1. Drawing up of the scientific report nr. 1 with the intermediate results of the research	3/1/2019
2. Drawing up of the scientific report nr. 2 with the intermediate result of the research	2/1/2020
3. Publication of a number of $n_1 = 2$ scientific papers in light of attending conferences rated IDB	5/1/2021
4. Publication of a number of $n_2 = 3$ scientific papers in the ISI circuit ($n_2 \geq 2$)	5/1/2021

Signature of the Ph.D. student: ________________________ Date ________________________

The scientific supervisor's remarks: (approval, feasibility, other assessments and remarks) [viii]
____
____
____
____
____
____
____
____

Signatures: 1. ______________________ 2. ____________________________________
3. ______________________ 4. ____________________________________
Date_________

[i] The individual curriculum will be conceived by the Ph.D. student under the guidance of his/her scientific supervisor. It will be presented by the Ph.D. student by the end of the 1st doctoral year. It will have, as a first page, this form.
[ii] The individual curriculum will be designed in conformity with the content below.
[iii] The presentation of the current status of the research will be mandatory authenticated by using the titles mentioned in the bibliography from point 6.
[iv] The justification will contain at least: the thesis' objectives, their anchorage within the stage presented in point 2, and the means by which the objectives are to be achieved.
[v] The contend of the research will be focused on the objectives presented in point 3. The implementation schedule must cover the entire period of time onto which the doctoral research extends (including the 1st year).
[vi] The estimation of resources must be realistic and will be discussed, in depth, with the scientific supervisor and the advisory committee.
[vii] Any title from the bibliography must be referred to, mandatory, in points 1 – 5.
[viii] The remarks will be filled in after presenting the individual curriculum. They must underline, first of all the feasibility of the individual curriculum undertaken by the Ph.D. student under the guidance of the scientific supervisor and the advisory committee.

1 Presentation of the research topic

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

The current trend recommends that general dependency management methods and tools should also include logical dependencies besides the structural dependencies [10], [1].

A dependency is created by two elements that are in a relationship and indicates that an element of the relationship, in some manner, depends on the other element of the relationship [2], [3]. In the case of object oriented software systems, dependency models are usually class dependency models where elements are entities such as classes and interfaces [14]. There are several types of relationships between these source code entities, for example a method of a class can call a method of another class, a class extends another class, all those create *structural dependencies* between classes (a.k.a syntactic dependencies or structural coupling). These dependencies can be found by the analysis of the source code.

Software engineering practice has shown that sometimes modules which do not present structural dependencies still appear to be related. Co-evolution represents the phenomenon when one component changes in response to a change in another component [18]. Those changes can be found in the software history maintained by the versioning system. Gall [8] identified as logical coupling between two modules the fact that these modules *repeatedly* change together during the historical evolution of the software system. *Logical dependencies* (a.k.a logical coupling) can be found by software history analysis and can reveal relationships that are not always present in the source code (structural dependencies).

The concepts of logical coupling and logical dependencies were first used in different analysis tasks, all related to changes: for software change impact analysis [13], for identifying the potential ripple effects caused by software changes during software maintenance and evolution [11], [10], [12], [9] or for their link to deffects [17], [19].

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 [6], [4], [5], 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 [7] shows that historical information has been rarely used in architectural reconstruction.

Another survey [15] 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

The software architecture is important in order to understand and maintain a system. Often code updates are made without checking or updating the architecture. This kind of updates cause the architecture to drift from the reality of the code over time.[7] So reconstructing the architecture and verifying if still matches the reality is important.

Surveys show that architectural reconstruction is mainly made based on structural dependencies [15] [7], 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.

Currently there is no set of rules or best practices that can be applied to the extracted class co-changes and can guarantee their filtering into a set of valid logical dependencies. This is mainly because not all the updates made in the versioning system are code related. For example 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 co-changing pairs of entities can be introduced. So, in order to exclude this kind of situations the information extracted from the versioning system has to be filtered first and then used.

Other works have tried to filter co-changes [10], [1]. One of the used co-changes filter is the commit size.The commit size is the number of code files changed in that particular commit. Ajienka and Capiluppi established a threshold of 10 for the maximum accepted size for a commit [1]. This means that all the commits that had more than 10 code files changed where discarded from the research. But setting a harcoded threshold for the commit size is debatable because in order to say that a commit is big or small you have to look first at the size of the system and at the trends from the versioning system. Even thought the best practices encourage small and often commits, the developers culture is the one that influences the most the trending size of commits from one system.

Filtering only after commit size is not enought, this type of filtering can indeed have an impact on the total number of extracted co-changes, but will only shrink the number of co-changes extracted without actually guaranteeing that the remaining ones have more relevancy and are more logical linked.

Although, some unrelated files can be updated by human error in small commits, for example: one file was forgot to be commited in the current commit and will be commited in the next one among some unrelated files. This kind of situation can introduce a set of co-changing pairs that are definetly not logical liked. In order to avoid this kind of situation a filter for the occurrence rate of co-changing pairs must be introduced. Co-changing pairs that occur multiple times are more prone to be logically dependent than the ones that occur only once. Currently there are no concrete examples of how the threshold for this type of filter can be calculated. In order to do that, incrementing the threshold by a certain step will be the start and then studying the impact on the remaining co-changing pairs for different systems.

Taking into account also structural dependencies from all the revisions of the system was not made in previous works, this step is important 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.

Also, 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 the intention is to introduce those logical dependencies among with structural dependencies in architectural reconstruction systems. Current studies have shown a relatively small percentage of overlapping between them with and without any kind of filtering [1]. This means that a lot of non related entities update together in the versioning system, the goal here is to establish the factors that determine such a small percentage of overlapping.

4 Research content and stages of research-implementation schedule

The research will be made by following the next stages of implementation:

A. DEVELOPMENT OF CONTENT AND TOOLS

Stage 1: Build tool to extract structural dependencies from code and co-changes from git for a given set of projects.

Stage 2: Find filters for the co-changes extracted, the filters can be the ones already mentioned in previous works or new ones. Establish different thresholds for those filters.

Stage 3: Study the impact of those filters and the coresponding thresholds on the remaining quantity of co-changes for each system. Study the overlappings between the remaining pairs of co-changing entites and the structural dependencies extracted. [16]

Stage 4: Establish a dynamic way to determine the thresholds for filters in order to fit the best each studied sistem. Main focus on the threshold for number of occurences of co-changing pairs. Use plots or other visual instruments in order to see the highests and the lowest rates for the numbers of occurrences among co-changing pairs. Also filter those rates into normal and abnormal ones and study what was the cause of the highest rates (code or human related).

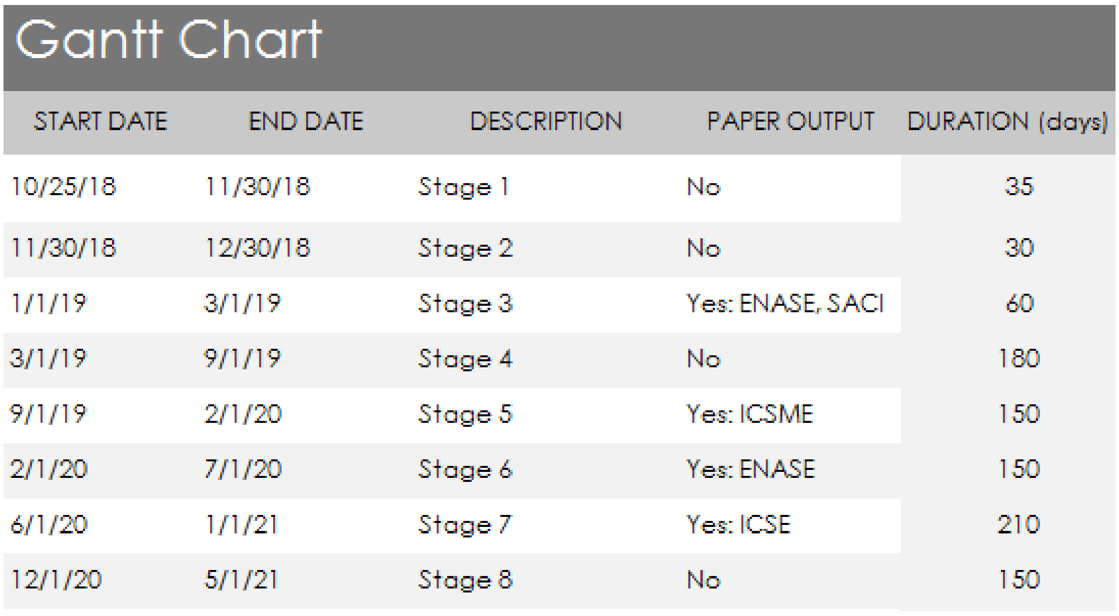
Stage 5: Take into account also structural dependencies from all the revisions of the system to filter out the old, out-of-date logical dependencies. Study how this affects the remaining number of logical dependencies.Here an extra check is needed, it can be a case in which old structural dependencies that were also logically linked to continue to be logically linked even after the structural dependency was removed.

B. USAGE

Stage 6: Export the remaining co-changes whom at this step we can call logical dependencies and use them among structural dependencies in tools for architectural reconstruction to evaluate the improvement.

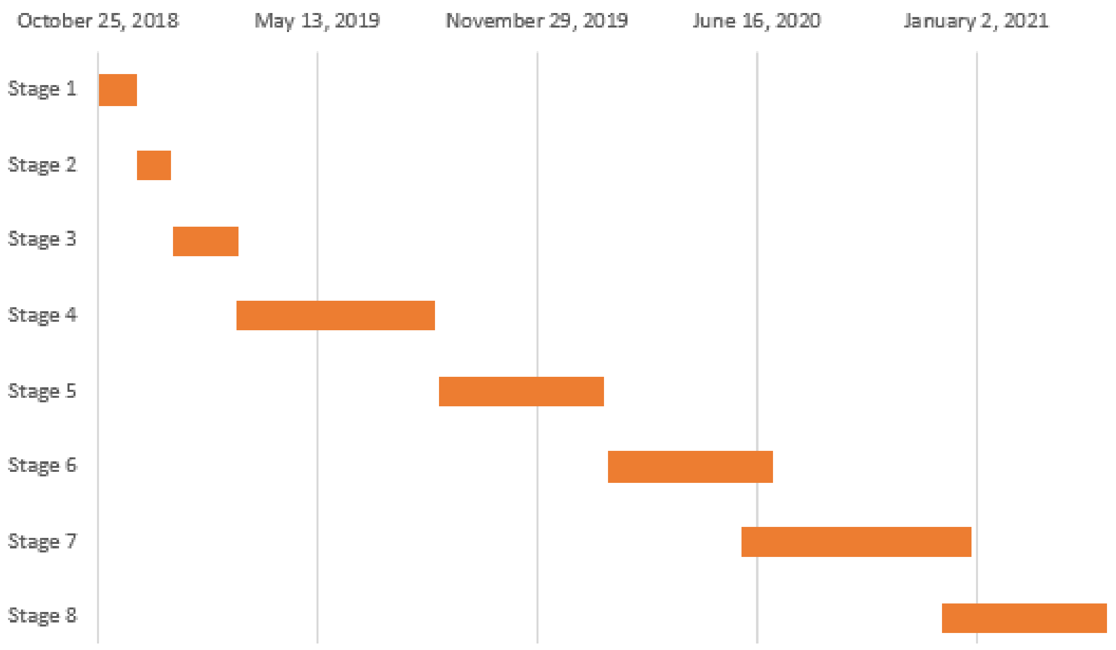
Stage 7: Compare the number of logical dependencies with metrics like Fan Out, Fan In, Efferent Coupling (Ce), Afferent Coupling (Ca) and study their connections.

Stage 8: Identify other tools that use historical information and evaluate the impact of co-changes filtering into logical dependences for them.



Gantt Chart

START DATE	END DATE	DESCRIPTION	PAPER OUTPUT	DURATION (days)
10/25/18	11/30/18	Stage 1	No	35
11/30/18	12/30/18	Stage 2	No	30
1/1/19	3/1/19	Stage 3	Yes: ENASE, SACI	60
3/1/19	9/1/19	Stage 4	No	180
9/1/19	2/1/20	Stage 5	Yes: ICSME	150
2/1/20	7/1/20	Stage 6	Yes: ENASE	150
6/1/20	1/1/21	Stage 7	Yes: ICSE	210
12/1/20	5/1/21	Stage 8	No	150



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] Grady Booch. *Object-Oriented Analysis and Design with Applications (3rd Edition)*. Addison Wesley Longman Publishing Co., Inc., Redwood City, CA, USA, 2004.

[3] Marcelo Cataldo, Audris Mockus, Jeffrey A. Roberts, and James D. Herbsleb. Software dependencies, work dependencies, and their impact on failures. *IEEE Transactions on Software Engineering*, 35:864–878, 2009.

[4] 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.

[5] 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.

[6] 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.

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

[8] Harald Gall, Karin Hajek, and Mehdi Jazayeri. 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, 1998. IEEE Computer Society.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] Neeraj Sangal, Ev Jordan, Vineet Sinha, and Daniel Jackson. Using dependency models to manage complex software architecture. In *Proceedings of the 20th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*, OOPSLA '05, pages 167–176, New York, NY, USA, 2005. ACM.

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

[16] Adelina Diana Stana. and Ioana ora. Identifying logical dependencies from co-changing classes. In *Proceedings of the 14th International Conference on Evaluation of Novel Approaches to Software Engineering - Volume 1: ENASE,*, pages 486–493. INSTICC, SciTePress, 2019.

[17] 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.

[18] Liguo Yu. Understanding component co-evolution with a study on linux. *Empirical Softw. Engg.*, 12(2):123–141, April 2007.

[19] 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.

Identifying logical dependencies from co-changing classes

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

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 valid logical dependencies. In order to find the characteristics of logical dependencies, 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 27 open source software projects written in Java and C# with their historical evolutions which sum up to over 70000 commit transactions. Identifying valid 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, which both can be determined by source code analysis.

The state of the art has found 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. 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. 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.

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 valid logical dependencies are identified and we 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. We have analyzed several open-source software systems of different sizes with our tool, obtaining the experimental results presented in Section 4. Section 5 discusses the experimental results and answers the research questions.

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. We consider that such a big number of logical dependencies needs additional filtering.

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 valid 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 size of commit transactions which are accepted to generate 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 threshold values for the maximum size of commit transactions which are accepted to generate logical dependencies.

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) it 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 Experimental results

We have analyzed 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) ex-

[1]http://github.com/

tracted; 4th column shows the number of most recent commits analyzed from the active branch of each project and the 5th column shows the language in which the project was developed.

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	C#
16	aspnetboilerplate	2353	1615	C#
17	orleans	3485	3353	C#
18	cli	767	2397	C#
19	cake	2250	1853	C#
20	Avalonia	1677	2445	C#
21	EntityFramework	7107	2443	C#
22	jellyfin	2179	4065	C#
23	PowerShell	861	2033	C#
24	WeiXinMPSDK	2029	2723	C#
25	ArchiSteamFarm	117	2181	C#
26	VisualStudio	1016	4417	C#
27	CppSharp	259	3882	C#

In a first experiment, we determined the commit sizes *cs* for all commit transactions for all projects and grouped them into 4 categories: small transactions (ST), when $cs \leq 5$; medium transactions (MT), when $5 < cs \leq 10$; large transactions (LT), when $10 < cs \leq 20$; and very large transactions (VLT), when $20 < cs$. Also, we counted how many logical dependencies are generated by transactions from each category. The results are presented in Tables 2 and 3 as percent distributions.

In the main series of experiments, 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

In all cases, we varied the following threshold values:

- commit size (*cs*): the maximum size of commit transactions which are accepted to generate logical dependencies. 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 val-

Table 2: The percent distribution of commit transactions in 4 categories according to their size

	ST	MT	LT	VLT
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 3: The percent distribution of logical dependencies generated by commit transactions from each size category

	ST	MT	LT	VLT
1	9,70	2,61	4,12	83,57
2	1,50	1,87	3,59	93,03
3	3,75	5,02	5,97	85,26
4	31,40	7,64	8,84	52,12
5	1,01	3,92	4,67	90,41
6	0,37	0,22	0,47	98,94
7	1,48	1,86	2,48	94,18
8	1,44	2,01	3,73	92,82
9	6,77	7,88	11,99	73,36
10	12,53	17,77	21,59	48,10
11	6,80	8,25	13,74	71,22
12	22,09	21,73	20,51	35,67
13	10,46	20,48	8,08	60,98
14	1,90	0,90	1,29	95,91
15	1,14	1,25	1,86	95,76
16	1,89	2,47	3,12	92,52
17	2,13	2,19	5,25	90,44
18	1,77	3,66	6,51	88,06
19	0,59	0,68	1,57	97,17
20	0,41	0,73	1,42	97,45
21	1,50	1,22	37,85	59,43
22	2,00	4,12	5,95	87,92
23	1,02	1,22	0,94	96,82
24	0,71	0,74	1,63	96,91
25	37,86	16,51	11,12	34,50
26	2,86	3,22	6,79	87,13
27	23,43	21,56	28,28	26,73
Avg	6,98	5,99	8,27	78,76

ues:

- 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, 4, 5, 6, 7, 8, 9 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 filtering conditions, while the lower left corner corresponds to the most restrictive filtering conditions.

Table 4: Ratio of number of LD to number of SD, case with comments

	$cs \leq 5$	$cs \leq 10$	$cs \leq 20$	$cs < \infty$
$occ \geq 1$	3,39	5,67	9,00	80,31
$occ \geq 2$	2,24	3,47	5,02	60,14
$occ \geq 3$	1,04	2,53	3,52	44,68
$occ \geq 4$	0,90	2,16	2,88	33,47

Table 5: Ratio of number of LD to number of SD, case without comments

	$cs \leq 5$	$cs \leq 10$	$cs \leq 20$	$cs < \infty$
$occ \geq 1$	3,24	5,33	7,90	67,16
$occ \geq 2$	1,35	3,27	4,72	47,39
$occ \geq 3$	1,00	1,67	2,49	32,39
$occ \geq 4$	0,43	1,26	1,93	22,15

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

	$cs \leq 5$	$cs \leq 10$	$cs \leq 20$	$cs < \infty$
$occ \geq 1$	19,75	29,86	39,29	76,59
$occ \geq 2$	12,50	20,20	27,68	66,11
$occ \geq 3$	8,49	14,22	19,94	55,99
$occ \geq 4$	6,58	10,95	15,76	47,12

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

	$cs \leq 5$	$cs \leq 10$	$cs \leq 20$	$cs < \infty$
$occ \geq 1$	18,88	28,47	37,44	71,12
$occ \geq 2$	11,87	19,03	25,93	59,58
$occ \geq 3$	8,00	13,09	18,15	48,65
$occ \geq 4$	5,85	9,94	14,27	39,07

Table 8: Percentage of LD that are also SD, case with comments

	$cs \leq 5$	$cs \leq 10$	$cs \leq 20$	$cs < \infty$
$occ \geq 1$	12,02	8,86	6,72	1,79
$occ \geq 2$	15,05	11,71	9,38	2,21
$occ \geq 3$	17,45	13,97	11,57	2,86
$occ \geq 4$	18,96	15,28	12,94	3,67

Table 9: Percentage of LD that are also SD, case without comments

	$cs \leq 5$	$cs \leq 10$	$cs \leq 20$	$cs < \infty$
$occ \geq 1$	12,05	9,02	6,98	1,93
$occ \geq 2$	15,08	12,03	9,66	2,42
$occ \geq 3$	17,78	14,37	12,24	3,28
$occ \geq 4$	19,22	15,59	13,30	4,21

5 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 2 presents the size distribution for commit transactions in percentage relative to the total number of commits for each system presented in Table 1. The small commit transactions (with less than 5 source code files)represent in average 78.76% from the total number of transactions. On the opposite side are the very large commit transactions (with more than 20 source code files) which represent an average percentage of 5.99% from the total number of transactions. Based on these results we can say that the vast 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 ? Logical dependencies are generated for all pairs of classes which have changed in the same commit transaction. The number of logical dependencies generated from a commit transaction is proportional with the square of the number of participating classes. Table 3 presents how many logical dependencies are extracted from commit transactions of different sizes. Based on the results from Table 3 and Table 2 we see that the commit transactions with less than 5 files, which are the most frequent types of commits, produce in average only 6.98% of the total logical dependencies extracted from the systems. On the other hand, a small amount of very large commits (those with more than 20 source code files) can lead to a vast amount of logical dependencies. But very large commit transactions can be caused by merging development branches into the main branch. In this case the very large commit transaction is actually the sum of many other commit transactions made into a different branch and we cannot consider them as one single commit and definitely we cannot consider the logical dependencies extracted as valid logical dependencies. So a threshold to filter this kind of commit transactions is required.

Based on the results presented in Tables 4 and 5, 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 4 and 5) then the number of logical dependencies outnumbers the structural dependencies with a factor of up to 80. 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 72 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 4 and 5 that the values on the last column still do not fall below 20. This number is still too big to accept for logical dependencies. It is clear that it is necessary to 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 6 and 7 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. Setting a threshold leads to a smaller number of logical dependencies overall and 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 4 that the total number of logical dependencies decreases approx 20 times compared with no threshold. In the same time, we can see in Table 6 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 4 and 5, Tables 6 and 7 and Tables 8 and 9. 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 4 and 5 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 67.1, compared to the value with comments which is 80.3. The decrease represents 13% of the value with comments. In the case of the percentage of structural dependencies that are also logical dependencies, from Tables 6 and 7, we can see that the maximum difference is also for the values from the first row, last column. Without comments, the overlap is 76.5, compared to the value with comments which is 71.1. The decrease represents less than 6% 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 4 or in Table 5, 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 not be 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 ?

The overlap between structural and logical dependencies is given by the number of pairs of classes that have both structural and logical dependencies. We evaluate this overlap as a percentage relative to the number of structural dependencies in Tables 6 and 7, respectively as a percentage relative to the number of logical dependencies in Tables 8 and 9.

A first observation from Tables 6 and 7 is that not all pairs of classes with structural dependencies cochange. The biggest value for the percentage of structural dependencies that are also logical dependencies is 76.5% obtained in the case when no filterings are done.

From Tables 8 and 9 we notice that the percentage of logical dependencies which are also structural is always low to very low. This means that most cochanges are recorded between classes that have no structural dependencies to each other.

6 Future work

We consider that in the future, the validation of extracted logical dependencies will occur by using them to enhance dependency models used by different applications such as architectural reconstruction (Şora et al., 2010), (Şora, 2015), and evaluating the positive impact on their results.

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 logical dependencies. Some logical dependencies may have been also structural in previous revisions of the system but not in the current one. Another way to investigate this problem could be to study the trend of occurrencies 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.

7 Conclusion

In this work we experimentally define methods to filter out the valid logical dependencies from cochanging classes.

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

We conclude that it is important to put a threshold on the maximum size of commit transactions which are accepted to generate logical dependencies. Only small commit transactions (changing up to 5 source code files) can be reliably used for introducing logical dependencies. We have also determined that small commit transactions are the most frequent kind of transactions, representing in average 80% of all commit transactions. Under these conditions, we have determined that increasing the threshold for the minimum number of repeated occurrences for a co-change to be counted as a logical dependency reduces significantly the number of logical dependencies. In average, increasing with 1 the threshold for repeated occurrences determines a reduction to half for the number of logical dependencies. A value of 4 for the threshold for repeated occurrences, combined with the condition of accepting only small commit transactions, already keeps the number of logical dependencies in the same range as the number of structural dependencies. Future work will investigate further the issue of repeated occurrences, analyzing also their trend in time.

The analysis of the experimental data shows that logical dependencies are distinct from structural dependencies. Even after filtering, a very big percentage of logical dependencies are between classes without structural dependencies. This leads to the conclusion that including into dependency models also logical dependencies besides structural dependencies has the potential to improve analysis applications based on dependency models.

8 Acknowledgements

This work was partially supported by a grant of the Romanian National Authority for Scientic Research and Innovation, CNCS/CCCDI UEFISCDI, project number PN-III-P2-2.1-PED-2016-0999, within PNCDI III.

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.

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.

Authors' Instructions: Preparation of Camera-Ready Contributions to SCITEPRESS Proceedings

First Author Name[1][a], Second Author Name[1][b] and Third Author Name[2][c]
[1]*Institute of Problem Solving, XYZ University, My Street, MyTown, MyCountry*
[2]*Department of Computing, Main University, MySecondTown, MyCountry*
{f_author, s_author}@ips.xyz.edu, t_author@dc.mu.edu

Keywords: The paper must have at least one keyword. The text must be set to 9-point font size and without the use of bold or italic font style. For more than one keyword, please use a comma as a separator. Keywords must be titlecased.

Abstract: The abstract should summarize the contents of the paper and should contain at least 70 and at most 200 words. The text must be set to 9-point font size.

1 INTRODUCTION

Your paper will be part of the conference proceedings therefore we ask that authors follow the guidelines explained in this example in order to achieve the highest quality possible (Smith, 1998).

Be advised that papers in a technically unsuitable form will be returned for retyping. After returned the manuscript must be appropriately modified.

2 MANUSCRIPT PREPARATION

We strongly encourage authors to use this document for the preparation of the camera-ready. Please follow the instructions closely in order to make the volume look as uniform as possible (Moore and Lopes, 1999).

Please remember that all the papers must be in English and without orthographic errors.

Do not add any text to the headers (do not set running heads) and footers, not even page numbers, because text will be added electronically.

For a best viewing experience the used font must be Times New Roman, except on special occasions, such as program code 2.4.8.

[a] https://orcid.org/0000-1111-2222-3333
[b] https://orcid.org/1111-2222-3333-4444
[c] https://orcid.org/2222-3333-4444-5555

2.1 Manuscript Setup

The template is composed by a set of 7 files, in the following 2 groups:
Group 1. To format your paper you will need to copy into your working directory, but NOT edit, the following 4 files:

- `apalike.bst`
- `apalike.sty`
- `article.cls`
- `scitepress.sty`

Group 2. Additionally, you may wish to copy and edit the following 3 example files:

- `example.bib`
- `example.tex`
- `scitepress.eps`

2.2 Page Setup

The paper size must be set to A4 (210x297 mm). The document margins must be the following:

- Top: 3,3 cm;
- Bottom: 4,2 cm;
- Left: 2,6 cm;
- Right: 2,6 cm.

It is advisable to keep all the given values because any text or material outside the aforementioned margins will not be printed.

2.3 First Section

This section must be in one column.

2.3.1 Title and Subtitle

Use the command *title* and follow the given structure in ”example.tex”. The title and subtitle must be with initial letters capitalized (titlecased). The separation between the title and subtitle is done by adding a colon ”:” just before the subtitle beginning. In the title or subtitle, words like ”is”, ”or”, ”then”, etc. should not be capitalized unless they are the first word of the title or subtitle. No formulas or special characters of any form or language are allowed in the title or subtitle.

2.3.2 Authors and Affiliations

Use the command *author* and follow the given structure in ”example.tex”.

2.3.3 Keywords

Use the command *keywords* and follow the given structure in ”example.tex”. Each paper must have at least one keyword. If more than one is specified, please use a comma as a separator. The sentence must end with a period.

2.3.4 Abstract

Use the command *abstract* and follow the given structure in ”example.tex”. Each paper must have an abstract up to 200 words. The sentence must end with a period.

2.4 Second Section

Files ”example.tex” and ”example.bib” show how to create a paper with a corresponding list of references.

This section must be in two columns.

Each column must be 7,5-centimeter wide with a column spacing of 0,8-centimeter.

The section text must be set to 10-point.

Section, subsection and sub-subsection first paragraph should not have the first line indent.

To remove the paragraph indentation (only necessary for the sections), use the command *noindent* before the paragraph first word.

If you use other style files (.sty) you MUST include them in the final manuscript zip file.

2.4.1 Section Titles

The heading of a section title should be in all-capitals.

Example: *section*{*FIRST TITLE*}

2.4.2 Subsection Titles

The heading of a subsection title must be with initial letters capitalized (titlecased).

Words like ”is”, ”or”, ”then”, etc. should not be capitalized unless they are the first word of the subsection title.

Example: *subsection*{*First Subtitle*}

2.4.3 Sub-Subsection Titles

The heading of a sub subsection title should be with initial letters capitalized (titlecased).

Words like ”is”, ”or”, ”then”, etc should not be capitalized unless they are the first word of the sub subsection title.

Example: *subsubsection*{*First Subsubtitle*}

2.4.4 Tables

Tables must appear inside the designated margins or they may span the two columns.

Tables in two columns must be positioned at the top or bottom of the page within the given margins. To span a table in two columns please add an asterisk (*) to the table *begin* and *end* command.

Example: *begin*{*table**}
end{*table**}

Tables should be centered and should always have a caption positioned above it. The font size to use is 9-point. No bold or italic font style should be used.

The final sentence of a caption should end with a period.

Table 1: This caption has one line so it is centered.

Example column 1	Example column 2
Example text 1	Example text 2

Table 2: This caption has more than one line so it has to be justified.

Example column 1	Example column 2
Example text 1	Example text 2

Please note that the word ”Table” is spelled out.

2.4.5 Figures

Please produce your figures electronically, and integrate them into your document and zip file.

Check that in line drawings, lines are not interrupted and have a constant width. Grids and details within the figures must be clearly readable and may not be written one on top of the other.

Figure resolution should be at least 300 dpi.

Figures must appear inside the designated margins or they may span the two columns.

Figures in two columns must be positioned at the top or bottom of the page within the given margins. To span a figure in two columns please add an asterisk (*) to the figure *begin* and *end* command.

Example: \begin{*figure**}
\end{*figure**}

Figures should be centered and should always have a caption positioned under it. The font size to use is 9-point. No bold or italic font style should be used.

Figure 1: This caption has one line so it is centered.

Figure 2: This caption has more than one line so it has to be justified.

The final sentence of a caption should end with a period.

Please note that the word ”Figure” is spelled out.

2.4.6 Equations

Equations should be placed on a separate line, numbered and centered.
The numbers accorded to equations should appear in consecutive order inside each section or within the contribution, with the number enclosed in brackets and justified to the right, starting with the number 1.

Example:

$$a = b + c \tag{1}$$

2.4.7 Algorithms and Listings

Algorithms and Listings captions should have font size 9-point, no bold or italic font style should be used and the final sentence of a caption should end with a period. Captions with one line should be centered and if it has more than one line it should be set to justified.

2.4.8 Program Code

Program listing or program commands in text should be set in typewriter form such as Courier New.

Example of a Computer Program in Pascal:

```
Begin
    Writeln('Hello World!!');
End.
```

The text must be aligned to the left and in 9-point type.

2.4.9 Reference Text and Citations

References and citations should follow the Harvard (Author, date) System Convention (see the References section in the compiled manuscript). As example you may consider the citation (Smith, 1998). Besides that, all references should be cited in the text. No numbers with or without brackets should be used to list the references.

References should be set to 9-point. Citations should be 10-point font size.

You may check the structure of ”example.bib” before constructing the references.

For more instructions about the references and citations usage please see the appropriate link at the conference website.

3 COPYRIGHT FORM

For the mutual benefit and protection of Authors and Publishers, it is necessary that Authors provide formal written Consent to Publish and Transfer of Copyright before publication of the Book. The signed Consent ensures that the publisher has the Author’s authorization to publish the Contribution.

The copyright form is located on the authors’ reserved area.

The form should be completed and signed by one author on behalf of all the other authors.

4 CONCLUSIONS

Please note that ONLY the files required to compile your paper should be submitted. Previous versions or examples MUST be removed from the compilation directory before submission.

We hope you find the information in this template useful in the preparation of your submission.

ACKNOWLEDGEMENTS

If any, should be placed before the references section without numbering. To do so please use the following command: \*section**{*ACKNOWLEDGEMENTS*}

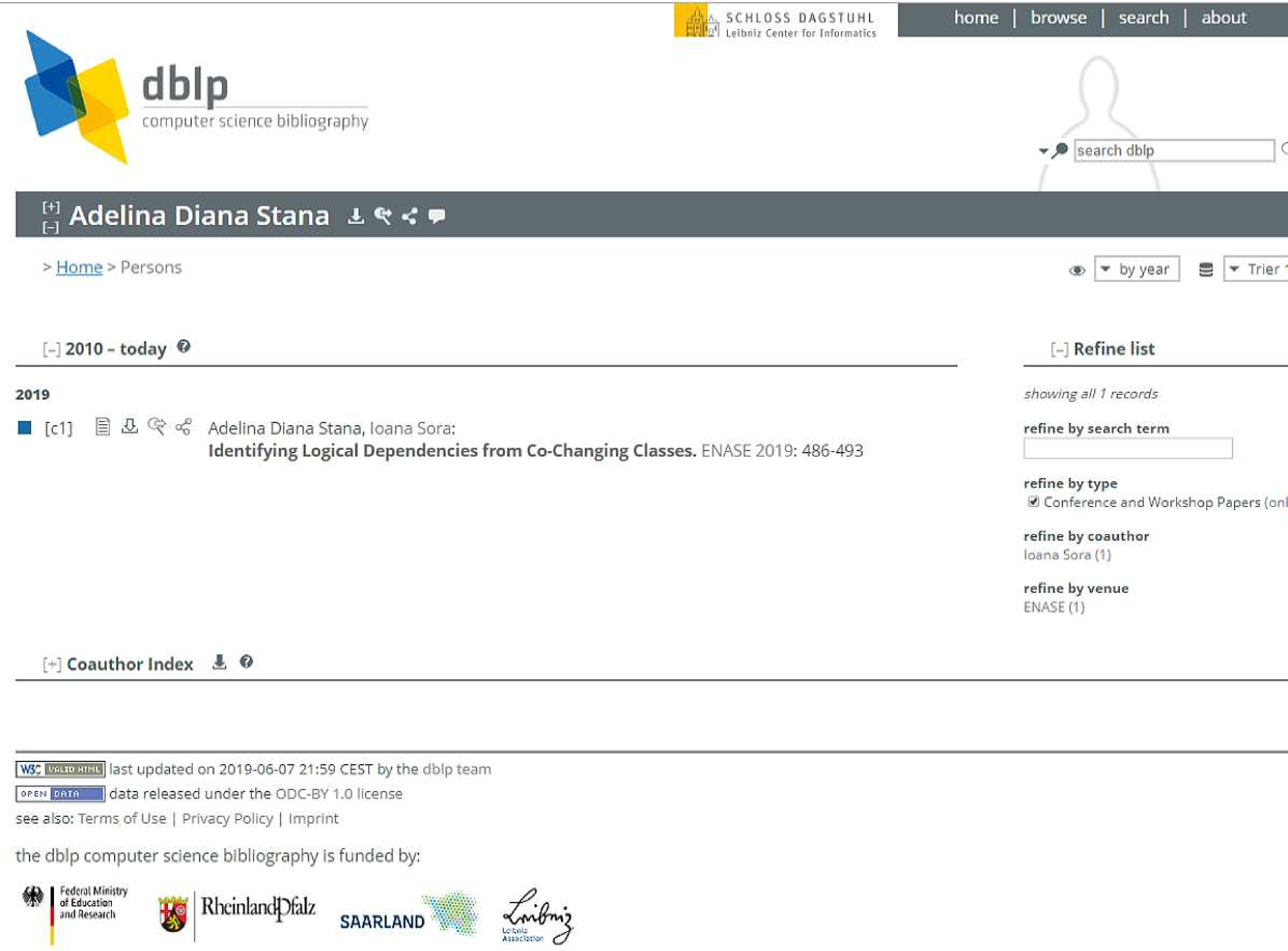
REFERENCES

Moore, R. and Lopes, J. (1999). Paper templates. In *TEMPLATE'06, 1st International Conference on Template Production*. SCITEPRESS.

Smith, J. (1998). *The Book*. The publishing company, London, 2nd edition.

APPENDIX

If any, the appendix should appear directly after the references without numbering, and not on a new page. To do so please use the following command: \*section**{*APPENDIX*}


SCHLOSS DAGSTUHL
Leibniz Center for Informatics
home | browse | search | about
dblp
computer science bibliography
search dblp

Adelina Diana Stana

> Home > Persons

[–] 2010 – today

2019

■ [c1] Adelina Diana Stana, Ioana Sora:
Identifying Logical Dependencies from Co-Changing Classes. ENASE 2019: 486-493

[–] Refine list

showing all 1 records

refine by search term

refine by type
☑ Conference and Workshop Papers (on

refine by coauthor
Ioana Sora (1)

refine by venue
ENASE (1)

[+] Coauthor Index

last updated on 2019-06-07 21:59 CEST by the dblp team
data released under the ODC-BY 1.0 license
see also: Terms of Use | Privacy Policy | Imprint

the dblp computer science bibliography is funded by:

Web of Science InCites Journal Citation Reports Essential Science Indicators EndNote Publons Kopernio Sign In Help English

Web of Science

Clarivate Analytics

Search Tools Searches and alerts Search History Marked List

Results: 63
(from Web of Science Core Collection)

You searched for: ALL FIELDS: (PROCEEDINGS OF THE 13TH INTERNATIONAL CONFERENCE ON EVALUATION OF NOVEL APPROACHES TO SOFTWARE ENGINEERING) **...More**

Create Alert

Refine Results

Search within results for...

Filter results by:

Open Access (3)

Refine

Publication Years

- 2018 (54)
- 2017 (2)
- 2016 (1)
- 2014 (3)
- 2011 (1)

more options / values...

Refine

Web of Science Categories

- COMPUTER SCIENCE SOFTWARE ENGINEERING (57)
- COMPUTER SCIENCE THEORY METHODS (4)
- COMPUTER SCIENCE ARTIFICIAL INTELLIGENCE (3)
- ENGINEERING ELECTRICAL ELECTRONIC (3)
- COMPUTER SCIENCE INFORMATION SYSTEMS (2)

more options / values...

Refine

Document Types

- PROCEEDINGS PAPER (63)

Refine

Organizations-Enhanced

- RIGA TECHNICAL UNIVERSITY (4)
- UNIVERSITE DE SFAX (4)

Sort by: Date Times Cited Usage Count Relevance More 1 of 7

Select Page Export... Add to Marked List

Analyze Results
Create Citation Report

1. **A Framework to Support Behavioral Design Pattern Detection from Software Execution Data**
By: Liu, Cong; van Dongen, Boudewijn; Assy, Nour; et al.
Conference: 13th International Conference on Evaluation of Novel Approaches to Software Engineering Location: Funchal, PORTUGAL Date: MAR 23-24, 2018
PROCEEDINGS OF THE 13TH INTERNATIONAL CONFERENCE ON EVALUATION OF NOVEL APPROACHES TO SOFTWARE ENGINEERING Pages: 65-76 Published: 2018
View Abstract
Times Cited: 0 *(from Web of Science Core Collection)*
Usage Count

2. **An Action Research Study towards the Use of Cloud Computing Scenarios in Undergraduate Computer Science Courses**
By: da Silva Filho, Heleno Cardoso; Carneiro, Glauco de Figueiredo
Conference: 13th International Conference on Evaluation of Novel Approaches to Software Engineering Location: Funchal, PORTUGAL Date: MAR 23-24, 2018
PROCEEDINGS OF THE 13TH INTERNATIONAL CONFERENCE ON EVALUATION OF NOVEL APPROACHES TO SOFTWARE ENGINEERING Pages: 15-25 Published: 2018
View Abstract
Times Cited: 0 *(from Web of Science Core Collection)*
Usage Count

3. **Problem-based Elicitation of Security Requirements The ProCOR Method**
By: Wirtz, Roman; Heisel, Maritta; Meis, Rene; et al.
Conference: 13th International Conference on Evaluation of Novel Approaches to Software Engineering Location: Funchal, PORTUGAL Date: MAR 23-24, 2018
PROCEEDINGS OF THE 13TH INTERNATIONAL CONFERENCE ON EVALUATION OF NOVEL APPROACHES TO SOFTWARE ENGINEERING Pages: 26-38 Published: 2018
View Abstract
Times Cited: 0 *(from Web of Science Core Collection)*
Usage Count

4. **Incremental Bidirectional Transformations: Applying QVT Relations to the Families to Persons Benchmark**
By: Westfechtel, Bernhard
Conference: 13th International Conference on Evaluation of Novel Approaches to Software Engineering Location: Funchal, PORTUGAL Date: MAR 23-24, 2018
PROCEEDINGS OF THE 13TH INTERNATIONAL CONFERENCE ON EVALUATION OF NOVEL APPROACHES TO SOFTWARE ENGINEERING Pages: 39-53 Published: 2018
View Abstract
Times Cited: 0 *(from Web of Science Core Collection)*
Usage Count

5. **Adopting Collaborative Games into Agile Requirements Engineering**
By: Przybylek, Adam; Zakrzewski, Mateusz
Conference: 13th International Conference on Evaluation of Novel Approaches to Software Engineering Location: Funchal, PORTUGAL Date: MAR 23-24, 2018
Times Cited: 3 *(from Web of Science Core Collection)*
Usage Count

- CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE CNRS (3)
- ROYAL MELBOURNE INSTITUTE OF TECHNOLOGY RMIT (3)
- UNIVERSITE DE CARTHAGE (3)

more options / values...

Refine

Funding Agencies

Authors

Source Titles

View all options

For advanced refine options, use

Analyze Results

PROCEEDINGS OF THE 13TH INTERNATIONAL CONFERENCE ON EVALUATION OF NOVEL APPROACHES TO SOFTWARE ENGINEERING Pages: 54-64 Published: 2018

View Abstract

6. **Optimized Realization of Software Components with Flexible OpenCL Functionality**

By: Campeanu, Gabriel; Carlson, Jan; Sentilles, Severine
Conference: 13th International Conference on Evaluation of Novel Approaches to Software Engineering Location: Funchal, PORTUGAL Date: MAR 23-24, 2018
PROCEEDINGS OF THE 13TH INTERNATIONAL CONFERENCE ON EVALUATION OF NOVEL APPROACHES TO SOFTWARE ENGINEERING Pages: 77-88 Published: 2018

View Abstract

Times Cited: 0
(from Web of Science Core Collection)

Usage Count

7. **A New Approach for Optimal Implementation of Multi-core Reconfigurable Real-time Systems**

By: Lakhdhar, Wafa; Mzid, Rania; Khalgui, Mohamed; et al.
Conference: 13th International Conference on Evaluation of Novel Approaches to Software Engineering Location: Funchal, PORTUGAL Date: MAR 23-24, 2018
PROCEEDINGS OF THE 13TH INTERNATIONAL CONFERENCE ON EVALUATION OF NOVEL APPROACHES TO SOFTWARE ENGINEERING Pages: 89-98 Published: 2018

View Abstract

Times Cited: 0
(from Web of Science Core Collection)

Usage Count

8. **Mapping of Periodic Tasks in Reconfigurable Heterogeneous Multi-core Platforms**

By: Gammoudi, Aymen; Chillet, Daniel; Khalgui, Mohamed; et al.
Conference: 13th International Conference on Evaluation of Novel Approaches to Software Engineering Location: Funchal, PORTUGAL Date: MAR 23-24, 2018
PROCEEDINGS OF THE 13TH INTERNATIONAL CONFERENCE ON EVALUATION OF NOVEL APPROACHES TO SOFTWARE ENGINEERING Pages: 99-110 Published: 2018

View Abstract

Times Cited: 1
(from Web of Science Core Collection)

Usage Count

9. **Refactoring Object-Oriented Applications for a Deployment in the Cloud Workflow Generation based on Static Analysis of Source Code**

By: Selmadji, Anfel; Seriai, Abdelhak-Djamel; Bouziane, Hinde Lilia; et al.
Conference: 13th International Conference on Evaluation of Novel Approaches to Software Engineering Location: Funchal, PORTUGAL Date: MAR 23-24, 2018
PROCEEDINGS OF THE 13TH INTERNATIONAL CONFERENCE ON EVALUATION OF NOVEL APPROACHES TO SOFTWARE ENGINEERING Pages: 111-123 Published: 2018

View Abstract

Times Cited: 0
(from Web of Science Core Collection)

Usage Count

10. **Using COSMIC FSM Method to Analyze the Impact of Functional Changes in Business Process Models**

By: Khlif, Wiem; Sellami, Asma; Haoues, Mariem; et al.
Conference: 13th International Conference on Evaluation of Novel Approaches to Software Engineering Location: Funchal, PORTUGAL Date: MAR 23-24, 2018
PROCEEDINGS OF THE 13TH INTERNATIONAL CONFERENCE ON EVALUATION OF NOVEL APPROACHES TO SOFTWARE ENGINEERING Pages: 124-136 Published: 2018

View Abstract

Times Cited: 0
(from Web of Science Core Collection)

Usage Count

Select Page Export... Add to Marked List

Sort by: Date Times Cited Usage Count Relevance More 1 of 7

Show: 10 per page

63 records matched your query of the 64,932,297 in the data limits you selected.
Key: = Structure available.