Analyzing the Instability of the Core Components of Software Projects

Lerina Aversano, Daniela Guardabascio, Maria Tortorella
Department of Engineering, University of Sannio, p.zza Roma 21, 82100 Benevento, Italy.
{aversano, guardabascio, tortorella}@unisannio.it

Abstract

Open source software projects represent a significant source of existing pieces of software to be identified and used to implement new or emerging requirements. However, the high complexity of the existing software systems makes difficult the identification of software components to be reused in other systems and the evaluation of their quality. This paper proposes an approach aiming at identifying the core components of a software system and proposing and evaluating some metrics for measuring the evolution of their architecture instability across multiple releases. Then, the paper analyses how the architecture of core components of a software system evolves respect to the whole system. It also investigates the different factors influencing the instability of the core components and it verifies if it decreases across multiple releases assumed that this is a good indication that they can constitute a good candidate to be reused.

1. Introduction

The availability of large repositories of open source software projects makes concrete the possibility of exploiting existing pieces of software to face new or emerging requirements. However, software systems are becoming more and more complex, making very difficult the hard task of identifying existing software components to be reused in other systems. Offering an effective support to software engineers in this complex task requires the definition of advanced methods and tools helping to achieve a view of a software architecture, identifying software components to be candidate for being reused and obtaining quantitative information regarding the quality of such software components.

This paper presents a study on the identification of the architectural core of a software system and on the analysis of its stability with reference to the software system itself and across multiple releases. The architectural core of the software system is composed of software components on which the large part of activities and functionality of the software system are concentrated. It is desirable that the architectural core of a software system should be more stable than the system itself [3] for being a good candidate for reuse. This aspect can be analyzed by evaluating the architectural stability of a considered software system [4] and related core. It represents the extent at which the software systems can endure changes in requirements, while leaving its architecture intact [11]. Stability information refers to a non-functional attribute that is significant for a software engineer for considering a component to reuse in a different project. Actually, higher the stability of a software component is, more easily it can be reused in a new software system.

The analysis performed in this paper is based on the historical data regarding the evolution of a set of software systems, with the aim of identifying their architecture cores, highlighting how their instability evolves across multiple releases. With this in mind, the paper proposes some metrics for measuring the architecture instability, and evaluates the instability trends of the cores across multiple releases of a set of software systems; then, the paper compares the core instability trends to the ones of their software systems.

The performed analysis considers software systems developed with different evolution trends and concerning different application domains. Due to the large availability of open source software projects and related releases, the study analyzes a set of such a kind of projects.

The remainder of this paper is organized as follows: Section 2 reports the main related works; Section 3 describes scope and definition of the analysis; Section 4 analyses the data source analysis and selection; the successive section presents the instability metrics adopted for the analysis of the software architectures. Section 6 discusses the obtained results. Finally, the last section presents concluding remarks and outlines future work.

2. Related Work

In the literature, there are several research works addressing the analysis of the architecture of an existing software system [14, 17, 18]. Initial architecture analysis methods, such as SAAM [11], SAAMER [13], APLSM [5], focused on various aspects of architectures, like modifiability, maintainability, or reusability.

Architectural stability is an important factor for software reuse, during either the reusable asset selection or library upgrades. In [6], two sets of metrics that measure the architectural stability and the evolution of software projects in the context of software reuse are introduced. The first set of architectural stability metrics measures the degree of consistency between consecutive versions of the same system and considers the common architectural elements. The second set of architectural evolution metrics quantifies the architectural evolution between consecutive versions of the same system and considers the newly introduced architectural elements, as well as their interaction with the remaining elements of the system.

In [7], the authors highlight that the available architectural stability measures lack some other important structural aspects of the architecture, such as inter-package connections (IPCs), and propose a new metric to measure the architectural stability of object-oriented (OO) system in terms of IPC. The contribution provides information regarding the package and its evolution, enabling the monitoring of trends in software evolution. In order to evaluate the stability of an OO software system developed using an agile design similar to Extreme Programming, the authors in [15] validated the metrics SDI - System Design Instability. In [8], the authors emphasize the diffusion of open source projects, asserting that their success is due to several factors, such as the fact that the developer is also the current user, the sound and the modularity of the architecture. The modularity is the most important factor and the authors apply a quantitative analysis of open source Java-based projects for measuring the level of modularity in open source projects.

In [10], the authors carried out an analysis to evaluate the stability of a software system. They analyzed the entire history of a system going to consider the changes implemented in the next release of the software product. This analysis has been combined with predictive analysis, in order to validate the forecasts of the project under consideration.

In [1] the authors proposed an empirical study of the class growth and the SDI metric in two OO systems, developed using an agile process similar to Extreme Programming (XP), concluding that the growth of the systems class follows observable trends. Moreover, the authors observed that the SDI metric can indicate project progress with certain trends, and the SDI metric is correlated with XP activities.

In [2], it has been analyzed the impact of refactoring on class and architecture stability. Actually, when applying refactoring it is necessary to assess how the changes to be performed could impact the entire system.

The study of the instability is not limited to software-only product, but also to libraries developed by third parties and integrated into real software products. In order to analyze the stability of a software system, in [16], the authors, focused on the use of such libraries.

The stability of a software product is important in order to facilitate maintainability and its evolution. In [19] the authors present a set of metrics designed to measure the stability of an open source software product such as: version Stability, branch stability, structure stability and aggregate stability.

In [9], the authors demonstrated the effectiveness of techniques based on the concepts that using software stability model (SSM) and Knowledge Maps (KMs), it is possible to realize software solutions that do not need excessive alterations, changes or additions.

In [12], a taxonomy of architectural smells, metrics, and their impacted quality properties is provided relating these smells to maintenance and evolution areas.

In [3], an initial definition of instability metrics was proposed and used for measuring the instability of the subset of software components responsible for the large part of interactions within the software system. That subset of components represented the core of a software system and could indicate software assets to be candidate for reuse. The metrics proposed in [3] have been considered in this paper for defining new metrics for evaluating a software system instability. Specifically, the contribution of this paper to the literature can be sensitized as follows:

- the instability of a software projects is not evaluated with reference to a single release but, instead, to the moving from a release to another;
- a set of metrics is defined to assess the instability splitting the design from the interaction point of view;
- the proposed metrics are validated with a detailed analysis that involved the assessment of 6 software projects over numerous releases.

3. Scope Definition of the Analysis

This section provides an overview of the objective of the performed study for analyzing the architecture stability of software systems, and describes the steps performed. In particular, the study involved different releases of 6 open source software projects and focuses on the instability of the software architecture and evolution of the main architectural components.

The main considered characteristic is the instability of software projects and related architecture components, analyzed with reference to a set of different releases of the software projects. Thus, the objectives definition specifically requires the understanding of how to measure the instability of the architecture of a software system. In particular, the study aims at identifying the architecture core of a software system, that is those components on which the large part of the interactions, about 80%, are concentrated. Once the architecture core is identified, the analysis explores if architectural variations can be observed across multiple releases of the software system, and if a relation with the instability metrics of the whole software system exists. Then, the study aims to establish in a preliminary way at what extent the software architecture of a set of software projects is instable, with reference to its core components, and by considering its evolution history.

For performing the analysis above, the paper objectives can be expressed by the following three research questions.

RQ1. *To what extent the core is more stable than the full system?* This research question aims to understand whether the instability of the full system evaluated at the package level through Project Design Instability and Project Call Instability is different respect to the instability of the core assessed at the package level, Core Design Instability and Core Call Instability.

RQ2. *Is the project instability mainly due to the adding of new packages, removal of old packages or change of existing packages?* This research questions aims to analyze the different contribution coming from the added, removed and changed packages in the evaluation of the instability of the full project. The aim is to understand if the full system instability is mainly due to the changes of the software system design or to the one of its packages, when the software system evolves.

RQ3. *Is the core instability mainly due to the adding of new core packages, removal of old core packages or change of existing core packages?* The question analyses the same aspect investigated in the previous research question with reference to the software system core. Therefore, the aim is to understand if the main contribution to the core instability comes from the added, removed or changed packages to the core during the evolution of the considered software system.

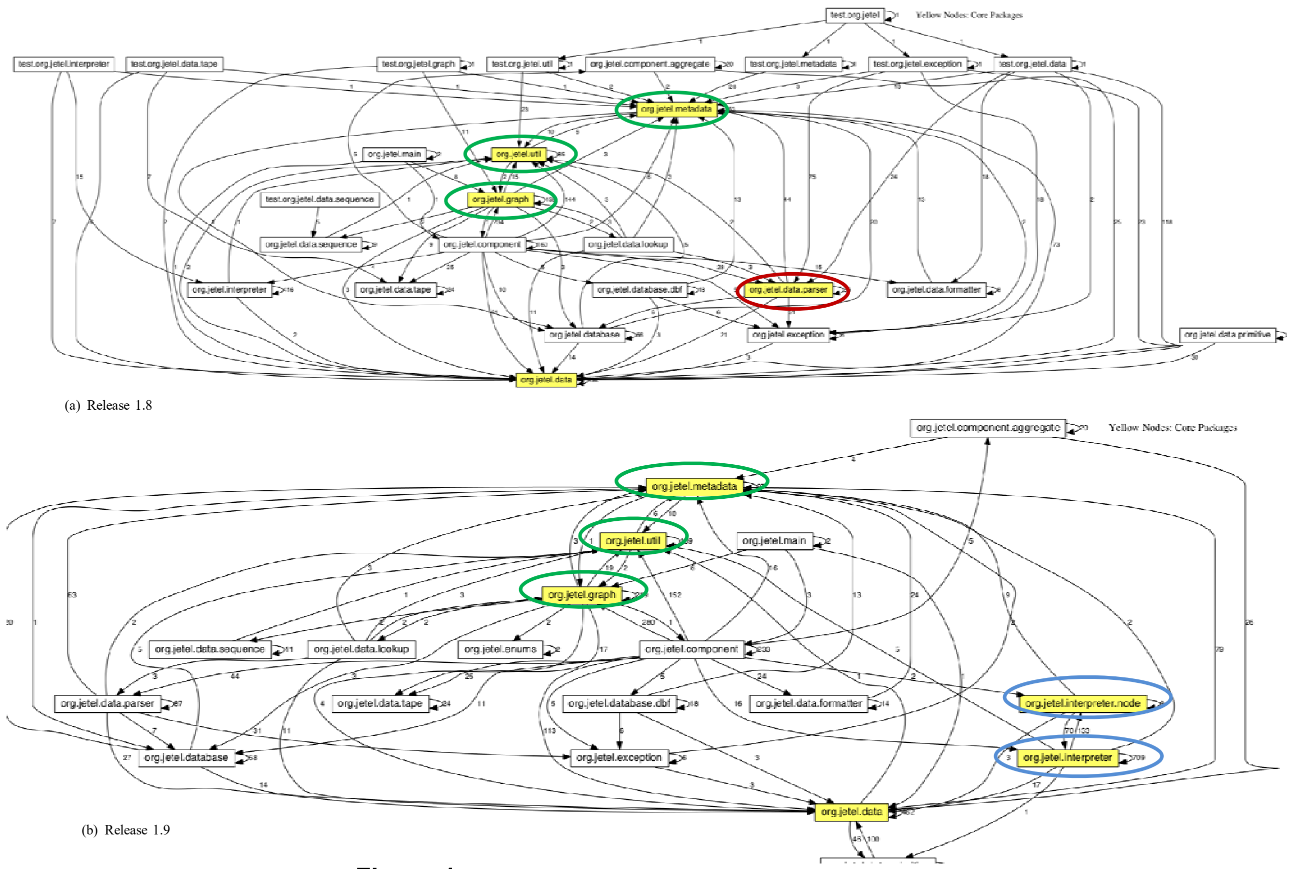
4. Datasource Analysis and Data Extraction

The principal source of information for the study are the source code repositories. In particular, SourceForge and GitHub were the primary source of the data required from the performed study. The projects hosted in SourceForge can have different status, classified as: Active, Inactive, Planning, Pre-Alpha, Alpha, Beta, Production/Stable and Mature. Actually, only the Active and Mature projects were considered, as the inclusion of immature or incomplete projects could influence the results. Some information regarding the project category were directly extracted from the SourceForge Platform by using a Python script. The output was an integrated database where all the required information was cleaned and reorganized. In order to conduct the analysis, an initial sampling was performed to focus on few projects. For the selection of the software projects to be considered the attention was focused only on those projects written by using Java programming language and classified as Mature projects. Then, the obtained list of projects was filtered again, by considering the project "popularity". Finally, the projects were manually validated, since it occurred that: some projects were linked to an empty repository; some projects had a fewer number of minor releases; and some projects were erroneously classified as Java projects. After the above filtering process, it was possible to identify the projects on which the experimental analysis could be concentrated. They were six projects, listed in Table 1.

A first evolution history was obtained for the 6 selected projects. With this in mind, the commits on the SourceForge and GitHub project repositories were considered and the analysis of their log files permitted to reconstruct the history of each project and choose the releases to be considered. They have been selected in order to obtain a reasonable distribution over the timeline of the considered projects and mainly select major releases. The last two column of Table 1 indicate the number of the releases available when the project releases were downloaded and the one of the actually considered releases. It is possible to verify from the table that the number of the available releases is much higher than that one of the considered releases. This depends on the fact that many available releases contained minor changes and it was decided to analyze just the major releases, whose comparison could more easily highlight a project instability, if it existed. The number of the considered releases of the projects are listed in Table 1. Therefore, project architectures and related evolution were analyzed by considering the packages of these releases and reciprocal interactions.

Table 1. Overview of the analyzed projects

PROJECT NAME	DESCRIPTION	WEB SITE	REPOSITORY	NUMBER OF AVAILABLE RELEASES	NUMBER OF CONSIDERED RELEASES
CloverETL	A Java ETL framework which transforms structured/unstructured data	http://sourceforge.net/projects/cloveretl.berlios/?source=directory	SourceForge	37	19
JPPF	A system making easy to parallelize computationally intensive tasks and execute them on a Grid.	http://www.jppf.org/	SourceForge	129	26
MessAdmin	A HttpSession administration and notification system application plug-in for J2EE Web Applications	http://messadmin.sourceforge.net/	SourceForge	25	16
OpenNMS	A Java based fault and performance management system	http://www.opennms.org/	GIT	181	17
OpenSearchServer	An open source search engine with RESTful API and crawlers	http://www.opensearchserver.com/	GIT	54	7
Sesame	A de-facto standard framework for processing RDF data	http://rdf4j.org/	SourceForge - GIT	108	14



(a) Release 1.8

(b) Release 1.9

Figure 1. Interaction graphs of project CloverETL

Besides the instability measures of the full projects, the ones of the project architectural core components were considered, it is opportune to anticipate that the architectural cores were identified for each releases of the six projects during the process of analysis. The architectural core of a release of a software project represents the set of main packages of that release. Specifically, the study considers that the architectural core is composed of those packages that produced at least 80% of the total interactions among the packages.

The analyzed interactions among the packages were: fan-in, the number of interactions from packages toward one package; fan-out, the number of interactions of one package to other packages; self-call, numbers of interaction of one package to the package itself. The identification of the architectural core considered only the values of the fan-in, that is the most relevant information. The fun-out was not considered for avoiding duplications of the number of calls already considered in the fun-in value; while the self-calls do not impact on the fun-in and fun-out values of the involved packages.

The set of packages composing the core may change from one release to the successive one. Figure 1 includes the interaction graphs of project CloverETL with reference to two successive releases. Figure1a concerns release 1.8 and the yellow colored nodes regards packages belongingto the core. Figure 1b includes the evolution of release 1.8 toward release 1.9, and indicates that, besides the number of project packages, even the core packages, highlighted with colored nodes, changed respect the previous release. The green colored circles highlight the packages of the core of release 1.8 that change in release 1.9; the blue circles indicate the packages added to the core in release 1.9; and the red circle in release 1.8 is removed form the core of the successive release.

5. Instability Metrics

To analyze the instability of a software system and related core, two set of metrics were used. Their definition evolves the metrics proposed in [3], by considering also the modified software components, and specializing them to packages and classes. Specifically, instability metrics have been grouped in two sets: Design Instability and Interaction Instability. Both metrics are evaluated with reference to both all the project and the project core.

The first kind of metric, named Design Instability (DI), is based on the evaluation of how the software system packages evolve going from a release, N, to the next considered one, N+1. The evolution actions regarding a package concern its adding, removal and modification. Prefix P or C to the DI metric indicate if it is concerns the Project Design Instability (PDI) or the Core Design Instability (CDI). The Interaction Instability metrics, referred as Calls Instability (CI), regards the interactions, in terms of fan-in and self-call values of each analyzed software component. In this case, the evolution concerns the modification of the number of interaction, which can be due to the adding of new interactions and/or removal of the existing ones. Even in this case, the prefix P or C to the CI metric indicates if it is regards the Project Call Instability (PCI) or the Core Call Instability (CCI).

The Design Instability (DI) aims to evaluate the changes performed on the architecture/core of release N of a considered software system when it evolves toward release N+1. Then, DI can be defined as follows:

$$DI = \frac{changed\_comp + added\_comp + removed\_comp}{number\_comp + changed\_comp + added\_comp + remove\_comp}$$

where:

- changed_comp is the number of software components of release N of the considered software project/core, that have been changed for obtaining release N+1;
- added_comp is the number of new software components added to release N of the considered software project/core for evolving it toward release N+1;
- removed_comp is the number of software components removed from release N of the considered software project/core for evolving it toward release N+1;
- number_comp is the number of software components composing release N of the considered software project/core.

The Calls Instability (CI) is referred to the changes of the interactions between the software components of release N of a considered software system/core for evolving it toward release N+1. It is computed as follows:

$$CI = \frac{added\_interactions + removed\_interactions}{total\_interactions + added\_interactions + removed\_interactions}$$

where:

- total_interactions is the total number of interactions between the software components belonging to the architecture of release N of the considered software system/core. Calls starting from external software components are excluded.
- added_interactions is the number of new software components interactions added to release N of the considered software system/core for evolving it toward release N+1 after changes are executed.
- removed_interactions is the number of software components interactions removed from release N of the software system/core for evolving it toward release N+1 after changes are performed.

Metrics PDI/CDI and PCI/CCI measure how much the packages of a software system/core change from release N to the next considered one, release N+1. They can assume a value in the range 0 to 1. Smaller their values are, less the system changes, and consequently more stable it is.

For answering the research questions RQ2 and RQ3, other metrics have been evaluated. They represent the contribution coming from the added (*DI_a* and *CI_a*), removed (*DI_r* and *CI_r*) and changed (*DI_c*) packages and interactions for the evaluation of the Design and Call Instability with reference to the project and core.

For the Design Instability, *DI_a*, *DI_r* and *DI_c* are evaluated as it follows:

$$DI\_a = \frac{added\_comp}{number\_comp + added\_comp}$$

$$DI\_r = \frac{removed\_comp}{number\_comp + remove\_comp}$$

$$DI\_c = \frac{changed\_comp}{number\_comp + changed\_comp}$$

For the Call Instability, *CI_a*, *CI_r* are the following:

$$CI\_a = \frac{added\_interactions}{total\_interactions + added\_interactions}$$

$$CI\_r = \frac{removed\_interactions}{total\_interactions + removed\_interactions}$$

With reference to the project they are called *PDI_a*, *PDI_r*, *PDI_c*, *PCI_a* and *PCI_r*. While they are *CDI_a*, *CDI_r*, *CDI_c*, *CCI_a* and *CCI_r* for the core.

6. Achieved results

This section discusses the results achieved in our study aimed at responding to our three research questions.

RQ1. *To what extent the core is more stable than the full system?*

This question aimed at investigating how differently the instability measures of the whole system are respect to the instability measures of its core components. It is useful to understand if the architectural core of a software system are more stable than the software system they belong to. For answering this question, a quantitative investigation, supported by graphical representations showing the instability trends of the analyzed software system, is reported. Specifically, the trends of the Design Instability measures, PDI and CDI, are depicted in Figure 2; while Figure 3 shows the trends of the call instability, PCI and CCI. Figure 2 points out that the Core Design Instability assumes values close to the ones of the full project. Specifically, the core instability assumes values higher than the instability of the entire project in projects CloverETL, JPPF, and OpenNMS. In OpenSearchServer, MessAdmin and Sesame their trends are very close, even if the core instability tends to be higher than the system one. This is an unexpected result, as it is generally expected that the core components have a greater responsibility; then they should be more stable.

Figure 3 shows the Call Instability trend. It is possible to note that even the trends of this metric analyzed for all the considered projects and their cores are close, even if they are very oscillating and the core instability is often higher than the project instability. This again indicates the contrary to what is desirable.

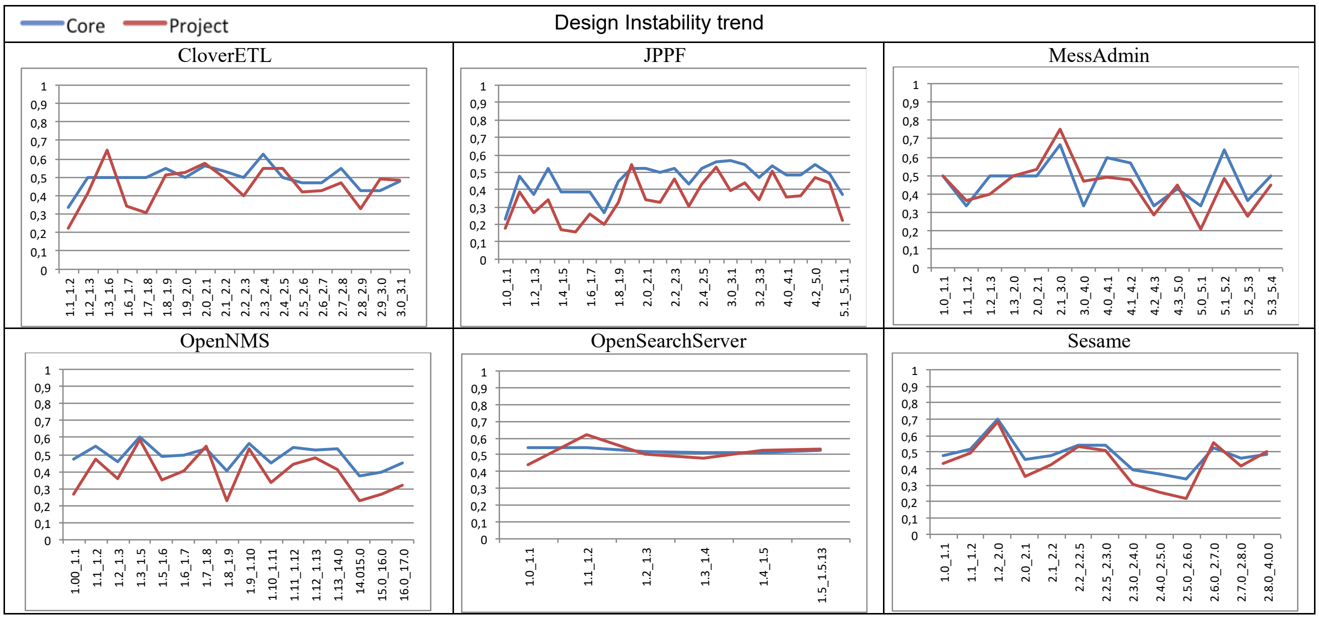


Figure 2. Design Instability trend for the analyzed projects and their core

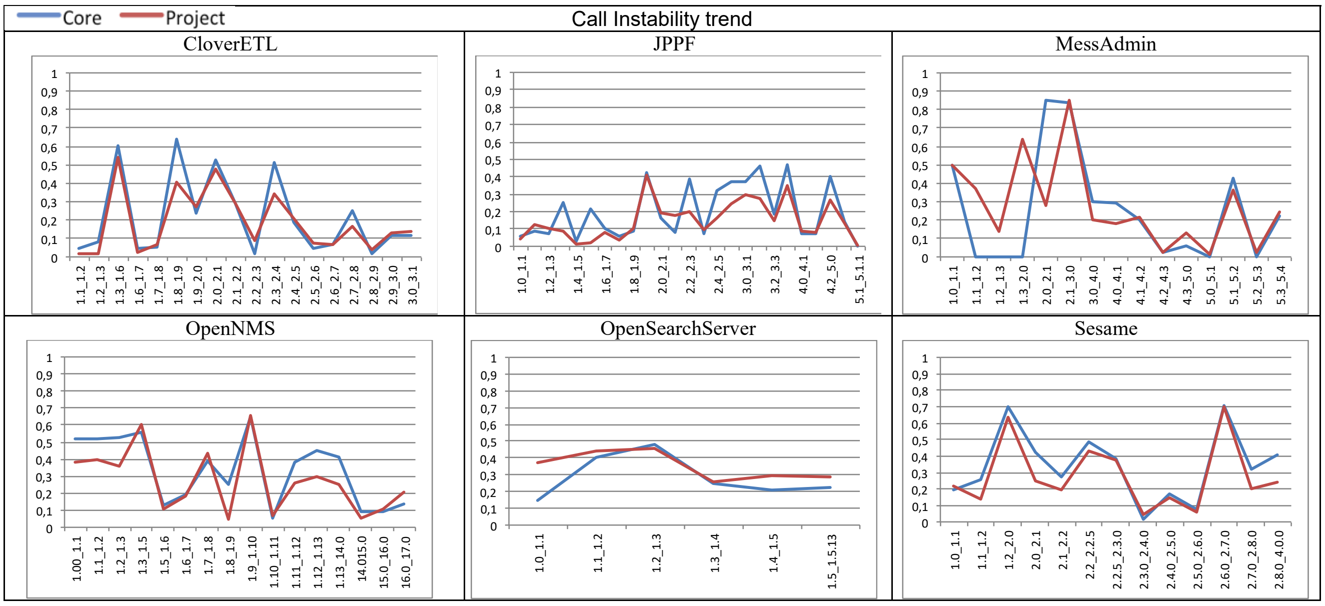


Figure 3. Call Instability trend for the analyzed projects and their core

This results are even more unexpected if Figure 4 is analyzed. It shows the number of packages of each analyzed software system release and that one of the packages of the related core. In particular, it emerges that the core packages are relatively few respect to the packages number of the project. Moreover, the number of core packages changes much less than the total number of the packages. This could probably justify the higher values of the call instability for the core packages.

This analysis suggests that a deeper investigation concerning the project and its core instability should be performed for a complete understanding of their values and related variations.

RQ2. *Is the project instability mainly due to the adding of new packages, removal of old packages or change of existing packages?*

The aims is to understand which are the principal causes of the project instability. Then, the instability measure has been divided for highlighting the contribution coming from the added and removed and changed packages. Figure 5 shows the trend of the three different instability components with reference to the Design Instability. The blue line indicates the instability of the added packages, PDI_a, the red line indicates the one due to the removed packages, PDI_r; and the green line is the instability due the changed packages, PDI_c.

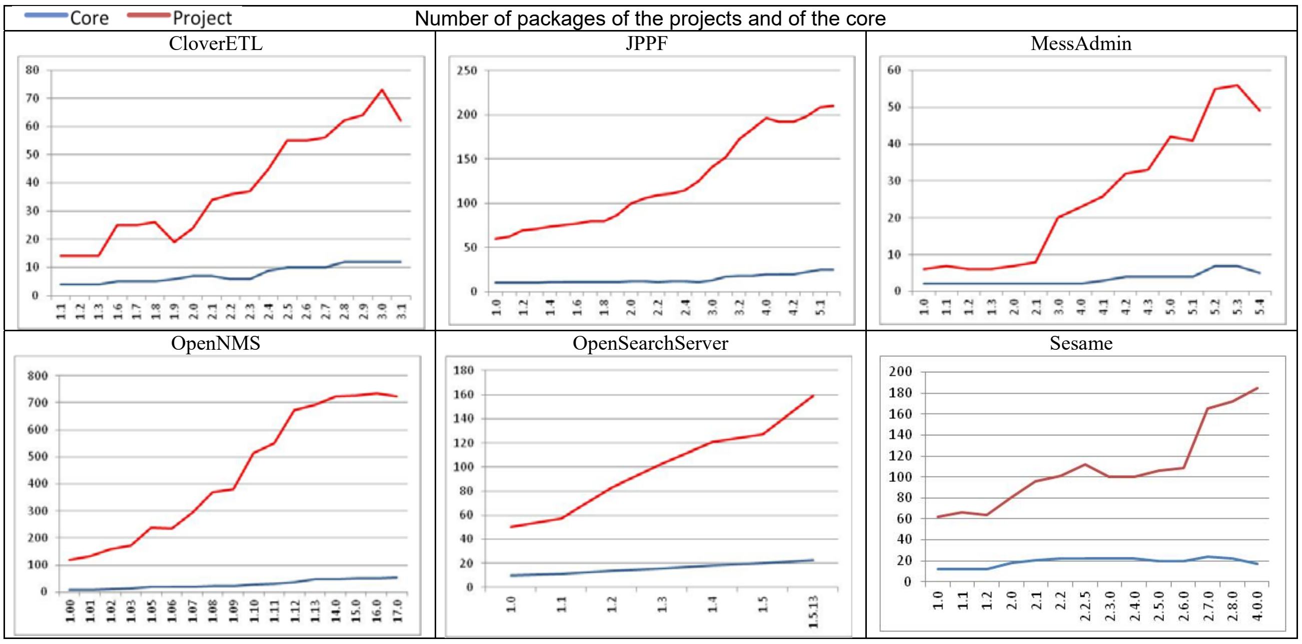


Figure 4. Number of packages of the projects and number of packages belonging to their core

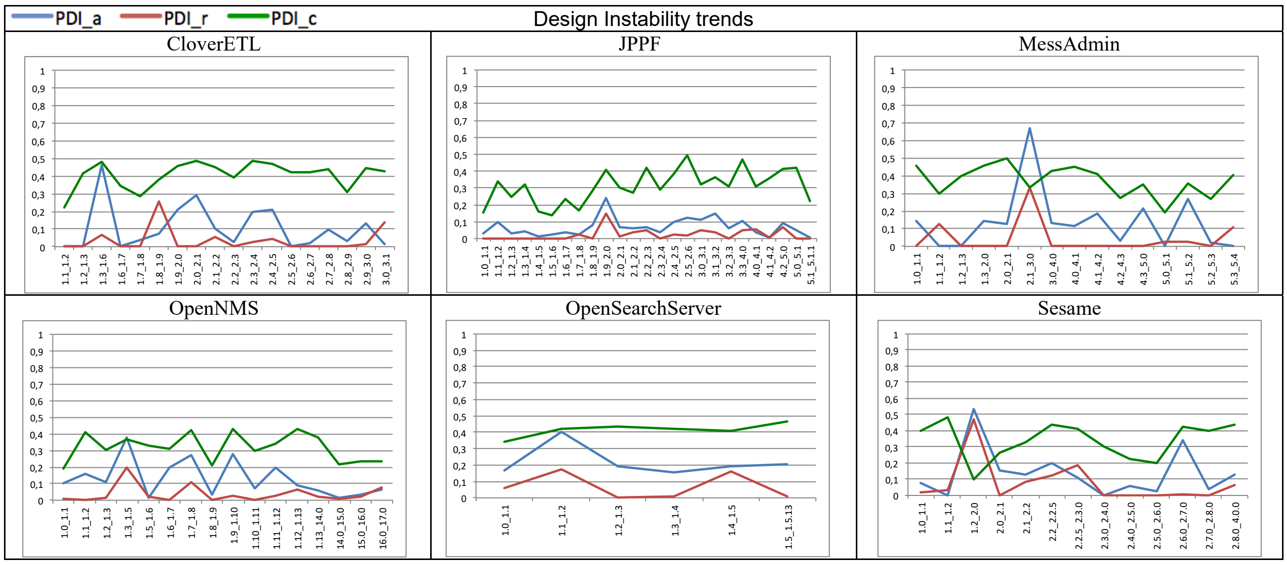


Figure 5. Design Instability trends divided for added, removed and changed packages of the entire system

It is possible to note that in all the analyzed systems the different instability components assume values less than 0.5, even if the contribution of the changed packages is much higher than the ones of the added and removed packages. The trends of PDI_a and PDI_r are almost similar and assume low values. This indicates that the large part of changes performed on the packages are mainly due to changes of the current system packages rather than the system structure. Figure 6 also depicts the number of added, removed and changed packages in all the analyzed software systems. There are also packages that do not change, but they are not traced in the graphics, as they do not contribute to the instability measure. It emerges from the figure that the number of changed packages grows going from the first to the last considered release. On the contrary, the numbers of added and removed packages assume values that are lower and about constant going from the first to the last release. Similarly, Figure 7 depicts the trends for the Call Instability divided into instability caused by the added interactions, PCI_a indicated with blue lines, and the one due to the removed interactions, PCI_r traced with the red lines.

It can be observed that in all the projects the instability due to the adding of new interactions between packages is generally higher than the one due to interaction removal. This causes an increasing of the package coupling during the evolution of the software systems, and this implies a decreasing of the quality of the software systems.

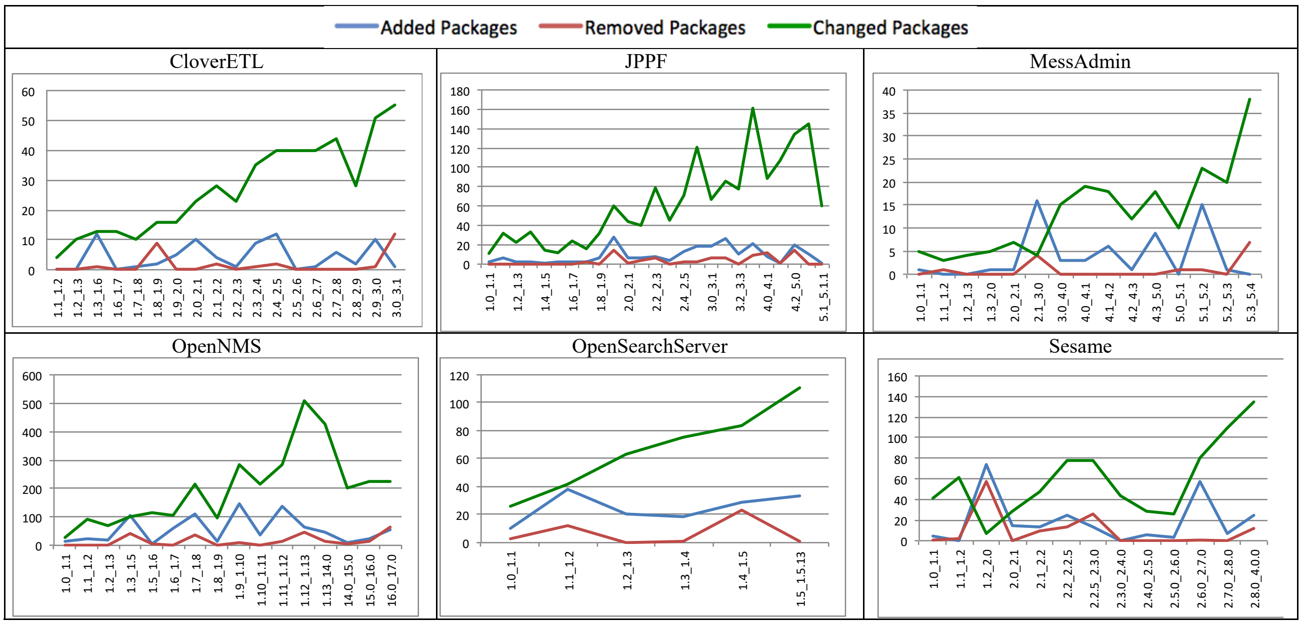


Figure 6. Trends of the number of added, removed and changed packages in the releases of the software systems

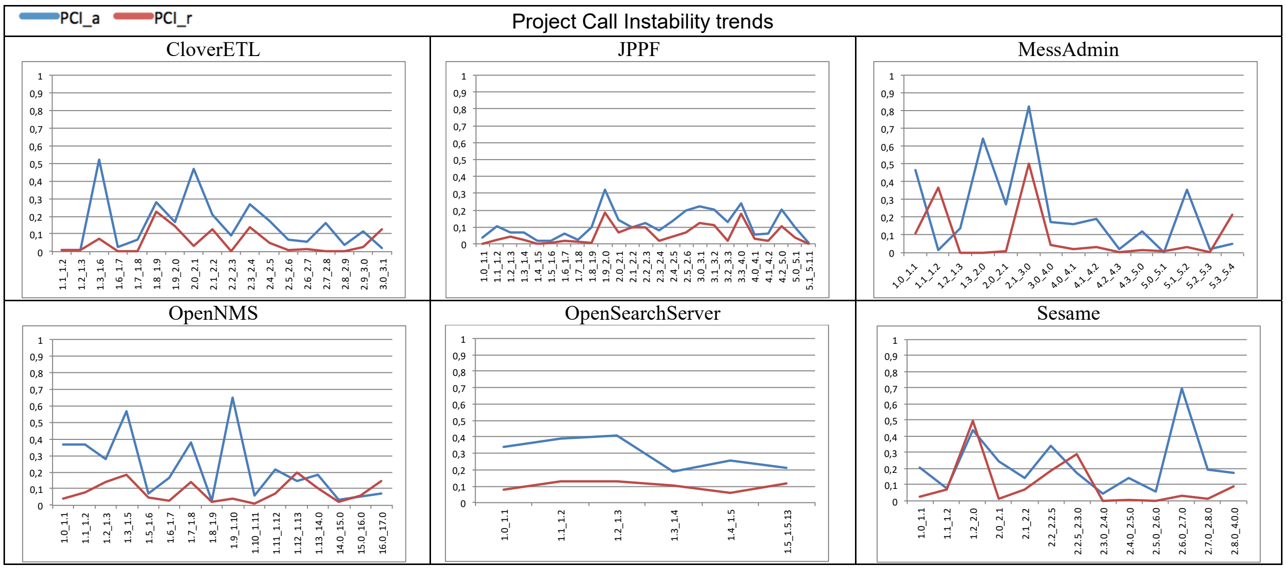


Figure 7. Project Call Instability trends divided into instability due to the added and removed interactions

RQ3. *Is the core instability mainly due to the adding of new core packages, removal of old core packages or change of existing core packages?*

This research question analyzes the instability of the core packages, paying attention to the three considered instability sources regarding the adding, removal and changing packages and reciprocal interactions. Figure 8 includes the graphics of the Core Design Instability, split into the three cited components, highlighting that the core instability in all the systems is mainly due to the change performed in the packages belonging to the core, rather than their addition and removal. Actually, this is confirmed by the graphics in Figure 9, containing the absolute numbers of the added, removed and changed packages during the core evolution. Just projects MessAdmin and Sesame make few exceptions in some releases, where the addition and removal instability is higher than the change instability. Moreover, Figure 9 shows that the number of the removed packages from the cores is generally higher than the number of the added packages to the cores. Finally, Figure10 shows the distinction of the Core Call Instability in Instability due to the adding of new interactions and the one caused by the removal of old interactions, within the system core and going from one release to the successive one. The figure indicates that the core instability due to the adding of new

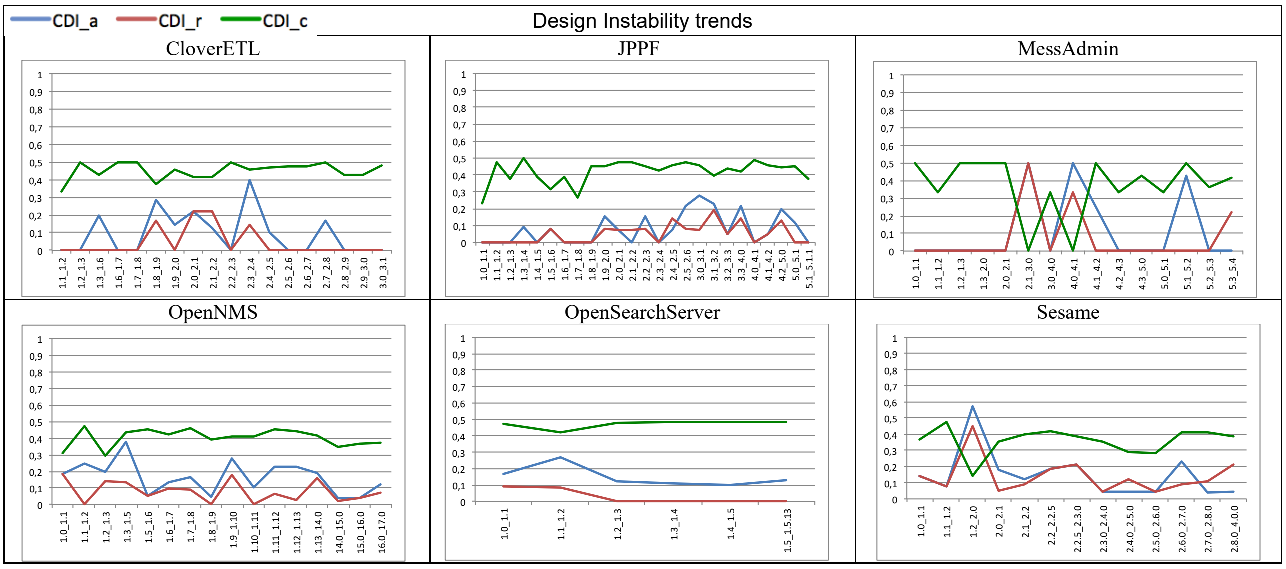


Figure 8. Design Instability trends divided for added, removed and changed packages of the core system

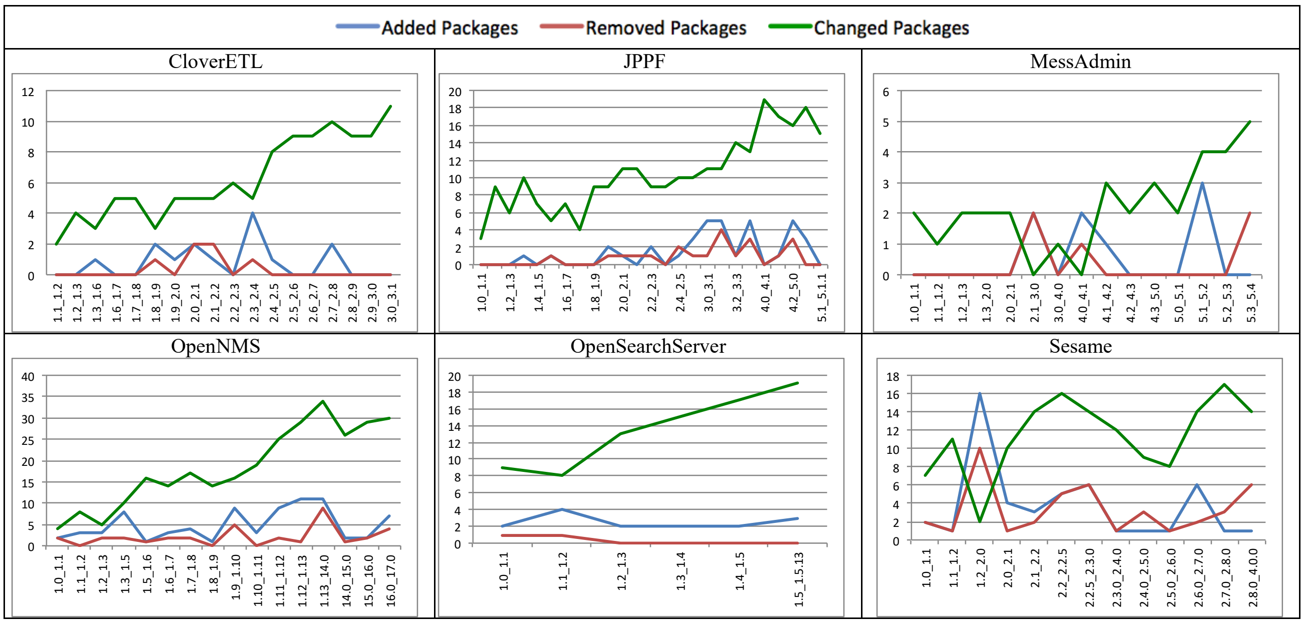


Figure 9. Trends of the added, removed and changed packages of the cores in the releases of the software systems

interactions is always higher than the one concerning the removal of old interactions. Even in the case, the adding of new interactions among packages implies an increasing of the coupling within the cores of the systems. Then, this causes a worsening of the quality of the system cores when they evolve from a release to successive one.

7. Conclusion

This paper reported an empirical study for investigating the evolution of architecture instability in 6 different open source system. The study is based on an automatic process for the assessment of metrics used for evaluating the architecture instability. The availability of such an automatic process allowed us to perform a deeper analysis of a large number of releases for each software system. The obtained results highlight the following aspects:

- the instability measure of the core is close to the one of the full project. The analysis of the trend of instability metrics PDI, CDI and PCI, CCI pointed out that it does not exist a significant different behavior*;*
- the Design Instability of the considered systems is mainly due to the changed packages instead of the added and removed ones; while the Interaction Instability is caused by the added interactions; that makes to increase the packages coupling and, then, decreases the quality;

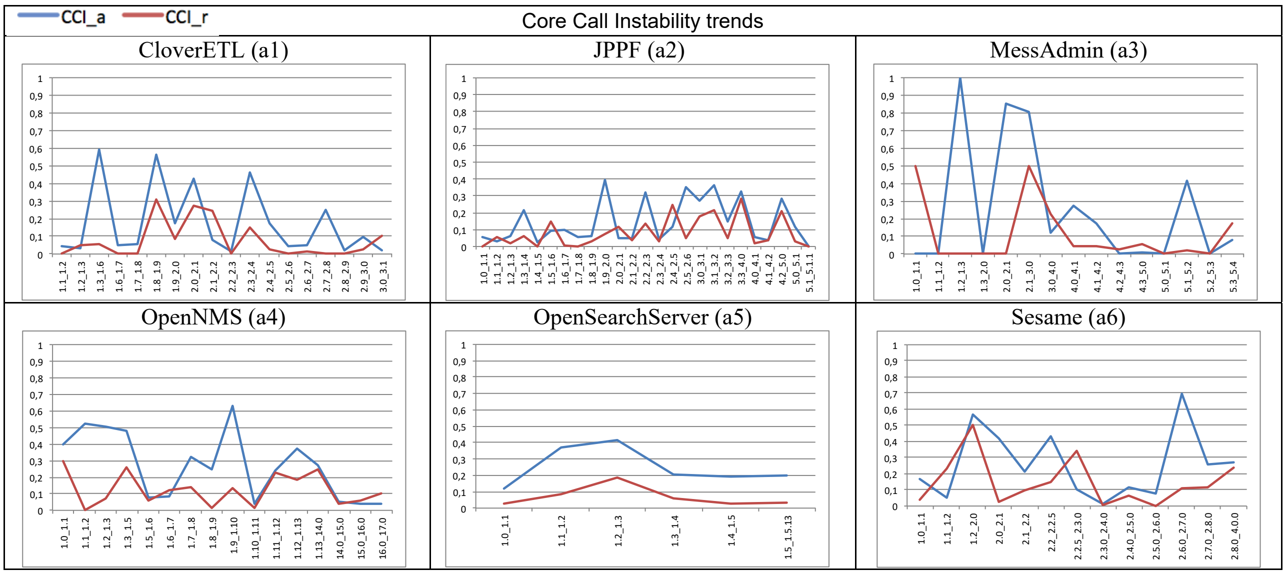


Figure 10. Core Call Instability trends divided into instability due to the added and removed interactions

- the core instability is mainly due to the changes of the core packages; then, an equivalent behavior between the software systems and their cores exists.

There are still a number of factors to be investigated. Firstly, the study can be replicated on additional software systems of different domains and developed using different programming languages. Then, it can be useful to analyze the differences between the proposed instability metrics and other ones found in the literature. Moreover, the presence of the subsets of software component mainly affecting the instability could be further investigated.

8. References

[1] Alshayeb M. and Li W.,"An Empirical Study of System Design Instability Metric and Design Evolution in an Agile Software Process", Journal of Systems and Softwrae, 74 (3), , Elsevier Science Inc, 2005.

[2] Alshayeb M., “The Impact of Refactoring on Class and Architecture Stability”, Journal of Research and Practice in Information Technology, 43(4), November 2011.

[3] Aversano L., Molfetta M., Tortorella M., "Evaluating architecture stability of software projects", Proc of 20th Working Conference on Reverse Engineering, WCRE 2013, Koblenz, Germany, October 14-17, IEEE Computer Society, 2013

[4] Bahsoon R. and Emmerich W., “Architectural Stability”. Proc. of the Confederated International Workshops and Posters on On the Move to Meaningful Internet Systems, 2009.

[5] Bengtsson, P.O. and Bosch, J., “Architecture level prediction of software maintenance”. Proc. of Third European Conference on Software Maintenance and Reengineering, 1999.

[6] Constantinou E. and Stamelos I., “Architectural stability and Evolution Measurement for software reuse”. Proc. of the 30th Annual Symposium on Applied Computing, ACM NY, 2015.

[7] Ebad S. A., Ahmed M. A., “Measuring stability of object-oriented software architectures”, IET Software, 9(3), 2015.

[8] Emanuel A.W.R., Wardoyo R., Istiyanto J.E. ad Mustofa K., “Modularity Index Metrics for Java-Based Open Source Software Projects”, Int. Journal of Advanced Computer Science and Applications, 2(11), 2011.

[9] Fayad M. E. and Flood C. A., “Unified Software Engineering Reuse (USER) using stable analysis, design and architectural patterns”, *Future Technologies Conference (FTC)*, CA, 2016

[10] Jazayeri M., “On Architectural Stability and Evolution”. Proc. of the 7th Europe International Conference on Reliable Software Technologies, Springer-Verlag, 2002

[11] Kazman R., Abowd G., Bass, L., and Webb M., “SAAM: A method for analyzing the properties of software architectures”, Proc. of 16th International Conference on Software Engineering. Sorrento, Italy, Los Alamitos, CA: IEEE Comp. Society, 1994.

[12] Le D. M., Carrillo C., Capilla R. and Medvidovic N., “Relating Architectural Decay and Sustainability of Software Systems”. Proc. of 13th Working IEEE/IFIP Conference on Software Architecture (WICSA), 2016.

[13] Lung C.-H., Bot S., Kalaichelvan K., and Kazman, R., “An approach to software architecture analysis for evolution and reusability”, Proc. of CASCON ’97, 1997.

[14] Medvidovic N., Jakobac V., “Using software evolution to focus architectural recovery”. Automated Software Engineering. 13(2), 2006.

[15] Olague H. M., Etzkorn L. H., Li W. and Cox G., “Assessing design instability in iterative (agile) object-oriented projects”. Journal of Software Maintenance and Evolution., 18: 237–266. 2006.

[16] Raemaekers S., van Deursen A. and Visser J., "Measuring Software Library Stability Through Historical Version Analysis". Proc. of 28th IEEE International Conference on Software Maintenance, 2012.

[17] Sartipi K. and Kontogiannis K., “Component Clustering Based on Maximal Association”. Proc. of IEEE Working Conference on Reverse Engineering, Germany, 2001

[18] Sartipi K. and Kontogiannis K., “Pattern-based Software Architecture Recovery”. Proc. of Second ASERC Workshop on Software Architecture, Alberta, Canada. 2003.

[19] Threm D., Yu L., Ramaswamy S., Sudarsan S. D, “Using Normalized Compression Distance to Measure the Evolutionary Stability of Software Systems”. Proc. of 2015 IEEE 26th Int. Symposium on Software Reliability Engineering, 2015.