Towards a (semi)-automatic reference process to support the reverse engineering and reconstruction of software architectures

Daniel Guamán
Universidad Técnica Particular de Loja
Ecuador
daguaman@utpl.edu.ec
Universidad Politécnica de Madrid
Spain
da.guaman@alumnos.upm.es

Jennifer Pérez
Universidad Politécnica de Madrid
Spain
jperez@etsisi.upm.es

Jessica Díaz
Universidad Politécnica de Madrid
Spain
yesica.diaz@upm.es

ABSTRACT

The purpose of this work is to define a reference process to support the software architecture reconstruction in a systematic and automatic way. This process aims to be applied to the construction and maintenance phases within *Agile methodologies* and *Continuous Integration* processes, where the quick and continuous test and changes at design or coding level can generate an increase or reduction of *technical debt* and *green software levels*. This process is based on phases, activities, techniques, and strategies proposed by related works about reverse engineering and software architecture reconstruction. Specifically, it integrates all of them to create a complete process; which may be a reference process by providing green and technical debt-oriented recommendations during the decision-making of software architecture at design level or coding level. This recommendation phase will be based on algorithms and techniques of Machine Learning, that will allow to apply the process in an Agile way and taking into account previous knowledge.

CCS CONCEPTS

• **Software and its engineering → Software organization and properties → Software system structures**; Software architectures

KEYWORDS

Reverse engineering, software architecture, software reconstruction, green software, software metrics, machine learning, technical debt.

ACM Reference format:
Daniel Guamán, Jennifer Pérez, Jessica Díaz (2018): Towards a (semi)-automatic reference process to support the reverse engineering and reconstruction of software architectures, In Proceedings of 12th European Conference on Software Architecture: Companion Proceedings, Madrid, Spain, September 24-28, 2018 (ECSA'18), 4 pages.
https://doi.org/10.1145/3241403.3241414

1 INTRODUCTION

Software systems must be *maintained* and *evolved* to cope with a changing environment without neglecting the problems that may arise due to architectural design deficiencies, poorly written or unstructured code, errors in the implementation or in the functionalities validation, poor documentation, and obsolete technology. All these changes and problems in some cases lead to generate *technical debt* as exposed by Tom et al. [1], and in fact, most of the life cycle cost refers to software maintenance and evolution.

To avoid such erosion and problems, *re-engineering* helps to reduce the impact of change and its implementation. The *re-engineering* process is defined by Chikofsky and Cross [2] as the analysis and modification of a system to reconstitute it in a new form and the subsequent implementation of it. In terms of r*everse engineering*, this is defined by the same authors as an analysis process (not about change or replication), that generally implies to extract design and construction artifacts to synthesize abstractions that are less dependent on implementation, i.e. to create representations at a higher level of abstraction.

In some cases, the *reverse engineering* process requires the generation of a specification, and specifically an *architecture reconstruction*, defined as "the process by which the architecture of an implemented system is obtained from the existing system" [3]. *Architecture reconstruction* is described by Ducasse and Pollet [4] as the reverse engineering approach where the main objective is to extract, analyze and reconstruct architectural views while preserving qualities that allow to understand, interpret, improve or verify the software architecture. The software engineering community has performed considerable work in reverse engineering and architecture reconstruction.

The current state of society and the new era of information imply a continuous change in the architecture and technology of software systems, guided by quick changes in business, requirements, technology and by society's needs [5]. Software systems are the key enablers of companies to achieve a competitive advantage in the market. Miorandi et al. [5] remark that "we are in a continuous changing environment in which the success of companies depends on the speed through which the change is made". This is one of the main reasons of the widely spread adoption of *Agile methodologies* [6] in software development and operation (*DevOps*) in software companies [7], stressing the need for small cycles of changes in software products. In addition, *DevOps* is going to imply more changes to accommodate the improvements and faults that are learnt during the continuous deployment of product increments [8]. These quick changes require that software systems evolve at the same velocity.

Therefore, the complexity of software systems and technologies is continuously growing and evolving in small change cycles producing erosion in the software at both, architecture and implementation, unless there are mechanisms to fix them at the same velocity. These changes are even more challenging when software architecture is missing [9] when (i) the *Agile adoption* promotes refactoring on the fly instead of constructing software architecture, or (ii) a system was initially conceived as a very simple system and it has grown over time. In both cases, the growth of complexity makes the construction and specification of software architecture mandatory. This evolution also implies a continuous reverse engineering of software systems [10],[11].

These deep changes require some guided reverse engineering reference process to assist, recommend and improve the software architecture design, in the maintenance and software evolution phases, using the architectural design or the source code of software systems as input, and analyzing/modifying its design, coding or even requirements specification. Currently, it is required to have mechanisms that assist in the process of systematically and efficiently changing software systems and their architecture, as quickly as possible. Such mechanisms may encompass use reverse engineering to extract, interpret, and analyze software features and architectural elements, evaluating the impact of two fundamental aspects as *technical debt* [12] and *software sustainability* [13], and *the* iteratively and interactively *reduction or the improvement of both* during the architecture *improvement phase* of the process. This work aims to define this process.

The paper has been structured as follows: section 2 presents the main goals of this research work, section 3 reports a summary of the related work, section 4 summarizes the reverse engineering process defined in this research contribution, and finally the evaluation of this reference process is presented in section 5.

2 GOAL

The main goal of this work is to identify and define a reference process to support the software architecture reconstruction in a systematic and (semi-)automatic way. This main goal address comprises a set of specific goals:

To identify, interpret and classify sources, patterns, models, features, metrics and mathematical models that have an impact on green software and/or technical debt.

To determine the correlation among patterns, software features, architectural elements, green software and technical debt, through iterative and interactive training of system datasets, using algorithms and techniques of Machine Learning.

To define a set of patterns which can be recommend as part of architecture design or implementation, to increase or improve the software sustainability and reduce technical debt, taking into account the previous correlation results.

To construct a framework that allow the adoption of the reference process in (semi)-automatic way to facilitate the development and architecting task in agile and non-architecture scenarios.

3 PREVIOUS WORKS

According to Gall et al. [14] architecture recovery is defined as a process to identify and to extract the software systems in a high level. The authors mention that there are factors that influence the architecture recovery among those that include: meta-information, search patterns, architectural styles, standardized documentation, generic components, human experience and source code. Rasool and Asif [15] propose a framework which together with its phases and some tools such as Dali, Imagix4D and Bauhaus allow the architectural recovery using a bottom-up process. With regard to architecture recovery, Kazman and Carriere [16] indicate that artifacts extracted from a source code model comprise an element collection (e.g., functions, files, variables, objects, etc.), a set of relationships between elements (e.g., “functions calls”, “object having an instance”) and a set of attributes of those elements and their relationships (e.g., “functions that call functions N times”) to represent the system.

When the reverse engineering is done for analyzing and obtaining software architecture “as built”, we are talking about reconstruction. The architecture reconstruction for Ducasse and Pollet [4] is considered as reverse engineering approach which main objective is to analyze and reconstruct architectural views while preserving qualities that allow understanding, improving or verifying software architecture.

Tilley et al. [17] propose some phases for architecture reconstruction. Among the sources of input to carry out the process are those proposed by O’Brien et al. [3]: (i) the software system, (ii) the meta-models, (iii) the strokes obtained when executing the system. In each of the phases techniques such as static analysis, dynamic analysis or others are used, processes such as bottom-up, top-down, hybrid to finally shows the results using different views or architectural representations.

The Software Architecture Reconstruction Method (SAR) [4] and Architecture Reconstruction Method (ARM) [18] proposed by Guo et al. are used to carry out the architectural reconstruction . Both methods use the source code and architectural information for the architecture reconstruction. The activities used in SAR are the following: architecture recovery, architecture discovery, extraction, and mining. Its aim is reconstructing viable architectural views of software applications through the extraction, abstraction, and presentation activities.

Some surveys, SLRs or empirical studies have been conducted in the area of architecture reconstruction and recovery. Ducasse and Pollet [4] define a taxonomy for the architecture reconstruction process. The work of Breivold et al. [19] revises the existing approaches in analyzing and improving software evolution and how to measure quality attributes. Herold et al. [20] focus on studying architecture degradation and consistency checking and post as current challenges for being addressed; Ali et al. [21] also analyze the state of the practice of architecture consistency. To address these challenges, [22] presents a survey that proposes to identify techniques to repair software. Laguna and Crespo [23] address a systematic mapping of product line evolution; specifically, this work studies approaches to the reengineering of legacy systems into SPLs and the refactoring of existing SPLs. Finally, the systematic review of Shahin et al. [19] focuses on the study and classification of the existing visualization techniques of software architectures. However, these SLRs or surveys do not address the study of the complete process of software architecture reconstruction or reverse engineering process.

4 A REVERSE ENGINEERING PROCESS

The main goal of this work is to create a reference process to support the software architecture reconstruction in a systematic and automatic way. Therefore, there was necessary to address still open questions such as: how much systematic, automated, or standardized are current reverse engineering and architecture reconstruction processes, to support current needs? Are these processes able to analyze, interpret and recommend reverse engineering and architectures reconstructions alternatives, depending on the goals of software system? To that end, it was conducted an SLR where the commonalities and complementary activities of existing processes were identified.

The commonalities were identified by functionality, since the name for the same functionality varies depending on the approach. The names of phases and activities were established, taking into consideration the common name used by the different approaches. As result, we defined a reference process and activities with the following phases: *Pre-processing, Extraction, Analysis, Visualization, and Improvement*.

The process use as input the software source code, and optionally, the software architecture and its documentation, since we are focusing on non-architecture scenarios or agile teams were they do not pay special attention to software architecture (See Figure 1). To make feasible the reference process phases and activities some specific aims will be addressed:

In the *Pre-processing* phase, it will be necessary to determine mechanisms to reduce the noise in the source code, to verify the programming language and programming paradigm, to analyze if the classes structure and implementation are correct, and to delete classes and libraries unused are part of this phase. In the next phase (*Extraction*), it will be necessary to determine mechanisms to perform lexical, syntactic and semantic analysis. In the *Analysis* phase, through static analysis or its combination with other techniques, it will determine mechanisms to extract the features or architectural elements of software system. This extraction will use mathematical models and metrics to measure the sustainability and technical debt. The resultant values are associated with a unit of measurement and indicator that allows evaluating each metric (*green* and technical debt). The architectural elements data, software features, mathematical models, metrics classified by type, and resultant values of the *Extraction* and *Analysis* phases, will be stored in the knowledge repository used for this process. It will be necessary to determine the structure of this knowledge repository to be mined in an efficient way during the process adoption. In the *Visualization* phase, it will be necessary to determine mechanisms to visualize the architecture in terms of diagrams, tables, and graphs, with the aim to interpret, analyze and improve the software architecture conformance of the system subject of study. In the *Improvement* phase, the software features, architectural elements and metrics stored in a Knowledge Repository are retrieved. It will be necessary to determine mechanisms to *recommend* software architecture alternatives in terms of technical debt and/or sustainability *improvement*, applying Machine Learning concepts, techniques and algorithms. To that end, it will be necessary to analyze the correlation between patterns, software features, and architectural elements, that allowing increase or improve green software and reduce technical debt.

Each phase and activities in the reference process should be iterative and interactive, where after several *selections* and *interpretations* of *new*, *recommended* and *improved* software architectures reconstructed, a final architecture could be found, and the process will finish. As part of reconstruction process, some tools can be used, in our case, we propose an own prototype called SCAP (Source Code Analysis Prototype) Web Prototype which implements part of reference framework.

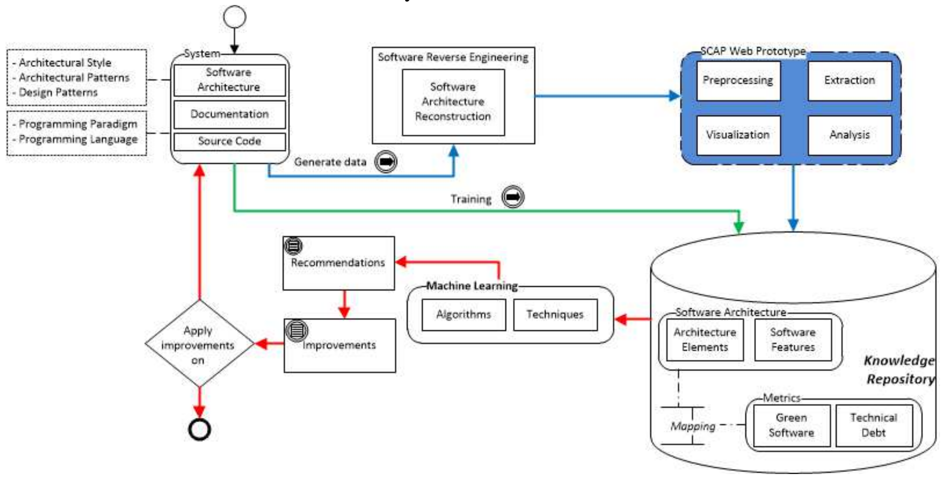

Figure 1 Implementation of the proposed reference framework

5 EVALUATION

To evaluate the results of this reference process, we plan to use software systems with different characteristics as the following:

- The software system could be medium or large size.
- Software systems without architectural design.
- Software systems using only architectural patterns, or design pattern, or architectural styles (individually).
- Software systems using architectural patterns, and design pattern, and architectural styles (and their combination).

We will create a base set of software systems with these different characteristics in order to be used during the evaluation with two different purposes. First, we will use a subset of these for training the dataset using machine learning algorithms in order to generate, define and interpret patterns derived from the architecture reconstruction process, and stored it in the Knowledge Repository to improve or recommend an architecture into a context. Second, we will use a different subset to evaluate the reference process by comparing the resulting architectures and their corresponding initial architectures in terms of software features, metrics and the different kind of systems. In addition, the result obtained of the two purposes, will be compared with some commercial and non-commercial tools, which could implement certain phases of the proposed process. Finally, a long-term evaluation of the reference framework on industrial and academic systems will be necessary to address two issues: (i) to measure the accuracy in the architectural recommendations throughout the time, and (ii) to determine in such a way, the enrichment of knowledge after a considerable amount of process executions will allows one to interpret and compared the generated recommendations and improvements with the initial architecture deployment.

ACKNOWLEDGMENTS

This work is supported by the Spanish fund CROWDSAVING (TIN2016-79726-C2-1-R) and H2020 CPSELabs (644400), and by Universidad Técnica Particular de Loja (*Dpto. de Ciencias de la Computación y Electrónica*).

REFERENCES

[1] E. Tom, A. Aurum, and R. Vidgen, “An exploration of technical debt”, *J. Syst. Softw.*, vol. 86, no. 6, pp. 1498–1516, 2013.

[2] E. J. Chikofsky and J. H. Cross, “Reverse engineering and design recovery: A taxonomy”, *IEEE Softw.*, vol. 7, no. 1, pp. 13–17, 1990.

[3] L. O’Brien, C. Stoermer, and C. Verhoef, “Software architecture reconstruction: Practice needs and current approaches”, 2002.

[4] S. Ducasse and D. Pollet, “Software architecture reconstruction: A process-oriented taxonomy”, *IEEE Trans. Softw. Eng.*, vol. 35, no. 4, pp. 573–591, 2009.

[5] D. Miorandi, S. Sicari, F. De Pellegrini, and I. Chlamtac, “Internet of things: Vision, applications and research challenges”, *Ad Hoc Networks*, vol. 10, no. 7, pp. 1497–1516, 2012.

[6] K. Beck *et al.*, “Manifesto for agile software development”, 2001.

[7] VersionOne, “State of Agiles”, 2017. [Online]. Available: http://stateofagile.versionone.com/.

[8] L. E. Lwakatare, P. Kuvaja, and M. Oivo, “Relationship of DevOps to Agile, Lean and Continuous Deployment”, in *Int. Conf. on Product-Focused Software Process Improvement*, pp. 399–415, 2016.

[9] M. M. Lehman, “Laws of software evolution revisited”, in *Software process technology*, Springer, pp. 108–124, 1996.

[10] R. S. Arnold, *Software Reengineering*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1993.

[11] R. S. Arnold, “Software Restructuring”, *Proc. IEEE*, vol. 77, no. 4, pp. 607–617, 1989.

[12] P. Kruchten, R. L. Nord, and I. Ozkaya, “Technical debt: From metaphor to theory and practice”, *IEEE Software*, vol. 29, no. 6. pp. 18–21, 2012.

[13] S. Naumann, M. Dick, E. Kern, and T. Johann, “The greensoft model: A reference model for green and sustainable software and its engineering”, *Sustain. Comput. Informatics Syst.*, vol. 1, no. 4, pp. 294–304, 2011.

[14] H. Gall, M. Jazayeri, R. Klösch, W. Lugmayr, and G. Trausmuth, “Architecture recovery in ARES”, in *Joint proceedings of the second international software architecture workshop (ISAW-2) and international workshop on multiple perspectives in software development (Viewpoints’ 96) on SIGSOFT’96 workshops*, pp. 111–115, 1996.

[15] G. Rasool and N. Asif, “Software architecture recovery”, *Int. J. Comput. Information, Syst. Sci. Eng.*, vol. 1, no. 3, 2007.

[16] R. Kazman and S. J. Carriere, “View extraction and view fusion in architectural understanding”, in *Software Reuse, Proceedings. Fifth International Conference on*, pp. 290–299, 1998.

[17] S. R. Tilley, S. Paul, and D. B. Smith, “Towards a framework for program understanding”, in *Program Comprehension Proc., 4th Workshop*, pp. 19–28, 1996.

[18] G. Y. Guo, J. M. Atlee, and R. Kazman, “A software architecture reconstruction method”, in *Software Architecture*, Springer, pp. 15–33, 1999.

[19] M. Shahin, P. Liang, and M. A. Babar, “A systematic review of software architecture visualization techniques”, *J. Syst. Softw.*, vol. 94, pp. 161–185, 2014.

[20] S. Herold, M. Blom, and J. Buckley, “Evidence in architecture degradation and consistency checking research: preliminary results from a literature review”, in *Proccedings of the 10th European Conference on Software Architecture Workshops*, p. 20, 2016.

[21] N. Ali, S. Baker, R. O’Crowley, S. Herold, and J. Buckley, “Architecture consistency: State of the practice, challenges and requirements”, *Empir. Softw. Eng.*, vol. 23, no. 1, pp. 224–258, 2018.

[22] L. De Silva and D. Balasubramaniam, “Controlling software architecture erosion: A survey”, *J. Syst. Softw.*, vol. 85, no. 1, pp. 132–151, 2012.

[23] M. A. Laguna and Y. Crespo, “A systematic mapping study on software product line evolution: From legacy system reengineering to product line refactoring”, *Sci.*.