Developing an evolution software architecture framework based on six dimensions

Noureddine Gasmallah*

Department of Computer Science,
Badji Mokhtar University of Annaba,
BP-12, Sidi Amar, Annaba, 23000, Algeria
Email: gasmallahedi@univ-soukahras.dz
*Corresponding author

Abdelkrim Amirat

Department of Maths and Computer Science,
University of Souk-Ahras,
BP 1553, RN 16, Souk-Ahras, 41000, Algeria
Email: a.amirat@univ-soukahras.dz

Mourad Oussalah

Department of Computer Science,
University of Nantes,
2, Rue de la Houssiniére, BP-92208,
Nantes, 44322, CEDEX 3, France
Email: mourad.oussalah@univ-nantes.fr

Hassina Seridi-Bouchelaghemi

Department of Computer Science,
Badji Mokhtar University of Annaba,
BP-12, Sidi Amar, Annaba, 23000, Algeria
Email: seridi@labget.net

Abstract: With the growing number of software architectural evolution methods, the need to develop a framework based on well defined dimensions to analyse approaches is now a prerequisite for practitioners in order to analyse, compare and classify methods within the field of architectural evolution. In this paper, we propose an evolution framework based on six dimensions for analysing, comparing and classifying existing and future evolution methods. The proposed architectural evolution framework adopts Zachman analytic tool (answering what, why, where, who, when and how questions). The framework relies upon identifying dimensions that researchers would take into account while developing a new evolution approach. The set of the proposed dimensions whether combined or individually can serve as a basis to explore further classification paradigms. The proposed framework is supported by an empirical study that involves surveying and analysing 119 research methods related to the area of architectural evolution.

Keywords: software architecture; architectural evolution; evolution framework; evolution dimensions; evolution process; abstraction level; modelling level; components; services; classification; taxonomy.

Reference to this paper should be made as follows: Gasmallah, N., Amirat, A., Oussalah, M. and Seridi-Bouchelaghemi, H. (2019) 'Developing an evolution software architecture framework based on six dimensions', *Int. J. Simulation and Process Modelling*, Vol. 14, No. 4, pp.325–337.

Biographical notes: Noureddine Gasmallah is a Full Teacher of Computer Science at Souk-Ahras University since 2010 and a PhD student at the University of Annaba (Algeria). His research focuses on software architecture evolution.

Abdelkrim Amirat received his PhD in Computer Science in 2007, and Habilitation in 2010. Currently, he is a Full Professor of Computer Science at the University of Souk-Ahras, Algeria. His main research concerns are software architectures and their evolution, modelling and meta-modelling. He worked on several national projects in software engineering. He has published several refereed journal and conference papers in the fields of software architecture, component-based, and object-oriented modelling.

Mourad Oussalah is a Full Professor of Computer Science at the University of Nantes and the Chief of the Software Architecture Modelling Team. His research concerns software architecture, object architecture and their evolution. He worked on several European project (Esprit, Ist, ...). He is (and was) the leader of national project (France Telecom, Bouygues telecom , Aker-Yard-STX, ...). He earned a BS degree in Mathematics in 1983, and Habilitation thesis from the University of Montpellier in 1992.

Hassina Seridi-Bouchelaghemi is a Full Professor at the Computer Science Department of Badji Mokhtar-Annaba University, Algeria and is affiliated to LABGED Laboratory. She has published several papers in international conferences and journals. Her research interests include information systems, recommender systems, e-learning, semantic web, social web, data mining and artificial intelligence.

1 Introduction

Software is constantly subjected to modifications and upgrades during its life cycle to ensure consistency and maintainability as well as to allow software systems to embrace new requirements and adapt to new strategic directions (Fitzgerald and Stol, 2016). Software evolution has become a major concern for stakeholders involved in the process of designing and modelling computer systems, which have become more complex to maintain and extend. Since the introduction of Lehman laws (Lehman, 2016) for formalising the rules related to software evolution, multiple approaches and tools have been proposed within the research literature to address this topic. Considerable body of research studies (Mittal and Vetter, 2016; Stol et al., 2016) have affirmed that software evolution should be considered at the architectural levels of the design process. The software architectural level provides an abstraction view with better structured and formalised mechanism to develop software systems with the potential to control and manage anticipated future changes (Shaw et al., 1995; Landi et al., 2017). As a result, a wide range of research studies have been found describing various methods, techniques and tools addressing software architectural evolution by using well-known techniques such as re-factoring, transformation, migration, adaptation, dynamic configuration and so forth (Behnamghader et al., 2017; Zimmermann, 2015; Monperrus, 2018; Villegas et al., 2017; Albassam et al., 2017; Gummalla and Rao, 2018; Cavalcante et al., 2015).

The area of software architecture evolution has received unprecedented interest during the last two decades where numerous research studies have been published describing various methods and frameworks. In return, few studies have been devoted to present the classification for software architecture evolution (Breivold et al., 2012). The main stream of most studies deals with this topic as a systematic review for architecture evolution as knowledge re-usability, quality evaluation, modelling techniques, etc. (Aleti et al., 2014; Ahmad et al., 2014). It becomes an essential necessity to establish a well-defined framework and taxonomy based on a common vocabulary of a wide range of approaches with the aim of classifying existing studies. An evolution framework maps a taxonomy for representing, analysing, comparing and classifying the interior design aspect within software architecture evolution and would help practitioners in the decision-making process. Buckley et al. (2005) have proposed a taxonomy of software changes that characterises the mechanisms of change of the software and their influencing factors without taking into consideration the software architectural view. In addition, Williams and Carver (2010) have described a framework to address software changes by identifying the major useful causes and effects to illustrate the potential impact of the change. In the same way to previous studies, the architectural view of the software was not addressed within their study.

In this paper, we describe a software architectural evolution framework based on an analytic tool for analysing, comparing and classifying existing and future evolution methods. The framework is inspired from the work of Buckley et al. (2005) for software change in addition to the work of Zachman (1987) for the architecture of information systems. The proposed framework is derived from the answers of the main questions proposed within the tool and generates six major dimensions explaining the evolution of the architecture: the object of evolution (what), the type of evolution (why), the planned time for evolving the architecture (when), the key stakeholders involved within the evolution process (who) and finally the reasoning or adopted mechanism used for the evolution (how). The proposed framework is devised to propose a newly set of metrics for the arena of software architecture evolution. Such dimensions should represent a broad spectrum of architectural evolution methods, facilitate comparisons and classify them. In addition, combining a set of these dimensions, the framework provides a classification, which is devised to regroup and categorise architectural evolution approaches. The set of the proposed dimensions whether combined or individually can serve as a basis to explore further classification paradigms. This will be a key insight for identifying classes as well as dimensions that require deep investigations in one part and would serve as a meta classification for the existing taxonomies. The proposed framework is supported by an empirical study, which has involved surveying and analysing 119 research methods related to area of architectural evolution from the literature.

The remainder of the paper is organised as follows: Section 2 is devoted to the related work. Section 3 details the proposed dimensions defined for analysing architectural evolution methods, while Section 4 defines and explains a method for classifying them. Section 5 illustrates the application of the suggested framework on a set of well-known frameworks in the field of software evolution followed by the presentation of results and a discussion based on the proposed dimensions within Section 6.

2 Related work

Although the arena of software architectural evolution is relatively mature (Ebert et al., 2016), there are a few studies having shown interest to survey and classify existing methods. Williams and Carver (2010) focused on software change characterisation mechanism, which allows developers to characterise the effects of a change by considering a set of criteria. The authors have provided a software change characterisation schema (SCCS) based on a systematic literature review of 82 papers entailed in providing changes to software architecture. The study is concerned mainly with the changes of software architecture while metrics are identified on the base of previous methods involved in software change. Another major study is proposed by Shahbazian et al. (2018). The authors have addressed architectural evolution towards the use of the code repositories of open source software systems. The aim of their study was to build a predictive model to examine the causes and change frequency of architectural design decisions. Furthermore, Behnamghader et al. (2017) have introduced the ARCADE framework for analysing recovered and updated architectures across several versions. The empirical study was conducted on hundred versions of 23 open-source systems but it was limited to calculate the frequency of architectural changes for software systems. Both studies (Behnamghader et al., 2017; Shahbazian et al., 2018) consider defining their metrics from a set of keywords expressed explicitly within the evaluated research papers.

For the architectural context, Ahmad et al. (2014) developed the PatEvol framework that integrates knowledge acquisition and knowledge execution specially for allowing the reusability of evolution architecture knowledge. PatEvol offers a continuous acquisition of knowledge by analysing architecture evolution histories referred to as architecture evolution mining. Afterwards, this knowledge can be executed to perform an architecture evolution referred to as application of architecture evolution knowledge. Moreover, Weinreicha et al. have analysed the support for central architecture knowledge management activities of existing approaches. The main drawback of such studies (Ahmad et al., 2014; Weinreicha and Grohera, 2016) is the focus on the technical aspect, which limits their use only for specialised stakeholders. Buckley et al. (2005) have provided a taxonomy that positions concrete tools, formalisms and methods within the domain of software evolution. Their proposed taxonomy is based on a set of key questions to guide the identification of factors impacting the use of tools for evolution by offering to practitioners a framework to ease comparison between such tools. However, the main focus is absolutely on the technical view of the software evolution without any preoccupation on why this evolution is needed. In addition, the study by Buckley does not cope with the architectural viewpoint of software systems. In a different study, Zachman (1987) has proposed an analytic tool for enterprise architecture. The Zachman framework is a normalised schema as it tackles one fact in one place. Moreover, its well-defined semantic construction shapes the structure independently of the system implementation. This is a key solution for analysing systems and providing a macro evaluation of such systems. In fact, the framework does not take into account the implementation processes nor the tools that are employed to achieve the evolution of the architecture of the software (Lapalme et al., 2016; Noran, 2003). Despite the fact that the study by Zachman is considered as a satisfactory analytical tool, there is no study conducted to explore the evolution at the architectural level with a set of well-defined taxonomies and metrics (Ni and Li, 2017; van de Weteringet and Bos, 2016; Bondar et al., 2017).

3 The proposed framework for architecture evolution

The key aim of the framework is to identify a common and explicit benchmark on which methods for architectural evolution are supposed to resolve. This will highlight the differences between operating evolution, distinguishing between the mechanisms by which architecture evolves from initial to final state, and the specifications according to which approaches can be classified. This section describes the review, how the proposed framework is developed, and determines inter-dimensions relationship implied by their aggregation.

3.1 Framework dimensions

As the assessment and classification of evolution methods should be based on a set of well-defined dimensions, a set of metrics are proposed to provide answers about the six questions introduced earlier. By considering these answers, six dimensions are distinctively provided and constitute the basis of the proposed conceptual framework. It is worth noting that in order to ensure a good understanding of such dimensions, an illustrative example concerning electronic circuits of the arithmetic logic unit (ALU) is developed throughout the definition of the different dimensions.

3.1.1 Object of evolution dimension: what

The object of evolution is the subject on which the evolution is operated. The changes are applied to specific components of the software system that are well predefined at the very earliest design stages, and that is operated in compliance with suitable decisions and stakeholders interventions. Object of evolution answers the what question and comprises the following:

- Artefact: is an abstraction of any element belonging to the architectural structure. It may include complex and abstract description as software architecture, a component, a service and so forth. It can be simple and even concrete description as program codes, e.g., the ALU performs arithmetic operations as addition and subtraction or logical operations as AND and OR. This means that an adder, AND and OR gates are all considered as artefacts.
- Process: ISO (ISO9000, 2017) defines a process as a set of interrelated or interacting activities that transforms inputs into outputs. It is an abstraction for all isolated or aggregate operations and methods employed to carry out evolution; therefore the process also evolves and presents suitable means for anchoring the benefits pertaining to cost optimisation and quality promotion (Pigoski, 1996; Lai et al., 2018), e.g., arithmetic functions of the ALU (+, –, *, /) are considered as a process of evolution. In the same way, fetch process, instruction decoder and task management are all considered as processes for computer CPU.

Succinctly, the evolution can be determined as oriented towards *artefact* or *process* based on the value of the *object* as a dimension.

3.1.2 Type of evolution dimension: why

It is one of the most commonly-used metrics when addressing classification issues. This typology is a reasonable choice in the prospect of providing answers about the main motivation for architectural evolution (why?). Indeed, this is inspired from maintenance typology according to Abran et al. (2001), Swanson (1976) and Alenezi and Zarour (2016). This dimension encloses the two following main-subdimensions:

- Main-forms of resolution: architects can use several strategies for the evolution problem-solving according to different forms. From the literature, the following two major studies discussed the various forms in the area of evolution.

  1. Chaki et al. (2009) grouped the different methods into two main strategies: *open-evolution* means a deduction, from an initial architecture, of a new architecture reflecting a system solution in which a set of invariants and constraints are respected. The final architecture is not a priori known and is inferred by successively applying several operations on a starting architecture. *Close-evolution*, on the other hand, refers to the evolution activity, which uses induction resolution to find the best permissible sequence of operations to be applied to achieve the desired result. In this case, the target architecture is a priori known.
  2. Front-Conte et al. (1999) proposed two other forms: *break*, it means that interventions are applied directly to the initial architecture without having the ability to go back on the trace of the evolution. The second is evolution with *seamless* denotes an evolution with trace whereby the architecture keeps a trace of its initial properties and operations performed before each evolving process.

  For the classification of evolution approaches through the form aspect, further sub-categorisation is introduced based on previous studies. The main-forms sub-dimension can be set either to *close* or *open* and *break* or *seamless*.

- Main-categories: instead of categories based on the architect's intentions, main-categories sub-dimension is responding to the objective evidence of architect's activities identifiable from observations and artefacts before and after comparison of the software architecture. According to Chapin et al. (2001), objectives are derived from five specific sub-types (categories) of evolution: enhancive, corrective, reductive, adaptive and perfective. In this research paper, the three commonly used subtypes (corrective, perfective and adaptive) are solely considered for the reason of simplicity.

Table 1 The inter-relationships for the types of evolution

		Type of evolution	
		Curative	*Anticipative*
Main-form	Open	○	●
	Close	●	⊙
	Break	⊙	⊙
	Seamless	○	○
Main-category	Corrective	●	○
	Adaptive	○	●
	Perfective	○	⊙

Note: Legend for the relationship levels: ● – higher; ⊙ – medium; ○ – weak.

The proposed structuring for deducing the assumed evolution type for a particular method is shown in Table 1 such that the categorisation is based hybridly on categories and forms of resolution. Herein, the proposed type of evolution is structured according to two main forms (curative or anticipative), and is explained as follows:

- Curative: can be applied in run or load time either when new requirements arise unpredictably or are poorly defined or even unspecified during the life cycle. Meanwhile the environment helps to correct and enhance them by integrating the desired changes, whether permanently or temporarily;
- Anticipative: can be applied when evolution requirements are taken into account during the analysis phase and are specified during the design-time or load-time.

Table 1 shows that the *closed* form and the *corrective* category are highly related to the *curative* type. Similarly, the *adaptive* category in *open* form is more correlated with the *anticipative* type. The weak link characterises the *anticipative* type when evolution deals with *corrective* category or *seamless* form.

3.1.3 Hierarchical level dimension: where

Software systems undergo several modelling phases at different levels before the final delivery. Two types of hierarchical levels (modelling and abstraction levels) are commonly identified within the software engineering literature as depicted in Figure 1.

- Modelling levels (M_0 to M_2): software systems must be mapped over several levels, from lower-level constructs (code) to higher-level constructs, to ensure convergence (Shaw et al., 1995). Modelling level refers to one of the four levels (from M_0 to M_3) as defined by the OMG (Bézivin and Briot, 2004). It is worth mentioning that, with the exception of the meta-meta level (M_3), evolution can be performed in one or several modelling levels by reference to M_0, M_1 and M_2 levels. Obviously, the switch between these levels requires the use of different set of concepts and tools related to the hosting level. For example, addressing components or classes or instances concepts reflect different modelling levels. To give a concrete illustration relating to the ALU example, as previously discussed: *implementing code related to an adder circuit representation presents a lower modelling-level (specified here by the M_0 level). Whereas the associated logic-design diagram can be seen at the M_1 modelling-level.*
- Abstraction levels (a_0 to a_n): Experts often use abstraction as an approach to address complex problems. This is still very often used for solving problems related to software modelling. Thereby, a solution is performed through a series of model description at a number of abstraction levels referred within Figure 1 as a_0, a_1,..., a_n. By defining a suitable level of abstraction for a given application phase, the amount of semantic information being considered by relegating irrelevant details to a lower level of abstraction is limited (Oussalah, 2014). In practice, the flowcharts, algorithms and pseudo-codes are often used as a starting point for creating a computer program. In fact, these presentations correspond to different abstraction-levels embedded at the M_0 modelling-level. For the ALU example: *an 8-bit adder circuit is represented as a set of interconnected 1-bit adder. Refining this representation by including the logic gates needed for implementing the 1-bit adder will introduce more details and then indicates another lower abstraction-level.*

It is therefore noteworthy that the relationship between the modelling and abstraction levels emphasises the non-isomorphism between the two concepts. It indicates that a modelling level can be composed of several abstraction levels. Moreover, each object of evolution is subject to change at several hierarchical levels.

3.1.4 Time of evolution: when

This dimension asserts the importance of the temporal aspect in terms of architecture modifications (when?). This indicates that evolution mechanisms occur during different times of software development phases. From an architectural viewpoint, time of evolution can be linked to one or several hierarchical levels. However, the system environment has a profound impact on the time and tools to be used during the software evolution. For example, evolving architecture at run-time, must ensure the reconfiguration and system integrity over all software levels (Oreizy and Taylor, 1998). For the most dominant metrics used in software architecture, mainstream research studies focus on design-time and run-time evolution (Ahmad et al., 2014) instead of static-time, load-time (build-time or link-time), initiation-time and run-time (execution-time) adopted in software change (Buckley et al., 2005; Bass et al., 2003; Nigro et al., 2018). In respect of architectural viewpoint, the time of evolution embraces one of these metrics:

- Design-time: predicting evolution at the earlier stages of design allows improving and extending the system architecture. Model Driven Architecture is a good example of methods addressing evolution at design-time, especially co-evolution approaches.
- Run-time: the evolution can occur while the system is running (Oreizy and Taylor, 1998). So far, evolving at run-time is considered as a major topic to address architecture adaptation (Mens et al., 2005). This metric encompasses:
  - a compile-time where the system source code entails necessarily a recompilation process for generating the new system after each modification
  - b load-time designates the elapsed time between stopping the software and its re-execution for the change to take effect (Kniesel et al., 2001)
  - c dynamic-time viewed as a promising sub-dimension (Mens et al., 2005), in which objects are upgraded while the system is running in light of adapting or reconfiguring software.

E.g., self-reconfiguration software allows modification at run-time of the appropriate behaviour in response to environment changes. So, architectural elements can be dynamically added into various software systems without entailing system restart. This is considered as dynamic-time evolution.

Figure 1 Modelling levels and abstraction-levels (where) (see online version for colours)

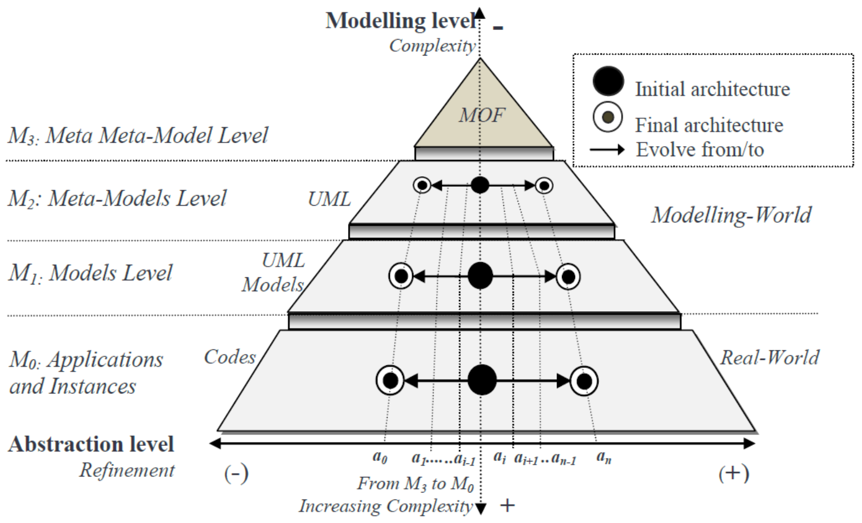
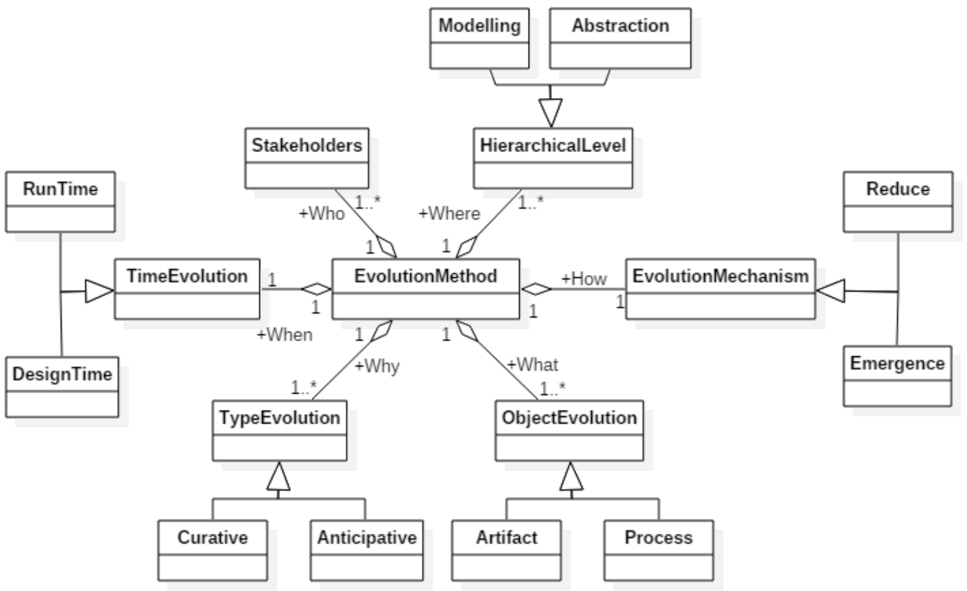


Figure 2 Architectural evolution framework based on six dimensions



3.1.5 Stakeholders: who

Stakeholders contribute in architecture enhancement at multiple levels and roles regarding the responsibility they assume and the range of challenges they face (Terho et al., 2017). For instance, this dimension (who?) covers the stakeholders who operate evolution on the software system itself by referring to the development team including architects, designers, developers, and so forth (Mens et al., 2005), as well as self-automation to deal with automated approaches. This dimension mainly depends on enlarging the expertise and knowledge of the stakeholders in decision-making, in particular architects, which can enable them to assess possible future requirements by integrating any anticipated changes at the different life cycles and hierarchical levels (Jazayeri, 2005; Clements et al., 2003). Therefore, they need well-defined requirements, accurate feedback from all those involved in this software system in order to make the chaos go away (Rising and Janoff, 2000), e.g., software programmers implement programs according to the new specifications.

3.1.6 Operating mechanism of evolution: how

The operating mechanism of evolution (OME) is introduced to refer to the general behaviour and employed procedure by taking into account solely the described hierarchical levels. The OME would therefore allow us to reveal how the evolution impacts hierarchical levels throughout the process of change. Mainly, evolution solving approaches commonly adhere to one of the two main following methods:

- Reduce: for the reductionist evolution, the process gets through a predefined evolution path; until the solution is satisfied (evolved model). Brooks (1989) defines the 'reductionist' approach as 'classical' approach to problem solving to which the overall resolution task is decomposed into subtasks, e.g., instances corresponding to the class of an adder within a model of electronic circuit are named functions/procedures declared at the lower level of design. The 4008 is a 4-bit binary full adder instance with two 4-bit data inputs.

- Emergence: On the contrary during an 'emergentist' evolution approach, the process builds the path to a solution. The emergence exposes a passage between the activity of 'micro-level' and that of 'macro-level', e.g., by aggregating multiple adders (1-bit adder), larger adders can be created (32-bit adder).

3.2 *Dimensions relationship*

Figure 2 maps the six dimensions upon which a model for architectural evolution can be made. The framework uses composition relations where all dimensions are required for evaluating the framework. Indeed, all dimensions can be evaluated by answering the appropriate questions whether implicitly or explicitly. The composition between the evolution method on the one side, and the time dimension (respectively the evolution mechanism) on the other side reveals that a given method solves the evolution issue by focusing on run time or design time (respectively on reduce or emergence reasoning). However, the 1,* multiplicity on a given dimension implies the need to have one value.

Two types of relationships can be yielded via comparing a pair of dimensions including the dimension to itself. This is better elaborated in Table 2 where the first relationship is an intra-dimension (cells with a grey background) that clarifies whether metrics, which belong to the same dimension, are mutually exclusive ($\perp$) to each other. In such case, the dimension is quantified with the value $v = 1$ when the approach deals explicitly with the considered metric. In the other case, the metrics are combined ($\bowtie$) and therefore the selected metrics are calculated relatively to the sum of all metrics. For example, OME of an evolution approach can be set exclusively to *Reduce* or *Emergence* meanwhile the hierarchical-levels affected by the evolution can be: *abstract*, *modelling* or both. Whereas, the type dimension is combined (50% for each sub-dimension: main category and main form) and values in each sub-dimension are equally distributed.

The second type of relationship is an inter-dimension that identifies the possible dependencies between metrics belonging to different dimensions. Cells with ($\exists$) symbol in Table 2 show the existence of a weak dependency between time, hierarchical levels, stakeholders and type of evolution due to the potential of overlap between certain values such as: run-time, lower levels of abstraction, correction type and programmers. Based on the analysed relationships, the metrics are observed to have loose dependencies across the proposed dimensions, denoted with $\exists$ and $\varnothing$ symbols.

Table 2 Intra and inter-relationships of the proposed dimensions

	Dimensions	*(1)*	*(2)*	*(3)*	*(4)*	*(5)*	*(6)*
1	What? Object	$\perp$					
2	Why? Type	$\varnothing$	$\bowtie$				
3	Where? Levels	$\varnothing$	$\exists$	$\bowtie$			
4	When? Time	$\varnothing$	$\exists$	$\exists$	$\perp$		
5	Who? Stakeh.	$\varnothing$	$\varnothing$	$\exists$	$\exists$	$\perp$	
6	How? O.M.E	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\varnothing$	$\perp$

Notes: $\perp$ – exclusive metrics; $\varnothing$ – no dependency between dimensions; $\exists$ – existance of a weak dependency between metrics; $\bowtie$ – combined metrics.

4 Classification of evolving software architectures

Classification provides a common vocabulary for comparison and distinction of the different research studies. The proposed classification comprises six classes based on the operating mechanism of evolution (OME dimension) reasoning rather than addressing classification by thematic as in Breivold et al. (2012) and Ahmad et al. (2014). De facto, it would be better for a taxonomy class to represent approaches that share a common method of reasoning according to an architectural viewpoint.

Herein, a taxonomic class presents a common method of classifying based on an architectural view-point as first class citizen and therefore focuses on the evolution mechanism dimension (how: reduce or emergence) across the architecture hierarchical level dimension (where: modelling and abstraction levels). For each metric of OME dimension, the evolved architecture goes through one of three cases:

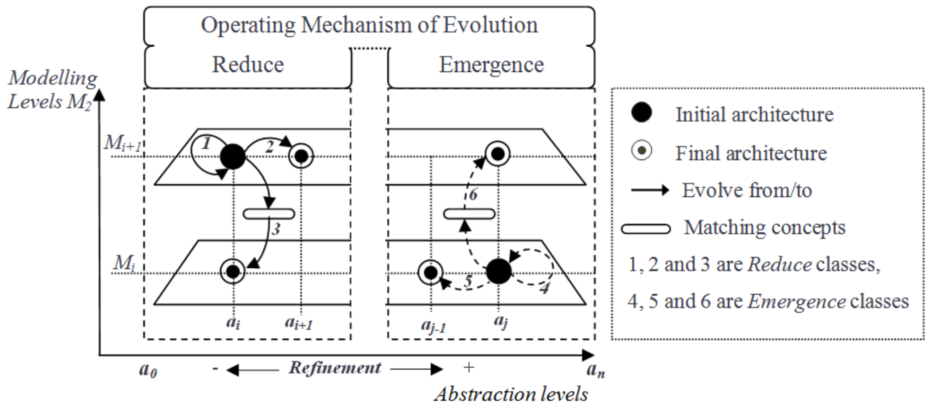
1 through modification of properties of certain architectural elements and/or functionalities (referred as 'intra-abstraction level')

2 through the derivation of several concepts within the same level of modelling (referred as 'inter-abstraction level')

3 through various concepts being handled at different modelling levels (referred as'inter-modelling level').

These paths imply the need for six classes for classifying approaches in accordance with the above assumptions as shown in Figure 3. Both dimensions would result in producing six different classes. As all dimensions are important and play a pivotal role for the classification, considering more dimensions would increase the number of classes substantially and without doubt yield deeper and finer classification. However, due to the limited scope of this study and the number of parameters, only six classes are used to showcase the framework meanwhile other metrics can be of course considered to reach a deep-level classification. The produced classes are explained as follows:

Figure 3 Taxonomy based on hierarchical-levels (where) and operating mechanism of evolution (how)



- Class 1 – intra-abstraction level reduce-oriented evolution: the evolution process consists of evolving one or more elements of an initial architecture without having an impact on its modelling level or its abstraction level, e.g., modification of one or more component properties of an architecture.
- Class 2 – inter-abstraction level reduce-oriented evolution: the impact of evolution on the architectural elements is improved over several abstraction levels while maintaining the same modelling level of the initial architecture, e.g., evolution of class impacts several sub-classes.
- Class 3 – inter-modelling level reduce-oriented evolution: the architectural elements are enhanced on a downward modelling path through several architectural modelling levels, e.g., evolution of class impacts instances.
- Class 4 – intra-abstraction level emergence-oriented evolution: the evolved architecture is enriched with new architectural elements by applying operations, which do not have an impact on levels of abstraction and modelling other than those of the initial architecture, e.g., categorisation of classes by creating a super-class.
- Class 5 – inter-abstraction level emergence-oriented evolution: using the same modelling level as the initial architecture, the emergence of new architectural elements is manifested through the involvement of several levels of abstractions for reason of simplicity, e.g., aggregation of classes in one class.
- Class 6 – inter-modelling level emergence-oriented evolution: the evolution process implicates several modelling levels on an upward modelling path in order to provide the final architecture with new architectural elements, e.g., creation of new classes (at M_1 level) to deal with differences on multiple instances belonging to M_0 modelling-level.

5 Experimental studies

The experiment study conducted here is based on studying existing research publications in order to devise metrics and dimensions to construct an analytical framework. Therefore, the input for the proposed framework would be existing or even future evolution research methods and studies. Meanwhile, the output would be the analysis and classification, which are based on the constructed taxonomy.

5.1 *Illustration example*

In order to explain how architectural evolution is evaluated, two case studies of architectural evolution are analysed and compared to show both the applicability of the proposal and the classification procedure. The selected approaches are only cited as an example for evolution approaches among many other relevant approaches. The first study by Oussalah et al. (2006) presents a software architecture evolution model (SAEV). It aims to describe and manage the evolution through the architectural elements of a given software architecture. SAEV considers software architecture elements (component, connector, interface and configuration) as first class entities. Managing these elements is conceived independently of any architecture language level by considering the different levels of modelling (meta-level, architectural level and application level). The second by Barnes et al. (2003) is an automatic approach provided to assist architects by planning alternative evolution paths for evolving software architecture. An evolution path is expressed in terms of intermediate architecture generated from the initial state. If the architect's goal is clear, this approach aims to assist architects to find the optimal path that meets the evolution requirements. The evolution path acts explicitly on software architecture and applies evolution from a higher to lower level to reach the desired architecture. This approach helps architects to find the optimal path for the evolved architecture. In order to apply the conceptual framework for the two approaches, all the discussed dimensions are quantified using three values one, 0.5 and zero, which reflect respectively explicit, implicit and not mentioned. Thus, this quantification is essential to help architects to assign an approach to a specific class from those suggested in Section 4. Clearly, the two selected approaches present different values for the six proposed dimensions as it is shown in Table 3.

Table 3 Comparison of two evolution approaches using the framework

Dimensions	*Papers*	*Oussalah et al. (2006)* Val	%	*Barnes et al. (2003)* Val	%
Object – what	Artefact	1	100	1	100
	Process	0		0	
Levels – where	Modelling	1	75	0	25
	Abstraction	0.5		0.5	
Stakeh. – who	Architects	1	100	1	100
Time – when	Run	0	100	0	100
	Design	1		1	
OME – how	Reduce	1	100	0.5	50
	Emergence	0		0	
Type – why	Main-forms				
	Open	1	18.75	0	18.75
	Close	0		1	
	Break	0.5		0	
	Seam	0		0.5	
	Main-categories				
	Correction	1	16.67	0.5	16.67
	Adaptive	0		0.5	
	Perfective	0		0	

Note: Main-form and main-category subdimensions are calculated as $\frac{\sum \text{metric values}/\text{total of metrics}}{2}$, i.e., main-form = $(1.5/4)/2$.

For the *hierarchical-level* dimension for example, the first approach has explicitly expressed evolution at different modelling levels and implicitly shows the consideration of the abstraction levels. This leads to consider the value of 1.5 (75%) from the whole expressiveness of this dimension. Whereas, the second approach expresses implicitly the abstraction level having as percentage 25%. These percentages serve as evaluation and comparison for determining the approach focus regarding the proposed dimensions. In similar fashion, the other dimensions are analysed and compared accordingly.

It is worthwhile noting the case of the type dimension, which is a combined metric, i.e., all possible values constitute the whole unit 100%. In this case, each subdimension is halved and the sum of the obtained options are calculated accordingly.

The calculations of where and how dimensions reveal that SAEV approach presents reduce operating mechanism of evolution at different hierarchical levels and hence the approach is classified within modelling level reduce-oriented evolution (class 3). Whereas, Barnes' method describes evolution by reducing at abstraction levels, and then is associated with the abstraction level reduce-oriented evolution (class 2) of the proposed classification. In summary, the first approach focuses on modelling levels of the architecture whereas the second is based on abstraction levels mainly used for reducing evolution complexity. It is noteworthy that the two approaches show identical values of the object, type, stakeholder, time of evolution and both support the 'reduce' operating mechanism of evolution.

5.2 Applying the framework

Three prominent existing studies have been selected and surveyed to better evaluate the conceptual framework using the proposed metrics on their set of surveyed papers as follows: The first considered survey study that involves 32 research papers was conducted by Ahmad et al. (2014). This study focuses on approaches wherein changes impact the architectural level when analysing and improving software evolvability. The investigation of the existing methods or techniques, either for systematic application or for empirical acquisition of architectural knowledge, categorises evolution reuse knowledge into six broad themes:

1 evolution styles
2 change patterns
3 adaptation strategies and policies
4 pattern discovery
5 architecture configuration analysis
6 evolution and maintenance prediction.

The proposed thematic classification focuses on both times of evolution and type of evolution for reuse knowledge.

In the second study by Breivold et al. (2012) five main categories of themes are identified based substantially on research topics through an investigation of 82 research papers:

1 techniques supporting quality consideration during software architecture design
2 architectural quality evaluation
3 economic valuation
4 architectural knowledge management
5 modelling techniques.

A set of specific characteristics is provided with a view to refine each category to subcategories reflecting a common specification in terms of research focus, research concepts and context. The third study by Chaki et al. (2009) recommends three classes of common type for architectural evolution:

1 Maintenance focused evolution – these are works concerned with the use of correction decisions and

architectural modifiability to address architects' concerns. These works often require intervention at the lowest modelling level.

2 Open evolution – refers to all software architectural evolution works whose final architecture is not known a priori. These approaches infer, from an initial architecture, one or more solutions in respect of a context known beforehand.

3 Closed evolution – categorises works in which the target architecture of the model is known before proceeding to the evolution of the initial architecture. It is a question of finding a sequence of operations that may guide an initial architecture to a desired one.

6 Results and discussion

The first step was to devise a set of projections of the presented studies for the proposed conceptual benchmarks and afterwards for the taxonomy. These projections are aimed to reveal the coverage of each of the benchmarks that appear most frequently across the range of architectural evolution studies. Table 4 summarises the quantification using weighted results of the six dimensions for the 119 papers being reviewed within this study. The percentage of each dimension is calculated by dividing the sum of expressed studies by the total number of investigated studies.

Table 4 Coverage percentages of the proposed dimensions

Dimensions	*Studies (119)*	*32 Ahmad et al. (2014)*	*82 Breivold et al. (2012)*	*5 Chaki et al. (2009)*	*W.A %*
Object *(what)*	Artefact	66.67	74.60	20	70.17
	Process	33.33	25.40	80	29.83
Levels *(where)*	Modelling	48.38	28.58	60.00	35.22
	Abstract	51.62	71.42	40.00	64.78
Stakeh. *(who)*	Architect	100	100	100	100
Time *(when)*	Run	34.62	24.44	22.22	27.08
	Design	65.38	75.56	77.78	72.92
OME *(how)*	Reduce	96.78	94.29	100	95.20
	Emerge.	3.22	5.71	0	4.80
Type *(why)*	M.-forms	10.59	41.86	100	35.89
	M.-categ.	89.41	58.14	0	64.11

Note: W.A – weighted average = $\frac{\sum percentage_i \times n_i}{\sum n_i}$ where n_i is the number of the surveyed papers.

Detailed illustration for the provided results in Table 4 relating to the *type dimension* (including *main-forms* and *categories*) are more detailed in Table 5. The results shown in both tables reveal the just proportion (or disproportion) between weights granted for each proposed dimension. However, Table 6 indicates implicit determination of the coverage using the proposed classification. Herein, the proportion and disproportion characteristics are linked to the determination of the trend of two approaches according to a given dimension. The two approaches are proportional (reciprocally disproportional) when the tendency of the two approaches is identical (reciprocally inverted) for a given dimension, e.g., Ahmad et al. (2014) and Breivold et al. (2012) are proportional to the object of evolution dimension since they have an artefact orientation. In contrast, both are disproportionate to Chaki et al. (2009), which is process oriented.

Table 5 Details of form and category percentages

Main	*Studies (119)*	*Ahmad et al. (2014) 32*	*Breivold et al. (2012) 82*	*Chaki et al. (2009) 5*
Form	Open-break	04.71	36.05	80.00
	Open-seam	03.53	04.65	00.00
	Close-break	02.35	01.16	00.00
	Close-seam	00.00	00.00	20.00
Categ.	Corrective	37.65	24.42	-
	Perfective	28.24	26.74	-
	Adaptive	23.52	06.98	-

Table 6 Summary of taxonomy-based OME percentages

Reduce OME				
Studies	*Nbr*	*Class 1*	*Class 2*	*Class 3*
Ahmad et al. (2014)	32	32.26	16.13	48.39
Breivold et al. (2012)	82	48.56	17.15	28.58
Chaki et al. (2009)	5	20	20	60
Weighted average		42.98	17.00	35.23
Reduce average			31.73%	
Emergence OME				
Studies	*Nbr*	*Class 4*	*Class 5*	*Class 6*
Ahmad et al. (2014)	32	3.23	0	0
Breivold et al. (2012)	82	5.71	0	0
Chaki et al. (2009)	5	0	0	0
Weighted average		4.80	0	0
Emergence average			1.60 %	

Significantly, Table 4 shows that over three-quarters of the selected studies have explicitly fostered modelling levels for all works. This clearly indicates the relevance of these studies to the architecture evolution topic. Furthermore, percentages show that a wide range of studies are committed to artefact as an *object of evolution*, except for Chaki et al. (2009) that focuses on process evolution studies. This is due to the fact that both existing evolution support formalism [*SAEV* (Oussalah et al., 2006), *Query/View/Transformation-based* (Mens and van Gorp, 2006)] and architecture description languages (e.g., ADL, UML, xADL) for evolution models are all artefact oriented. It is worth noting that styles and patterns are actually more suitable for evolving architecture elements (artefacts) but contribute little to address the evolution issue of the styles themselves (Hassan and Oussalah, 2016). In addition, the modelling level has attracted major interest of evolution community for the reason that it overlaps with the abstraction level concept. In the same way, an overwhelming percentage deal with reducing evolution from higher to lower level, i.e., top-down hierarchy modelling fashion. In our view, this preponderance is mostly due to a deductive reasoning influence linked to the top-down method (i.e., reduce OME), as shown in Table 6. However, this reductionism may deprive software architecture systems to introduce new solution opportunities according to the current and/or future environment assumptions. Regarding the temporal dimension, values are systematically in favour of the design-time, which in our opinion reflects the interest to the anticipation activity to deal with evolution. The percentages show that the main categories sub-dimension attracted an explicit interest from all the studies apart from Chaki et al. (2009). It is noted that all these studies specify the importance of stakeholders' requirements when dealing with evolution. Furthermore, Table 6 indicates implicit determination of the coverage using the proposed classification. Clearly, class 1 and class 3 are the ones where most of evolution issues are focused on the most, whereas class 5 and class 6 are the least considered of all the classifications. This classification underlines that the emergence mechanisms for evolution are still underexploited and promise the best ground for architectural evolution research.

7 Conclusions

In this paper, a conceptual framework is presented for modelling the classification of software architecture evolution approaches. This model is based on evaluating six proposed dimensions. The proposal is part of a wider strategy to evaluate the tendency of existing research within software architecture evolution according to well-defined dimensions. Thus, the proposed model provides architects with the capacity for understanding and analysing the impact of a desired change.

Considering the impact of evolution process through the hierarchical levels, an evolution classification of six different classes is devised. In respect of architectural viewpoint, it would be better for a taxonomy class to represent approaches that share a common method of reasoning. This would help to sketch the interior design of the evolution solution space. The taxonomy is demonstrated to be expressive (i.e., capable of representing a wide spectrum of architectural evolution methods) and effective (i.e., facilitates similarities and comparisons of architecture involved in the software evolution).

We have conducted an empirical study by surveying and analysing 119 research methods related to evolution projected against three existing classification paradigms from the literature. The investigation of the coverage in software architecture evolution has drawn a number of conclusions on opportunities, strength and weakness of existing approaches. The obtained results confirm the consistency of the proposed framework in comparison with other existing surveyed studies based on the selected metrics. The proposed conceptual framework enjoys the merits of comparing different architectural evolution approaches in addition to the ease of classification based on the proposed dimensions.

However and over the many implications of our findings, the proposed evolution framework is limited on a conceptual aspect of evolution. Furthermore, we will extend the existing framework to include the quality aspect that is deemed to be important for analysing the evolution impact. As future perspective, we plan to implement a tool that can automatically evaluate any given or new approach based on the set of the proposed dimensions.

References

Abran, A., Bourque, P., Dupuis, R. and Moore, J.W. (2001) *Guide to the Software Engineering Body of Knowledge-SWEBOK*, IEEE Computer Society Press, USA, ISBN: 978-0-7695-5166-1. .

Ahmad, A., Jamshidi, P. and Pahl, C. (2014) 'Classification and comparison of architecture evolution reuse knowledge – a systematic review', *Journal of Software: Evolution and Process*, Vol. 26, No. 7, pp.654–691.

Albassam, E., Gomaa, H. and Menascé, D.A. (2016) 'Model-based recovery connectors for self-adaptation and self-healing', *11th International Joint Conference in Software Technologies ICSOFT-EA*, Lisbon, Portugal, pp.79–90, Springer.

Alenezi, M. and Zarour, M. (2016) 'Does software structures quality improve over software evolution? Evidences from open-source projects', *International Journal of Computer Science and Information Security*, Vol. 14, No. 1, p.61.

Aleti, A., Buhnova, B., Grunske, L., Koziolek, A. and Meedeniya, I. (2013) 'Software architecture optimization methods: a systematic literature review', *IEEE Transactions on Software Engineering*, Vol. 39, No. 5, pp.658–683.

Barnes, J.M., Garlan, D. and Schmerl, B. (2014) 'Evolution styles: foundations and models for software architecture evolution', *Software & Systems Modeling*, Vol. 13, No. 2, pp.649–678.

Bass, L., Clements, P. and Kazman, R. (2003) *Software Architecture in Practice*, pp.110–111, Addison-Wesley Professional, USA, ISBN: 978-0-3211-5495-8.

Behnamghader, P., Le, D.M., Garcia, J., Link, D., Shahbazian, A. and Medvidovic, N. (2017) 'A large-scale study of architectural evolution in open-source software systems', *Empirical Software Engineering*, Vol. 22, No. 3, pp.1146–1193.

Bézivin, J. and Briot, J.P. (2004) 'Sur les principes de base de l'ingénierie des modèles', *L'OBJET*, Vol. 10, No. 4, pp.145–157.

Bondar, S., Hsub, J.C., Pfouga, A., Stjepandic, J. and Aprostep, A.G. (2017) 'Zachman Framework in the Agile Digital Transformation', *Proceedings of the 24th International Conference on Transdisciplinary Engineering: A Paradigm Shift*, ISPE Inc., Singapore, Vol. 5, p. 67, IOS Press.

Breivold, H.P., Crnkovic, I. and Larsson, M. (2012) 'A systematic review of software architecture evolution research', *Information and Software Technology*, Vol. 54, No. 1, pp.16–40.

Brooks, R.A. (1989) 'A robot that walks; emergent behaviors from a carefully evolved network', *Neural Computation*, Vol. 1, No. 2, pp.253–262.

Buckley, J., Mens, T., Zenger, M., Rashid, A. and Kniesel, G. (2005) 'Towards a taxonomy of software change', *Journal of Software: Evolution and Process*, Vol. 17, No. 5, pp.309–332.

Cavalcante, E., Batista, T. and Oquendo, F. (2015) 'Supporting dynamic software architectures: from architectural description to implementation', *The 12th Working Conference on Software Architecture (WICSA), IEEE/IFIP*, pp.31–40.

Chaki, S., Diaz-Pace, A., Garlan, D., Gurfinkel, A. and Ozkaya, I. (2009) 'Towards engineered architecture evolution', *Proceedings of the ICSE Workshop on Modeling in Software Engineering, IEEE*, Washington, USA, pp.1–6.

Chapin, N., Hale, J.E., Khan, K.M., Ramil, J.F. and Tan, W.G. (2001) 'Types of software evolution and software maintenance', *Journal of Software: Evolution and Process'*, Vol. 13, No. 1, pp.3–30.

Clements, P., Garlan, D., Little, R., Nord, R. and Stafford, J. (2003) 'Documenting software architectures: views and beyond', *Proceedings of the 25th International Conference on Software Engineering, IEEE*, Portland, OR, USA, pp.740–741.

Ebert, C., Kuhrmann, M. and Prikladnicki, R. (2016) 'Global software engineering: Evolution and trends', *The 11th International Conference on Global Software Engineering (ICGSE), IEEE*, Orange County, California, USA, pp.144–153.

Fitzgerald, B. and Stol, K.J. (2017) 'Continuous software engineering: a roadmap and agenda', *Journal of Systems and Software*, Vol. 123, No. 1, pp.176–189.

Front-Conte, A., Giraudin, J.P., Rieu, D. and Saint-Marcel, C. (1999) 'Réutilisation et Patrons d'ingénierie', in Oussalah, C. (Ed.): *Génie objet: analyse et conception de l'évolution*, pp.91–136, Hermès, Paris.

Gummalla, M.S. and Rao, G.V. (2018) 'A review of adaptive dynamic with software architecture', *International Journal of Applied Engineering Research*, Vol. 13, No. 7, pp.4626–4632.

Hassan, A. and Oussalah, M. (2016) 'Meta-evolution style for software architecture evolution', *the International Conference on Current Trends in Theory and Practice of Informatics*, pp.478–489, Springer, Berlin, Heidelberg, ISBN 978-3662491911.

ISO9000 *Concept and Use of the Process Approach for Management Systems*, Document: ISO/TC 176/SC 2/N 544R3 [online] http://www.iso.org/iso/fr/04 (accessed 8 October 2017).

Jazayeri, M. (2005) 'Species evolve, individuals age', *Proceedings of the 8th International Workshop on Principles of Software Evolution*, IEEE Computer Society, Washington, DC, USA, pp.3–9.

Kniesel, G., Costanza, P. and Austermann, M. (2001) 'Jmangler – a framework for load-time transformation of java class files', *Proceedings of the 1st International Workshop on Source Code Analysis and Manipulation, IEEE*, Florence, Italy, pp.98–108.

Lai, Y., Liu, Z. and Ye, T. (2018) 'Software behaviour analysis method based on behaviour template', *International Journal of Simulation and Process Modelling*, Vol. 13, No. 2, pp.126–134.

Landi, A.D.S., Chagas, F., Santos, B.M., Costa, R.S., Durelli, R., Terra, R. and de Camargo, V.V. (2017) 'Supporting the specification and serialization of planned architectures in architecture-driven modernization context', *The 41st Annual Computer Software and Applications Conference (COMPSAC), IEEE*,Torino, Italy, pp.327–336.

Lapalme, J., Gerber, A., van der Merwe, A., Zachman, J., De Vries, M. and Hinkelmann, K. (2016) 'Exploring the future of enterprise architecture: a Zachman perspective', *Computers in Industry*, Vol. 79, No. 5, pp.103–113.

Lehman, M.M. (1980) 'Programs, life cycles, and laws of software evolution', *Proceedings of the IEEE*, Vol. 68, No. 9, pp.1060–1076.

Mens, T. and van Gorp, P. (2006) 'A taxonomy of model transformation', *Electronic Notes in Theoretical Computer Science*, No. 152, pp.125–142.

Mens, T., Wermelinger, M., Ducasse, S., Demeyer, S., Hirschfeld, R. and Jazayeri, M. (2005) 'Challenges in software evolution', *The 8th International Workshop on Principles of Software Evolution, IEEE*, pp.13–22.

Mittal, S. and Vetter, J.S. (2016) 'A survey of techniques for modelling and improving reliability of computing systems', *IEEE Transactions on Parallel and Distributed Systems*, Vol. 27, No. 4, pp.1226–1238.

Monperrus, M. (2018) 'Automatic software repair: a bibliography', *ACM Computing Surveys (CSUR)*, Vol. 51, No. 1, p.17.

Ni, F. and Li, R. (2017) 'Hierarchical iterative modeling approach for agile SOA', *The 2nd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC), IEEE*, Chengdu, China, pp.1778–1782.

Nigro, C., Nigro, L. and Sciammarella, P.F. (2018) 'Modelling and analysis of multi-agent systems using UPPAAL SMC', *International Journal of Simulation and Process Modelling*, Vol. 13, No. 1, pp.73–87.

Noran, O. (2003) 'An analysis of the Zachman framework for enterprise architecture from the GERAM perspective', *Annual Reviews in Control*, Vol. 27, No. 2, pp.163–183.

Oreizy, P. and Taylor, R.N. (1998) 'On the role of software architectures in runtime system reconfiguration', *IEEE Proceedings-Software*, Vol. 145, No. 5, pp.137–145.

Oussalah, M., Sadou, N. and Tamzalit, D. (2006) 'SAEV: a model to face evolution problem in software architecture', *The 2nd Proceedings of the International ERCIM Workshop on Software Evolution*, Lille, France, pp.137–146.

Oussalah, M. (2014) *Architectures logicielles: principes, téchniques et outils*, pp.347–384, Hermès Sciences-Lavoisier, Paris, ISBN 978-2-7462-4517-4.

Pigoski, T.M. (1996) *Practical Software Maintenance: Best Practices for Managing Your Software Investment*, John Wiley & Sons, New York, ISBN: 978-0-471-17001-3.

Rising, L. and Janoff, N.S. (2000) 'The Scrum software development process for small teams', *IEEE Software*, Vol. 17, No. 4, pp.26–32.

Shahbazian, A., Nam, D. and Medvidovic, N. (2018) 'Toward predicting architectural significance of implementation issues', *The 15th International Conference on Mining Software Repositories (MSR), IEEE/ACM NY*, Gothenburg, Sweden.

Shaw, M., DeLine, R., Klein, D.V., Ross, T.L., Young, D.M. and Zelesnik, G. (1995) 'Abstractions for software architecture and tools to support them', *IEEE Transactions on Software Engineering*, Vol. 21, No. 4, pp.314–335.

Stol, K.J., Ralph, P. and Fitzgerald, B. (2016) 'Grounded theory in software engineering research: a critical review and guidelines', *Proceedings of the 38th International Conference on Software Engineering (ICSE)*, IEEE/ACM, Austin, TX, USA, pp.120–131.

Swanson, E.B. (1976) 'The dimensions of maintenance', *Proceedings of the 2nd IEEE International Conference on Software Engineering*, San Francisco, California, USA, pp.492–497.

Terho, H., Suonsyrjä, S., Systä, K. and Mikkonen, T. (2017) 'Understanding the relations between iterative cycles in software engineering', *Proceedings of the 50th Hawaii International Conference on System Sciences, (HICSS)*, Waikoloa, pp.5900–5909, ISBN: 978-0-9981331-0-2.

van de Wetering, R. and Bos, R. (2016) 'A meta-framework for efficacious adaptive enterprise architectures', *The 19th International Conference on Business Information Systems*, Leipzig, Germany, pp.273–288, Springer.

Villegas, N.M., Tamura, G. and Müller, H.A. (2017) 'Architecting software systems for runtime self-adaptation: concepts, models, and challenges', *Managing Trade-offs in Adaptable Software Architectures*, pp.17–43, Elsevier, USA, ISBN: 9780128028919.

Weinreicha, R. and Grohera, I. (2016) 'Supplementary material to: 'Existing software architecture knowledge management approaches and their support for knowledge management activities: a systematic literature review'', *Journal Information and Software Technology*, Vol. 80, No. 3, pp.265–286.

Williams, B.J. and Carver, J.C. (2010) 'Characterizing software architecture changes: a systematic review', *Information and Software Technology*, Vol. 52, No. 1, pp.31–51.

Zachman, J.A. (1987) 'A framework for information systems architecture', *IBM Systems Journal*, Vol. 26, No. 3, pp.276–292.

Zimmermann, O. (2015) 'Architectural refactoring: a task-centric view on software evolution', *IEEE Software*, Vol. 32, No. 2, pp.26–29.