# JVLC

## Journal of Visual Language and Computing

Volume 2019, Number 1

Journal of Visual Language and Computing

journal homepage: www.ksiresearch.org/jvlc

# Comprehension of Software Architecture Evolution supported by Visual Solutions: A Systematic Mapping and a Proposed Taxonomy

Joao Werther[a], Glauco de Figueiredo Carneiro[b] and Rita Suzana Pitangueira Maciel[a]

[a]Federal University of Bahia, Salvador, BRA
[b]Universidade Salvador (UNIFACS), Salvador, BRA

ARTICLE INFO

*Article History:*
Submitted 8.1.2019
Revised 8.20.2019
Second Revision 8.20.2019
Accepted 8.20.2019

*Keywords*:
software architecture
software visualization
software architecture evolution
software architecture comprehension

ABSTRACT

*Context:* Software visualization has the potential to support specialized stakeholders to understand the software architecture (SA) evolution. To the best of our knowledge, there is no guideline to support the use of visual solutions towards SA evolution comprehension. *Goal:* Analyze the use of visual solutions for the purpose of comprehension with respect to software architecture evolution from the point of view of software architects and developers in the context of both academia and industry. *Method:* We conducted a Systematic Mapping Study to achieve the stated goal. *Results:* The study identified 211 papers published from January 2000 to May 2019 as a result of the search strings execution. We selected 21 primary studies and identified a gap in terms of a taxonomy to assist specialists in the development or classification of solutions to support the comprehension of software architecture evolution using visual resources. *Conclusion:* We observed that despite the relevance of the use of visual solutions to support the comprehension of software architecture evolution, only 21 studies have reported these initiatives, suggesting that there is still room for the use of different visual metaphors to represent its components, relationships and evolution throughout the releases.



## 1. Introduction

Software evolution reflects changes undergone by the software during its lifespan [1] [2]. The study of software evolution is essential to better support changes in software requirements over time, keeping its integrity at a lowest possible cost [3] [4] [5]. Software Architecture (SA) is the design model used to build and evolve a software system [6]. Throughout the analysis of the software architecture, it is possible to understand the dimensions along which a system is expected to evolve [1]. The importance of SA in software evolution process is that a software built without an adaptable architecture normally will degenerate sooner than others with a change-ready architecture [1]. The evolution of a SA can be the result of changes in the current SA to accommodate business demands, new technologies and/or platform or other reason that impacts the SA [7]. The comprehension of SA is essential for the development and evolution of software systems [8][9] and can be supported by software visualization resources to understand key SA characteristics regarding architectural models and design decisions [8] [10].

Software Visualization (SV) has been used to support the SA comprehension in the context of software systems. This support usually occurs through the use of different visual metaphors to represent its components, relationships and evolution throughout the releases [9]. The use of visual solutions to represent SA and its architecture design decision can improve significantly their understanding [9]. SA visualization may concern with the evolution of its components throughout the releases, not only its static visualization [11].

To the best of our knowledge, there is no guideline to support the use of visual solutions towards SA evolution comprehension. For this reason, we conducted this Systematic Mapping Study (SMS) to identify evidence in the literature on this issue provided by papers published in peer-reviewed conferences and journals from January 2000 to May 2019.

jwertherf@gmail.comn (J. Werther); glauco.carneiro@unifacs.br (G.d.F. Carneiro); ritasuzana@dcc.ufba.br (R.S.P. Maciel)
www.unifacs.br (G.d.F. Carneiro); pgcomp.dcc.ufba.br (R.S.P. Maciel)
ORCID(s): 0000-0001-6241-1612 (G.d.F. Carneiro); 0000-0003-3159-6065 (R.S.P. Maciel)

DOI reference number: 10.18293/JVLC2019N1-008

From the 211 studies retrieved by the search string applied in specific electronic databases, we selected 21 studies to gather evidence to answer the stated research questions.

We aim at identifying strengths, weaknesses and research gaps related to the use of visual solutions to support the comprehension of software architecture evolution. Additionally, the analysis of the selected papers allowed us to identify the opportunity to propose a taxonomy to characterize and evaluate visual solutions to support the comprehension of SA evolution, representing their main characteristics, properties and features. According to Price, Baccker and Small (1993) [12], a well-founded taxonomy provides a common terminology and a set of related concepts that facilitate the communication and classification of information in a specific area, enabling the identification and cataloging of new discoveries and ideas in this area. The selected studies were classified using the *category* dimension of the proposed taxonomy as follows: 14% were categorized as *Description*, 38% as *Technique*, 67% as *Tool* and 19% as *Environment*. Besides the *category* dimension, the proposed taxonomy contains other five additional dimensions: *stage*, *visualization form*, *static representation*, *dynamic representation* and *architectural tasks*.

This paper is an extension of an earlier conference paper [13]. Our original work related a systematic mapping conducted for analyze the use of visual solutions for the purpose of comprehension with respect to software architecture evolution from the point of view of software architects and developers in the context of both academia and industry. In this extension work, a new version of this systematic mapping, we adjusted the PICO criteria to build a new version of the search string maintaining the original goal and research questions. We aimed at increasing the number of selected papers and therefore improve the findings discussed in this study. We also included a new background section to present a contextualization of issues related to software architecture, software evolution, architecture evolution, software visualization and software architecture visualization considered relevant in this systematic mapping. We also narrowed the interval of publication of searched papers by changing the upper bound of the interval from December 2018 to May 2019. However, we did not identified impact of this change in the number of retrieved papers.

The remainder of this paper is organized as follows. Section 2 shows a brief contextualization on software architecture, software architecture evolution and their respective relationships with software visualization. Section 3 discusses related works and Section 4 presents the design we adopted to conduct this SMS. The Section 5 reports the results and findings of this study, and proposes a new taxonomy for visual solutions to software architecture evolution. The Section 6 presents the answers to the stated research questions. Finally, we conclude and mention future work in Section 7..

## 2. Background

SA refers to a set of components of a software system, their connections and their principles and guidelines to manage the development and evolution during software life cycle [14]. SA describes the system's structure, interaction of its components and their core properties, playing an important role as an interface between requirements and source code [15]. SA is a possible mean to provide evidence if in fact the software is in compliance with its non-functional requirements (e.g performance, reliability, scalability, etc.) [15]. SA is also used to describe high-level structures and behavior of a software system [16], which contributes to support the software evolution [17]. In addition, SA provides a better understanding of the software to its stakeholders [15].

Software evolution refers to the dynamic behavior of software systems as they are submitted to changes over time [18] [19]. The analysis of software evolution is essential to both understand past and to plan future changes in the software, keeping its integrity (mainly the architectural one), at a lowest possible cost [20]. The importance of SA in software evolution process is that a software built without an adaptable architecture is prone to have shorter lifespan than others that have a change-ready architecture, impairing its evolution over time [21]. The analysis of SA evolution significantly improves the perception of software evolution. SA allows the planning and restructuring of the software system in a high level of abstraction, being a valuable reference for the discussion with stakeholders regarding the quality and business trade-off [7]. The comprehension of SA is essential for the development and evolution of software systems. Due to the amount of information, this task can be more effective when supported by resources that help to soften the required cognitive effort to perform it [11].

Shahin, Liang and Khayyambashi [9] argued that the use of SV to represent SA and also its evolution can improve the comprehension of both. In fact, SV provides solutions to support SA comprehension and its corresponding evolution [8]. The SA visualization is an important area in SV [11] and visually represents components and structures from a given SA associated with its architecture design decisions [6]. It may involve not only the visualization of software structures and their relationships, but also the evolution of these structures in the software lifespan [11]. Gallagher, Hatch and Munro (2008) [22] stated that SA visualization can improve the comprehension of software systems for all their stakeholders in all their aspects, along their evolution. Besides being fundamental to discuss and understand the SA in accordance with the variety of project stakeholders, SA visualization is also critical for decisions related to SA [23] and can visually represent some architectural design decisions [6]. In the framework proposed by Gallagher, Hatch and Munro (2008) [22] to evaluate SA visualization tools, one of the features of the key area Task Support (TS) is the *TS Show evolution*. The question of this feature using the GQM approach [24] was "*Does visualization show the evolution of software architecture?*". This framework considers that a SA visualization tool should indeed provide facility to show

evolution, whether in basic or advanced way [22].

## 3. Related Works

Software visualization has been used in different areas of software engineering such as software architecture, software evolution and software design [8][25]. In the following paragraphs we present results provided by a selection of secondary studies that discussed the use of software visualization to support the execution of activities targeting software architecture. This is not an exhaustive list, it is rather an illustrative set of relevant papers that motivated the conduction of this systematic mapping.

Shahin, Liang and Babar [8] conducted a systematic review to characterize the use of Visualization Techniques (VT) to represent SA in different application domains. Results classified the VTs into four types, according to its popularity: graph-based, notation-based, matrix-based and metaphor-based. From this set, the graph-based stood out for its popularity in industry. The same authors argued that VTs have been used to support SA activities for several purposes: (i) the understanding of architecture evolution; (ii) the understanding of static characteristics of architecture; and (iii) search, navigation, and exploration of architecture design [8]. Additionally, the systematic review reported that VTs have been applied to support SA related activities in a large range of domains. From those domains, *software graphics* and *distributed system* have received special attention from the industry. Finally, the authors argued that SV is one of the interesting ways to support the understanding of the rationale behind design decisions that affect software architecture [8]. It should be mentioned that Shahin, Liang and Babar [8] focused on VT to represent SA. They did not discussed the use of SV to support activities related to the comprehension of SA evolution.

Telea, Voinea and Sassenburg (2010) [25] performed a survey to investigate the use of visual tools for the comprehension of SA from the perspective of stakeholders. They analyzed the results using software architecture visualizations tools (AVTs) aiming to guide industrial practitioners in the adoption of tools and techniques according requirements and capabilities of each type. The authors considered three types of stakeholders: technical users (developers), project managers/lead architects, and consultants. They concluded that AVTs were effective to support technical users and less adequate for consultants, according to expectancy of each stakeholder [25]. Although Telea, Voinea and Sassenburg (2010) [25] fovused on the use of visual solutions for SA comprehension, they did not focused on SA evolution solutions.

Breivold, Crnkovic and Larson (2012) [1] conducted a systematic review focusing on software architecture evolution. The goal of the review was to provide an overview at the architectural level of existing approaches in the analysis of software evolution, and also examine possible impacts of this theme on both research and industry. The authors identified five main categories related to this theme: (i) techniques supporting quality consideration during software architecture design, (ii) architectural quality evaluation, (iii) economic evaluation, (iv) architectural knowledge management, and (v) modeling techniques [1]. The conclusion of this study emphasized the need of development and improvement of methods, process and/or tools to design architecture in large systems, due to the amount and complexity of artifacts produced and used during their respective lifecycle. This study also presented conclusions for researchers and practitioners, including considering the possibility to elaborate new ideas beyond Lehman's lays (about software evolution). Additionally, this paper also reported the existence of only few works targeting the theme, indicating the need of more research effort in this ares [1]. Despite Breivold, Crnkovic and Larson (2012) [1] studied the evolution of SA, they did not emphasized how visual solutions can be used to support the comprehension of SA evolution.

We could identify the relevant contribution of the aforementioned studies to the SA area, including SA evolution. However, they did not focused on visual solutions to support the comprehension of the SA evolution.

**Table 1**
The Goal of this SMS according to the GQM Approach

| | |
|---|---|
| *Analyze* | the use of visual solutions |
| *for the purpose of* | comprehension |
| *with respect to* | software architecture evolution |
| *from the point of view of* | software architects and developers |
| *in the context of* | both academia and industry |

## 4. Research Design

We conducted a SMS to find evidence for the use of visual solutions to support the comprehension of SA evolution during the software lifespan. A SMS is a form of a systematic literature review (SLR) with more general research questions, aiming to provide an overview of the given research [26]. We decided to conduct a SMS due to the potential that this methodology has to reduce the analysis bias, through the establishment of selection procedures [27].

### 4.1. Planning

The protocol we adopted to conduct this secondary study was comprised of objectives, criteria for considering papers, research questions, selected electronic databases, search strings, selection procedures, exclusion, inclusion and quality criteria to select the studies from which we aim to answer the stated research questions [27]. The protocol of this SMS and related artifacts are available in a public Github repository [1]. The goal of this study is presented in Table 1 according to the GQM approach [28].

[1] https://github.com/jvlc2019saevolution/submission

**Table 2**
PICO Criteria for Search Strings

| | |
|---|---|
| **(P)opulation** | studies in software architecture |
| **(I)ntervention** | visual solutions to support the comprehension of software architecture evolution |
| **(C)omparison** | not applicable |
| **(O)utcomes** | solutions (i.e., tools, techniques, environment, approaches, models or methodology) with focus on visual resources to support software architecture evolution; visual solutions to software architecture evolution; use of visual resources to comprehension of software architecture evolution |

The Research Question (RQ) is "*How have researchers and practitioners from academia and industry used software visual solutions to support the comprehension of software architecture evolution based on papers published in the peer-reviewed literature?*". This research question is in line with the goal of this review, and has been derived into four specific research questions, as follows: Specific Research Question 1 (SRQ1): *What are the main visual solutions to support the software architecture evolution comprehension?* Specific Research Question 2 (SRQ2): *What are the purposes of each visual solution to support the software architecture evolution comprehension?* Specific Research Question 3 (SRQ3): *How the solutions designed to visually support comprehension of software architecture evolution can be classified?* Specific Research Question 4 (SRQ4): *Which visual forms are used to support comprehension of software architecture evolution?*

The motivation behind RQ is justified by the acknowledgment that the comprehension of the software architecture evolution is required to tackle issues or improvements related to the software architecture and its evolution throughout releases [23] [6] [9] [8] [7] [3]. The specific research questions have the goal to gather evidence to support the answer of the stated RQ.

We considered the PICO criteria to define the search strings, as shown in Table 2. The search strings are based on this criteria for the selective process of papers for this review.

The formation of the search string applied in the electronic databases is shown in Tables 3 and 4. The Table 3 refers to major terms for the research objectives, built using the PICO criteria. We also used of alternative terms and synonyms of these major terms. For example, the term *visualization* can be associated with terms such as *visual*, *visualizing* and *visualize*. These alternative terms, as shown in Table 4,are also included in the search string. We built the final search string by joining the major terms with the boolean "AND" and joining the alternative terms to the main terms with the boolean "OR". The focus of the formed search strings is to focus on papers targeting the research questions of this systematic mapping.

Table 5 presents the electronic databases from which we retrieved the papers along with the respective search strings used to retrieve the papers. Table 6 presents the criteria for exclusion and inclusion of papers in this review. The *OR* connective adopted in the exclusion criteria, means that the exclusion criteria are independent, i.e., meeting only one criterion is enough to exclude the paper. On the other hand, the *AND* connective in the inclusion criteria, means that all inclusion criteria must met to select the paper under analysis. Table 6 also presents the quality criteria used for this review represented as questions that were adopted and adjusted from Dyba and Dingsoyr [29]. A critical examination following the quality criteria established in this table was performed in all remaining papers that passed the exclusion and inclusion criteria. All these criteria must met (i.e., the answer must be YES for each one) to permanently select the paper, otherwise the paper must be excluded. The exclusion,

**Table 3**
Major terms for the research objectives

| Criteria | Major Terms |
|---|---|
| (P)opulation | AND "software architecture" |
| (I)ntervention | AND "comprehension" AND "evolution" |
| (C)omparison | Not Applicable |
| (O)utcomes | AND "visual" AND "solution" |

**Table 4**
Alternative terms from majors terms

| Major Term | Alternative Terms |
|---|---|
| "evolution" | ("evolution" OR "evolve" OR "evolving") |
| "comprehension" | ("comprehension" OR "understanding" OR "understand" OR "support" OR "analysis" OR "evaluation" OR "examination" OR "explore" OR "exploring") |
| "solution" | ("tool" OR "environment" OR "technique" OR "approach" OR "model" OR "methodology" OR "solution") |
| "visual" | ("visualization" OR "visualizing" OR "visualize" OR "visual") |

**Table 5**
Electronic Databases Selected for this SMS

| Database and URL | Search Strings |
|---|---|
| Scopus www.scopus.com | ("software architecture" AND ("evolution" OR "evolve" OR "evolving") AND ("comprehension" OR "understanding" OR "understand" OR "support" OR "analysis" OR "evaluation" OR "examination" OR "explore" OR "exploring") AND ("tool" OR "environment" OR "technique" OR "approach" OR "model" OR "methodology" OR "solution") AND ("visualization" OR "visualizing" OR "visualize" OR "visual") |
| ACM Digital Library portal.acm.org | (+"software architecture" +("evolution" "evolve" "evolving") +("comprehension" "understanding" "understand" "support" "analysis" "evaluation" "examination" "explore" "exploring") +("tool" "environment" "technique" "approach" "model" "methodology" "solution") +("visualization" "visualizing" "visualize" "visual")) |
| Engineering Village (Ei Compendex) www.engineeringvillage.com | ("software architecture" AND ("evolution" OR "evolve" OR "evolving") AND ("comprehension" OR "understanding" OR "understand" OR "support" OR "analysis" OR "evaluation" OR "examination" OR "explore" OR "exploring") AND ("tool" OR "environment" OR "technique" OR "approach" OR "model" OR "methodology" OR "solution") AND ("visualization" OR "visualizing" OR "visualize" OR "visual")) |
| IEEE Xplore ieeexplore.ieee.org | ("software architecture" AND ("evolution" OR "evolve" OR "evolving") AND ("comprehension" OR "understanding" OR "understand" OR "support" OR "analysis" OR "evaluation" OR "examination" OR "explore" OR "exploring") AND ("tool" OR "environment" OR "technique" OR "approach" OR "model" OR "methodology" OR "solution") AND ("visualization" OR "visualizing" OR "visualize" OR "visual") |

**Table 6**
Exclusion, Inclusion and Quality Criteria

| Type | Id | Description | Connective or Answer |
|---|---|---|---|
| Exclusion | E1 | Published earlier than 2000 | OR |
| Exclusion | E2 | The paper was not published in a peer-reviewed journal or conference | OR |
| Exclusion | E3 | The paper does not present a primary study | OR |
| Exclusion | E4 | The paper is not written in English | OR |
| Exclusion | E5 | The paper has less than 3 pages | OR |
| Inclusion | I1 | The paper must present an approach in the usage of visual solution to support the comprehension of software architecture evolution | AND |
| Quality | Q1 | Are the aims of the study clearly specified? | YES/NO |
| Quality | Q2 | Is the context of the study clearly stated? | YES/NO |
| Quality | Q3 | Does the research design support the aims of the study? | YES/NO |
| Quality | Q4 | Has the study an adequate description of the visual solution? | YES/NO |
| Quality | Q5 | Is there a clear statement of findings by applying the visual solution to support the comprehension of software architecture evolution? | YES/NO |

**Table 7**
Steps of the Selection Process

| Step | Description |
|---|---|
| 1 | Apply the search strings to obtain a list of candidate papers in specific eletronic databases. |
| 2 | Remove replicated papers from the list. |
| 3 | Apply the exclusion criteria in the listed papers. |
| 4 | Apply the inclusion criteria after reading abstracts, introduction and conclusion in papers not excluded in step 3. |
| 5 | Apply quality criteria in selected papers in step 4. |

**Table 8**
Classification Options for Each Retrieved Paper

| Classification | Description |
|---|---|
| Excluded | Papers met the exclusion criteria. |
| Not Selected | Papers not excluded due to the exclusion criteria, but did not meet the inclusion or quality criteria. |
| Selected | Papers did not meet the exclusion criteria and met both the inclusion and quality criteria. |

inclusion and quality criteria were used in the selection process as presented in Table 7. According to Table 8, at the end of the selection process, all the retrieved papers were classified in one of the three options: *Excluded*, *Not Selected* and *Selected*.

## 4.2. Execution

The quantitative evolution of papers throughout the execution of this SMS is summarized in Figure 1. The figure uses the PRISMA flow diagram [30] and shows the performed steps and the respective number of documents for each phase of the SMS, following the outline described in Subsection 4.1.

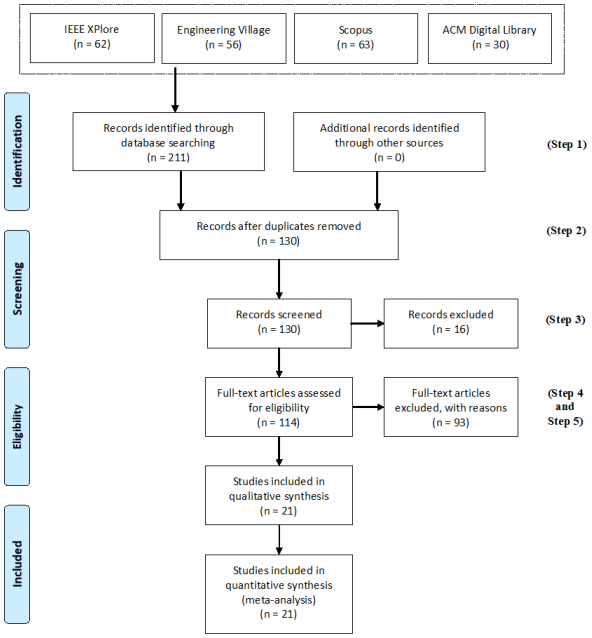


**Figure 1:** Procedures and its results in the papers selection process.

**Table 9**
Effectiveness of the Search Strings

| Database | Selected Papers | Excluded Papers | Not Selected Papers | Replicated Papers | Total Search Results | Search Effectiveness |
|---|---|---|---|---|---|---|
| ACM Digital Library | 3 | 3 | 23 | 1 | 30 | 10.0% |
| Engineering Village | 2 | 3 | 16 | 35 | 56 | 3.6% |
| IEEE Xplore | 12 | 5 | 38 | 7 | 62 | 19.4% |
| Scopus | 4 | 5 | 16 | 38 | 63 | 6.3% |
| **TOTAL** | **21** | **16** | **93** | **81** | **211** | **10.0%** |

Table 9 presents the effectiveness of the the search strings showed in Table 5 considering the 211 retrieved papers. The electronic database that more contributed with selected studies was the *IEEE Xplore* with five papers, corresponding to a search effectiveness of 18.5%. The twelve selected papers represented 13.0% of all 211 retrieved papers.

The Figure 2 presents a overview of contribution of each exclusion criterion in total of excluded papers. The exclusion criterion that had more contribution was the E1 criterion that says "*Published date less than 2000*", accounting for 50% of excluded papers.

The Figure 3 presents graphics that provide a overview of sources (electronic databases) distribution by papers status. The papers status is according to Table 8. The Figure 3a shows the selected papers distribution by source. The "IEEE Xplore" had the major contribution for selected papers with 57% of occurrences. The "Scopus" was the second with 19% of selected studies. The Figure 3b shows the not-selected papers distribution by source. The "IEEE Xplore" leaded with 41% of not-selected papers and "ACM Digital Library" came in second with 25%. The Figure 3c presents the excluded papers distribution by source. The "Scopus"

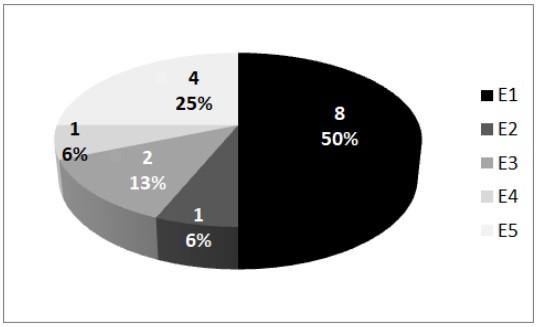


**Figure 2:** Contribution of exclusion criteria in total of excluded papers.

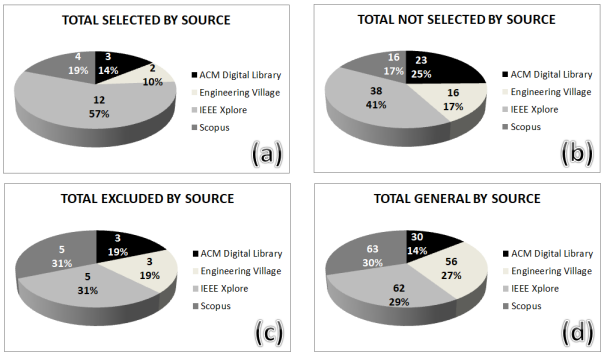


**Figure 3:** Overview of paper totalization by status and by electronic databases.

and “IEEE Xplore" databases had the majors contributions with 31% of excluded papers each one. The Figure 3d presents the distribution of all found papers by source. The “Scopus" had the major contribution with 30%, close to “IEEE Xplore" with 29% of all papers found in its databases searches.

## 5. Results

The Table 10 shows the list of 21 selected papers of this systematic mapping. All papers are labeled as “S” followed by the paper reference number. There is 18 selected papers that were published in conferences. The other 3 articles (S02, SO3 and S16) were published in journals. They are discriminated in the “Venue" column.

The paper **S01** [4] describes a solution aimed at enhancing the comprehension of the software architectural evolution based on visual resources. This solution proposes the use of efficient navigation and visualization of the history of software architectural changes throughout releases, integrating the use of evolution metrics with software visualization techniques. This integration has the goal to support both tracking and analysis of architectural changes from past releases. The authors developed the so called *Origin Analysis* method to analyze software structural change. This method supports the identification of possible origin of function or file that appears to be new in a later release of the software system, if it already existed in the system elsewhere [4]. This method highlights the use of two techniques in its implementation: *Bertillonage Analysis* and *Dependency Analysis*. The paper also performs a study of evolution of a real tool to demonstrate the use of *BEAGLE*, a prototype implementation of this solution that works as an integrated environment for studying software architecture evolution, as a validation form of the *Origin Analysis* [4]. This paper does not discuss explicitly its limitations, even though cites some of them.

The paper **S02** [31] proposes the usage of architecture stability or resilience concepts to evaluate a SA, using *Retrospective Analysis* to achieve this goal. *Retrospective Analysis* is a technique that verify the amount of changes applied is successive releases of a software system and analyze how smoothly the evolution took place. It works with a set of software metrics based in size, growth, changes and coupling, using visual tools to graphically observe the evolution of these metrics. The authors [31] affirms that the *Retrospective Analysis* can have many uses, not only to verify the stability, but also to calibrate the predictive evaluation results as well as to predict trends in evolution of software. The paper describes a case study of twenty releases of a telecommunication software system containing a few million lines of code to show how *Retrospective Analysis* may be performed. However, the paper does not provide any procedures to perform the *Retrospective Analysis*.

The paper **S03** [32] introduces a graphical and formal model to represent architecture styles and their reconfigurations in software evolution. The model specifies a SA using graphs and graph grammars to represent components (also called edges) and connections (also referred as nodes). Two techniques are formally presented. The first uses *Synchronized Hyperedge Replacement Systems*, dynamically allowing changes of components and connections according to their synchronization requirements specified in the nodes. The second technique specifies complex reconfigurations as transformations over derivations of graph grammars using *lambda-calculus*. However, the techniques are only described in a formal and summarized way and the paper does not mention any implementation or case study of them [32] .

The paper **S04** [33] presents the tool-set and methodology *Complex Systems Analysis Based Architecture (ABACUS)* as a visual solution to model complex systems, comprehend their architecture and analyze their characteristics and its potential changes. The paper reports that *ABACUS* allows to collect and merge all enterprise architectural information of a system into a unified repository and also evaluate system properties like performance, openness, and evolvability. The paper also highlights that *ABACUS* provides a hierarchical 3D visualization to allow to look across the enterprise architecture. The authors [33] emphasize that the use of *ABACUS* is not only use its tool-set, but also follow its methodology, as illustrated in Figure 4. They conclude affirming that the usage of *ABACUS* allows the architects conduct the architecture design and evolution based on quantifiable non-functional requirements. However, the paper does not presents any case study of *ABACUS* neither presents any evidence of its practical usage.

The paper **S05** [34] proposes a graphical technical description of the architectural instance of a software system,

**Table 10**
List of Selected Papers

| Ref. Label | Title | Venue | Year |
|---|---|---|---|
| S01 | An integrated approach for studying architectural evolution [4] | 10th International Workshop on Program Comprehension | 2002 |
| S02 | On architectural stability and evolution [31] | Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Vol. 2361, pp. 13-23 (JOURNAL) | 2002 |
| S03 | Two graph-based techniques for software architecture reconfiguration [32] | Electronic Notes in Theoretical Computer Science, Vol. 51, pp. 177 - 190 (JOURNAL) | 2002 |
| S04 | The ABACUS architectural approach to computer-based system and enterprise evolution [33] | 12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS) | 2005 |
| S05 | An Approach based on Bigraphical Reactive Systems to Check Architectural Instance Conforming to its Style [34] | First Joint IEEE/IFIP Symp. Theoretical Aspects of Software Engineering (TASE) | 2007 |
| S06 | Exploring Inter-Module Relationships in Evolving Software Systems [5] | 11th European Conference on Software Maintenance and Reengineering (CSMR) | 2007 |
| S07 | Technology Infusion of SAVE into the Ground Software Development Process for NASA Missions at JHU/APL[35] | IEEE Aerospace Conference | 2007 |
| S08 | The SAVE Tool and Process Applied to Ground Software Development at JHU/APL: An Experience Report on Technology Infusion [36] | 31st IEEE Software Engineering Workshop (SEW) | 2007 |
| S09 | Visualizing Software Architecture Evolution Using Change-Sets [37] | 14th Working Conference on Reverse Engineering (WCRE) | 2007 |
| S10 | YARN: Animating Software Evolution [38] | 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis | 2007 |
| S11 | Development of a Methodology, Software-Suite and Service for Supporting Software Architecture Reconstruction [39] | 14th European Conf. Software Maintenance and Reengineering | 2010 |
| S12 | Evolve: tool support for architecture evolution [40] | 33rd Int. Conf. Software Engineering (ICSE) | 2011 |
| S13 | Model-Based Software Architecture Evolution and Evaluation [41] | 19th Asia-Pacific Software Engineering Conference | 2012 |
| S14 | eCITY: A Tool to Track Software Structural Changes Using an Evolving City [42] | IEEE International Conference on Software Maintenance | 2013 |
| S15 | Run-time monitoring and real-time visualization of software architecture [43] | Asia-Pacific Software Engineering Conference, APSEC | 2013 |
| S16 | eCITY: Evolutionary software architecture visualization - An evaluation [44] | Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Vol. 8345 LNCS, pp. 201-224 (JOURNAL) | 2014 |
| S17 | eCITY+: A Tool to Analyze Software Architectural Relations Through Interactive Visual Support [45] | European Conference on Software Architecture Workshops | 2014 |
| S18 | The ARAMIS Workbench for Monitoring, Analysis and Visualization of Architectures Based on Run-time Interactions [46] | European Conference on Software Architecture Workshops | 2015 |
| S19 | Towards the understanding and evolution of monolithic applications as microservices [47] | 42nd Latin American Computing Conference, CLEI | 2016 |
| S20 | Supporting software architecture evolution by functional decomposition [48] | 5th International Conference on Model-Driven Engineering and Software Development - MODELSWARD | 2017 |
| S21 | EVA: A Tool for Visualizing Software Architectural Evolution [3] | 40th International Conference on Software Engineering (ICSE) | 2018 |

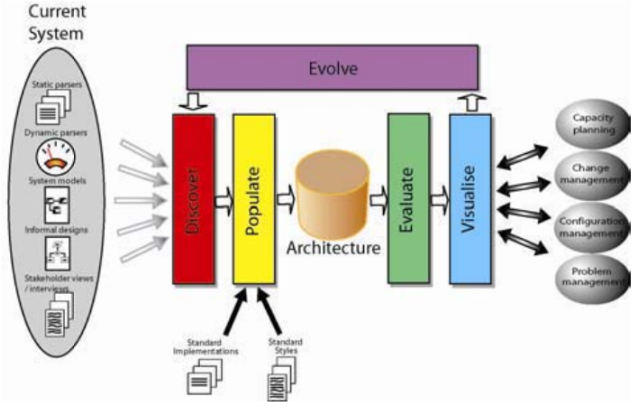


**Figure 4:** The *ABACUS* methodology [33]

and verify the compliance to its corresponding style. This solution is based on the *Bigraphical Reactive Systems* (BRS) to perform the verification with formal methods, and uses an extended version of a Bigraph to describe the instance. Besides supplying a visual method to specify architectural instances and styles, the solution proposed can enhance the ability to design evolving systems. Additionally, the paper shows two study cases in order to prove the effectiveness of this solution [34].

The paper **S06** [5] proposes an approach based on the visual representation of inter-module dependencies and relationships between SA components and modules throughout multiple versions of the software system. The *Semantic Dependency Matrix* is a visualization technique that shows dependencies between two modules with similar behavior classes. The *Edge Evolution Film Strip* is a visualization technique that presents the evolution of an inter-module relations in a software system along its multiples versions. These techniques were applied in two large open source software systems, in reverse engineering context, to exemplify them. The paper also purposes a pattern language for inter-module relationships. The studied examples are provided from an exploration prototype named *Softwarenaut* [5].

The papers **S07** [35] and **S08** [36] describe the NASA JHU/APL's experiences in using the *SAVE (Software Architecture Visualization and Evaluation*) tool and process. The *SAVE* tool addresses the understanding, maintenance and evolving issues, allowing software architects to navigate, visualize, analyze, compare, evaluate, and improve their software systems, all in only one environment. This tool can be also used to develop a new architecture, compare with the current one and still helps in change impact analysis, among others features. The architecture comparison can also occur between distinct software systems. The papers show how the *SAVE* tool has been successfully applied to the Common Ground software, a shared software architecture used by NASA missions software systems, in order to avoid further SA maintenance and evolution problems [35] [36]. However, the paper **S07** [35] reports in more detail the workshops that exposed the results found by the *SAVE* tool in the Common Ground's architecture analysis and evolution. Despite the presentation of the tool resources and features along the studies, these papers do not discuss its limitations.

The paper S09 [37] presents *Motive*, a prototype of an alternative approach to comprehension of software and its architecture evolution, which in addition to showing the evolution or changes of entities or components filtered by period and level of abstraction - like most of existing visual tools for SA evolution - allows users to visualize the net effect on the SA of any set of logically related changes. This set is called *change-set*. Two java open source systems were used to study and evaluate the *Motive* tool and this alternative approach. The authors [37] report that this evaluation showed that the identification and visualization of the impact of *change-sets* seems very promising to help architects and developers comprehend the evolution of a software system and its architecture.

The paper **S10** [38] introduces *YARN (Yet Another Reverse-engineering Narrative)*, a prototype tool that implements an approach to modeling, extracting and animating a SA evolution. Animating the changing dependencies is an intuitive and natural way to visually realize, identify and compare changes over system lifespan. The *YARN* generates animation through the *YARN balls*. The *YARN ball* is a type of circle composed by subsystems (vertices) and changing dependencies (edges), as shown in Figure 5. The animation starts by showing a *YARN ball* at a chosen baseline and then showing all the sequences of releases of a *YARN ball* progressively. Colour and thickness of edges varying according to the number of changes and how many dependencies exist between two modules. One of the suggested uses of a *YARN ball* by the authors [38] is to help the communication between stakeholders of change based dependency information in software projects. The authors also report that there was created an informal user survey to evaluate the usefulness of the solution and the understanding of the software and architecture evolution.

The paper **S11** [39] presents the description and goals of the project titled *"Development of a methodology, software-suite and service for supporting software architecture reconstruction"*, intended to develop a methodology and a toolset (environment) to do automatic architecture reconstruction of software systems through visual resources utilization. It also provides tracking of changes in architectural components during software evolution. At the time this paper was written (2010) the project was focused to systems that has been built using Java or .NET technologies and deal with SQL databases. This paper presents limitation of its study, but only in a summarized way. It also shows details of the current status (2010) of this project [39].

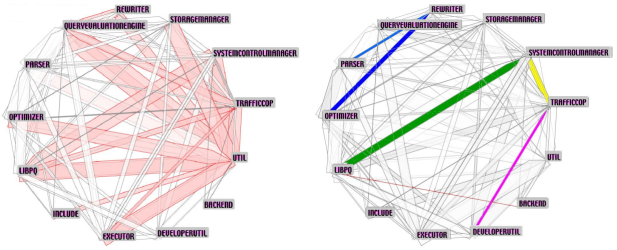


**Figure 5:** Shots of *YARN Balls* [38]

The paper **S12** [40] introduces *Evolve*, a graphical modeling tool that implements an ADL (*Architectural Description Language*) named *Backbone* that is focused on software architecture evolution. *Evolve* supports definition and evolution of SA using the *Backbone*, with particular attention to incremental change and unplanned change processing, very common activities in the software development and evolution process. *Backbone* provides constructs that allow changes that may result in architectural anomalies but *Evolve* is able to detect these anomalies. The paper shows the main characteristics of *Evolve* and how it deal with changes definition in SA, besides a brief use historic. The *Evolve* tool, at the time the paper was published (2011), was freely available for academic research and the production of open source software under the *GNU Affero General Public License version 3*.

The paper **S13** [41] presents and proposes the development of *ARAMIS (Architecture Analysis and Monitoring Infrastructure)*, an architecture meta-model based solution to extract run-time architecture information and provide data to generate new dynamic architecture views in real time. The solution provides visual representations of the monitored architecture at several abstraction levels, as well as the availability of methods to evaluate this architecture [41]. This study does not present any type of implementation (only technical description), despite presents some limitations of the solution.

The paper **S14** [42] introduces *eCITY*, a tool that helps software architects and developers to understand the software structure of their system. It allows the track of components' insertion, removal, or modification over system lifespan and provides an interactive visualization that provides an overview of changes. All of this implemented under a city metaphor using animations to represent the transitions of the architecture components and color coding to highlight the evolution and changes of these components (Figure 6). The

**Figure 6:** *eCITY*: a City View [42]

[h!]

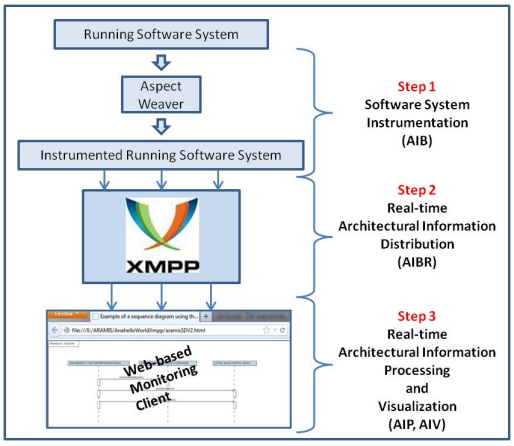


**Figure 7:** *ARAMIS*: a Prototype Overview [43]

*eCITY* provides an overview of the entire system at a desired point into its evolution process (life cycle), implementing it under a city metaphor, allowing the user an interactive way to understand and explore these changes. *eCITY* provides views to help the changes over time, like: *Timeline View*, an administrative view that uses charts and color to emphasize changes between software system versions; and *City View*, a city layout using animations to represent the transitions of the architecture components and color coding to highlight the evolution and changes of these components, as shown in Figure 6. The *eCITY* tool works with compile-time information, not providing dynamic views [42]. The *eCITY* was originally designed as an Eclipse plug-in. This paper also presents a summary of a conducted user study to emphasize its usefulness.

The paper **S15** [43] describes the implementation results of some core characteristics of *ARAMIS*, previously proposed in S07 [41]. *ARAMIS* is a approach for evolution and evaluation of software systems that relies on a infrastructure of run-time monitoring to manage the behavior of the system, in several abstraction levels. In this paper, a prototype of *ARAMIS* was developed focused only in reconstruction of object-level interactions. The prototype uses aspect-oriented techniques to extract and gather the run-time architectural information, and the *XMPP (Extensible Messaging and Presence Protocol)* to distribute the gathered information for visualization in real-time, through specialized components, as we can see in Figure 7. The evaluation of this process, according to prototype results, shows that *ARAMIS* can easily be used to demonstrate the behavior of the run-time monitored systems [43].

The paper **S16** [44] evaluates the *eCITY* tool, presented in S14 [42]. The authors designed and conducted a controlled experiment to perform the evaluation. In this experiment, they proposed *eCITY* as a tool for improving the analysis of SA evolution along its lifespan. They hoped that through the use of this tool, the architects would be more efficient and effective when perform the analysis of architectural changes. As a result of this experiment, the participants obtained an average gain of 170% in efficiency and an average gain of 15% in the effectiveness of the basic tasks of SA evolution. The authors [44] considered that efficiency means the time required for accomplishing a set of given tasks and effectiveness means the difference between the true and actual score related to a task.

The paper **S17** [45] presents *eCITY+*, an improved version of eCITY, presented in S14 [42], now combining the stable city layout and the *Hierarchical Edge Bundling (HEB)*, an useful technique to help the implementation of 3D visualization with the use of animation. The *eCITY+* is a later version of *eCITY* tool presented in S08 [42], owning characteristics similar from its predecessor. The *eCITY+* tool primarily differs from its earlier version in the use of *HEB* to highlight changes in both the hierarchical structure as well as the inter-dependencies between software components. The *eCITY+* tool, as its predecessor, performs analysis of software architecture relationships through interactive visual support, using the city metaphor to provide an overview of the entire system [45]. The *eCITY+* also was consciously developed as a plug-in to traditional SA maintenance tools.

The paper **S18** [46] updates the current status of the *ARAMIS*, previously presented in S13 [41] and S15[43], presenting it now for understanding, communication integrity validation and evaluation of the behavior view of a software architecture. This paper analysis a J2EE application to exemplify how *ARAMIS* can automatically validates the communication between the units of a software system, verifying if it corresponds to its architecture model, including making visualizations of these interactions available on higher and more comprehensible abstraction levels [46].

The paper **S19** [47] describes a technical solution to modernize monolithic applications into microservices using software visualization to support the comprehension of evolution process. This conceptual solution can provide a modernization process that uses a legacy system and generates a set of visual diagrams that help architects and developers to understand the system, also suggesting ways of code partitions for transforming into micro-services. The paper has focus only in an understanding stage of modernization, not in transformation stage [47]. The authors analyzed a large Java EE application to validate this solution.

The paper **S20** [48] presents a graphical approach that

combines a functional decomposition analysis technique with a non-functional impact analysis technique. Theoretically the functional decomposition prevents the architecture from degrading, but sometimes can be too expensive to implement it. This approach intend to avoid this problem. The functional decomposition technique uses individual relations (associations and attributes) as atomic units of decomposition, partitioning them between the subsystems according to how they are used by the system operations [49]. This technique facilitates the selection of decompositions of low coupling and high cohesion. The non-functional impact analysis technique uses the *KAMP (Karlsruhe Architectural Maintainability Prediction)* approach [50]. The combination of this two techniques guarantee a good equilibrium between functional modularity and non-functional concerns. Finally, this approach is illustrated with an example of evolution of a hypothetical system [48].

The paper **S21** [3] introduces *EVA (Evolution Visualization for Architectures)*, a visual tool to help software architects understand the evolution of architecture and therefore track and analyze architectural changes. This tool can visualize and explore architectures of software systems with a long life cycle, including stages of its evolution. *EVA* provides three main views: *Single-Release Architecture View*, that shows the architecture of only one software system version, as shown in Figure 8a; *3-D Architecture-Evolution View*, that depicts architectures of multiple software system versions in a single compositional view, as shown in Figure 8b; and Pairwise Architecture-Comparison View, that presents the architectural differences between two software system versions, as shown in Figure 8c. *EVA* allows its users to assess the impact of design decisions, as well as its rationale, which have influenced in the software architecture. As we can see in Figure 8, *EVA* uses color coding to distinct packages or groups of code level entities. The work is currently (2018) focused on showing the explicit reasons behind the architectural changes, in order to assist the rationale of tracking design during the software lifespan [3]. This article presents limitations of its study, but does not discuss them explicitly. The *EVA* tool was developed in Phyton and is available in a GitHub repository.

Unfortunately, we could not find any taxonomy to classify solutions that support the comprehension of SA evolution using visual resources. Then, to improve the understanding of these visual solutions previously described, we propose a taxonomy with the goal to classify their main characteristics, properties and features. Next, we present the taxonomy and the corresponding classification of each of the selected visual solutions in this SMS.

### 5.1. Taxonomy for Visual Solutions to Support SA Evolution Comprehension

In this subsection, we present the resulting taxonomy of visual solutions to support the comprehension of software architecture evolution. We argue that the concepts presented in this taxonomy are key to understand the characteristics of visual solutions and therefore to answer the research questions

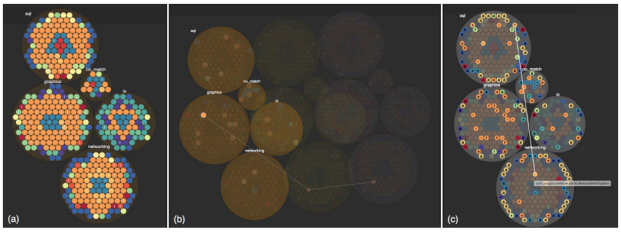

**Figure 8:** Three Types of Visualization of *EVA* [3]

of this secondary study. We used the software visualization taxonomy proposed by Price, Baecker and Small (1993) [12] and the framework purposed by Gallagher, Hatch and Munro (2008) [22] as references. The first focused on software visualization, whereas the latter focused on software architecture. For this reason, we adopted them as references to propose the new taxonomy. According to Sulr et al. [26], a taxonomy is consisted of a number of dimensions (e.g., “visualization form") with their attributes (e.g., “2D Elements",“Color Coding""). Each visual solution from the selected papers from this SMS can pertain to one or more attributes from a specific dimension, as we will describe in the following paragraphs.

The **Category** dimension is related to the type of proposed visual solution to support the comprehension of SA evolution. It is comprised of four attributes: *Description*, *Technique*, *Tool* and *Environment*. The *Description* attribute classifies the solution or its core idea as a main concept or a technical description of a visual solution. The *Technique* attribute informs whether the visual solution proposes the use of a technique, such as specialized procedures or processes. The *Tool* attribute means the solution presents or proposes a tool , or its development, to assist specialized users of software architecture to develop, or maintain software systems. The *Environment* attribute indicates that the solution explicitly presents or proposes an environment, i.e., an integrated set of tools to help software architecture specialized users in the development or maintenance of software systems. The value of this dimension for each solution is mandatory and it may have more than one attribute signalized.

The **Stage** dimension has four attributes. The *Conceptual* attribute indicates that the solution is still represented and referenced as a *concept*, not having any implementation. The *Project* attribute indicates that the solution has been under development as a project. The *Prototype* attribute means the solution is a prototype of the solution. Finally, the *Stable Release* attribute is related to the status of the solution as already in production as a stable release. The value of this dimension is also mandatory and only one attribute can be signalized for each solution.

The **Visualization Form** dimension defines the visual characteristics of the output of the solution [12]. It is characterizes by eight attributes as follows. The *2D Elements* attribute indicates that the solution uses 2D elements, such as 2D charts, diagrams, shapes, windows, figures and lines. The *3D Visualization* attribute indicates that the solution uses

3D resources for visualization. The *Animation* attribute means the solution uses resources of animation in the visual representations. The *Bigraphs* attribute marks the use of bigraphs in the visual solution. The *Visual Metaphor* attribute indicates the use of one or more visual metaphors in the visual representations to enhance the SA evolution understanding. The *Color Coding* attribute means the solution adopts a specific color system to represent the data. The *Tree-based* attribute is used to characterize solutions that use visual structures based on trees. The *UML-based* indicates that the solution uses UML diagrams as a form of visual representation.

The **Static Representation** dimension shows what architectural information can be extracted and represented before run-time [22]. It has two associated attributes: *Static Visualization* and *Recovery*. The *Static Visualization* attribute means the solution displays data exclusively related to static structure of the software system. The *Recovery* attribute indicates that the solution supports the retrieval of architectural data from specific sources. This dimension may have no attributes flagged.

The **Dynamic Representation** dimension shows what architectural information can be extracted and represented during run-time [22]. The *Dynamic Visualization* attribute means the solution displays data extracted during its execution (run-time). The *Events Monitoring* indicates if the solution perform the catch events during its execution. These events can be identified and associated with SA elements and thereby support the comprehension of specific scenarios of software architecture evolution. The *Live* attribute points out that the SA data is gathered in a *real time* fashion as the solution is executed [12] [22]. The *Post-mortem* attribute means the SA data to be gathered is produced in a *post-mortem* fashion by the solution, i.e., generated by its previous execution [12] [22]. This dimension may have no attributes flagged.

The **Architectural Tasks** dimension is related with features of the visual solution that support stakeholders to perform tasks that to some extent focuses on the software architecture and its evolution [22]. It has nine attributes as follows. The *Anomalies* attribute indicates that the solution supports the identification of anomalies, violations and inconsistencies occurrences related to SA. These occurrences can also influence the *Comprehension* attribute indicates the solution supports visual analysis tasks to improve the comprehension of SA and its evolution. Analysis tasks means tasks that generate results to facilitate the understanding the SA, its components, dependencies and relationships, as well as its evolution. They should support top-down or bottom-up approaches [22]. The *Styles* attribute indicates that the solution is able to identify architectural styles and/or verify its compliance with a predefined reference. The *Show Evolution* attribute indicates that the solution provides facilities to exhibit evolution evidence of a SA, in a basic or advanced way [22]. The *Construction* attribute indicates that the solution provides resources to add, change or remove SA elements in the visual representation. The *Evaluation* attribute means the solution supports SA quality analysis and also compliance evaluation. The *Comparison* attribute points out that the solution performs visual comparison among releases of the software system under analysis. A typical use of this attribute is the comparison between the *as-is* with the *to-be* architectures or the *as-designed* with *as-implemented* software architecture [22]. The *Tracking* indicates that the the solution supports the tracking of SA changes throughout its releases. This is a key resource to, for example, identify and trace the architectural decay of a SA, which impairs the software lifespan [3]. Finally, the *Rationale* indicates that the solution presents and make available the rationale behind the design decisions that somehow influences the SA.

## 5.2. Visual Solutions According to Taxonomy

The Table 11 shows the main characteristics, properties and features identified in the visual solutions of the selected papers from the perspective of the proposed taxonomy. These characteristics were identified and collected exclusively based on the text provided by the selected studies listed in Table 10.

The column labeled *Category* indicates different categories of solutions found in the selected papers, according the description presented in Subsection 5.1. The column labeled *Stage* means the stage of the solution proposed by the paper at the time it was published. The column labeled *Visualization Form* means the summary description of fundamental characteristics related to what can be exhibited in the visual solution. The content of this column is based on the *Form* category proposed by the taxonomy of Price [12]. The columns labeled *Static Representation*, *Dynamic Representation*, *Architectural Tasks* are based on keys areas proposed in Gallagher's framework [22]. The column *Others Features* shows complementary purposes, features and characteristics of the presented visual solution not listed before.

We decided to present a analysis about the visual solution named *EVA* to illustrate how the characteristics of the selected visual solutions presented in Table 11 can be determined. This analysis is shown in Table 12. The visual solution *EVA* was previously presented in the paper S21 [3] and its justifications come exclusively from this paper's content.

The Figure 9 presents the overview of current stages for all visual solutions referenced by Table 11. Its worth remembering that current stages means the stage at the time its study was published. Note that most of solutions (9 in 21) were in "Stable Release" stage, whereas few of them (2 in 21) were in "Project" stage.

The Figure 10 shows the categories of solution found in selected papers. The "Tool" category has the major preference in visual solutions, being present in 14 of 21 solutions found. The "Description" category is present in only three solutions.

Figure 11 shows that the tasks *Comprehension* and *Show Evolution* are adopted in all visual solutions from the selected studies. It means that all of them reported the support of analysis tasks to improve the SA comprehension. Moreover, they also reported the adoption of facilities to exhibit the SA evolution. These are in fact, minimum requirements of a visual solutions to support comprehension of SA evolution. The task *Comparison* is also representative in the an-

**Table 11**
Using the Proposed Taxonomy to Classify the Visual Solutions to SA Evolution

| Ref. Paper | Name of Visual Solution | Category | Stage | Visualization Form | Static Representation | Dynamic Representation | Architectural Tasks | Other Features |
|---|---|---|---|---|---|---|---|---|
| S01 | Beagle | Environment, Technique | Prototype | 2D Elements, Tree-based | Static Visualization, Recovery | N/A | Comprehension, Comparison, Show Evolution | Evolution metrics usage |
| S02 | Retrospective Analysis | Technique | Prototype | 2D Elements, Color Coding | Static Visualization, Recovery | N/A | Comprehension, Comparison, Show Evolution, Evaluation | N/A |
| S03 | Not named | Technique | Conceptual | 2D Elements, Visual Metaphor | Static Visualization | N/A | Comprehension, Construction, Show Evolution, Styles | Graph Grammar, Synchronized Hyperedge Replacement Systems, Hyperedge Replacement Grammar |
| S04 | ABACUS | Tool | Stable Release | 2D Elements, Tree-based, 3D Visualization, Color Coding | Static Visualization, Recovery | N/A | Comprehension, Construction, Show Evolution, Evaluation | Methodology-oriented |
| S05 | Not named | Description, Technique | Conceptual | 2D Elements, Bigraph | Static Visualization | Dynamic Visualization, Live | Comprehension, Comparison, Construction, Show Evolution, Styles | BRS resources |
| S06 | Film Strip and Dependency Matrix | Technique | Prototype | 2D Elements | Static Visualization, Recovery | N/A | Comprehension, Comparison, Show Evolution | N/A |
| S07 | SAVE | Tool, Environment | Stable Release | 2D Elements | Static Visualization, Recovery | N/A | Comprehension, Anomalies, Comparison, Construction, Show Evolution, Rationale, Styles, Evaluation | Process-oriented, Products comparison |
| S08 | SAVE | Tool, Environment | Stable Release | 2D Elements | Static Visualization, Recovery | N/A | Comprehension, Anomalies, Comparison, Construction, Show Evolution, Rationale, Styles, Evaluation | Process-oriented, Products comparison |
| S09 | Motive | Tool | Prototype | 2D Elements, UML-based | Static Visualization, Recovery | N/A | Comprehension, Show Evolution | N/A |
| S10 | YARN | Tool | Prototype | 2D Elements, Color Coding, Animation | Static Visualization, Recovery | N/A | Comprehension, Show Evolution | N/A |
| S11 | GOP | Tool, Environment | Project | 2D Elements, Color Coding | Static Visualization, Recovery | N/A | Comprehension, Comparison, Construction, Show Evolution, Tracking | Methodology-oriented |
| S12 | Evolve | Tool | Stable Release | 2D Elements, UML-based, Animation | Static Visualization | N/A | Comprehension, Anomalies, Comparison, Construction, Show Evolution | Model-driven, ADL implementation |
| S13 | ARAMIS | Tool | Conceptual | N/A | Static Visualization | Dynamic Visualization, Events Monitoring | Comprehension, Show Evolution, Evaluation | Model-driven |
| S14 | eCITY | Tool | Stable Release | 2D Elements, Color Coding, Animation, Visual Metaphor | Static Visualization | N/A | Comprehension, Comparison, Show Evolution, Tracking | N/A |
| S15 | ARAMIS | Tool, Technique | Prototype | 2D Elements, UML-based | N/A | Dynamic Visualization, Events Monitoring, Live, Post-mortem | Comprehension, Show Evolution, Evaluation | Model-driven, Traceability with requirements, Views creation |
| S16 | eCITY | Tool | Stable Release | 2D Elements, Color Coding, Animation, Visual Metaphor | Static Visualization | N/A | Comprehension, Comparison, Show Evolution, Tracking | N/A |
| S17 | eCITY+ | Tool, Technique | Stable Release | 2D Elements, 3D Visualization, Color Coding, Animation, Visual Metaphor | Static Visualization | N/A | Comprehension, Comparison, Show Evolution, Tracking | As-plugin |
| S18 | ARAMIS | Tool | Stable Release | 2D Elements, UML-based | N/A | Dynamic Visualization, Events Monitoring, Live, Post-mortem | Comprehension, Show Evolution, Evaluation | Model-driven, Traceability with requirements, Views creation, Communication integrity validation |
| S19 | Not named | Description | Project | 2D Elements, UML-based, Color Coding | Static Visualization | N/A | Comprehension, Comparison, Show Evolution | Model-driven, Modernization |
| S20 | Not named | Technique | Conceptual | 2D Elements, UML-based | Static Visualization | N/A | Comprehension, Show Evolution | KAMP approach |
| S21 | EVA | Tool | Stable Release | 2D Elements, 3D Visualization, Color Coding | Static Visualization, Recovery | N/A | Comprehension, Comparison, Show Evolution, Tracking, Rationale | ADD traceability |

alyzed solutions. They have reported the ability to perform visual comparison of SA characteristics among two or more releases of a specific software system, which corresponds to 62% of analyzed visual solutions. The tasks *Anomalies* and *Rationale* were not representative in the solutions, corresponding to only 14% and close to *Styles* with 19%. The task *Construction* explicitly appears in 33% of the visual solutions and half of them present specific resources to design

**Table 12**
Sample Analysis - Visual Solution *EVA* [3]

| Values | Justifications |
|---|---|
| Category = ("Tool") | Title of the paper is *"EVA: A Tool for Visualizing Software Architectural Evolution"*. |
| Stage = ("Stable Released") | The behavior of *EVA* presented in the paper always suggests that it is in production; *EVA* was already evaluated in use and is in the process of deploying to a large development organization. |
| Visualization Form = ("2D Elements", "3D Visualization", "Color Coding") | "2D Elements": suggested by figures displayed in the paper; "3D Visualization": *EVA* provides a view that shows architectures of multiple releases in a 3D visualization; "Color Coding": EVA differs code-level entities through color coding. |
| Static Representation = ("Static Visualization", "Recovery") | "Static Visualization": *EVA* only collects data at compile-time (static elements); "Recovery": *EVA* supports many SA recovery techniques. |
| Dynamic Representation = N/A | The paper does not present any characteristic or feature for dynamic representation. |
| Architectural Tasks = ("Comprehension", "Comparison", "Show Evolution", "Tracking", "Rationale") | "Comprehension": *EVA* provides a view that represents a single SA release, allowing users to interactively understand the functionality of each architecture component; "Comparison": *EVA* provides comparison view that shows the SA diïnĂerences between two releases; "Show Evolution": *EVA* visualizes the SA evolution through a set of source codes across multiple releases; "Tracking": *EVA* provides a view that allows representation of change tracking of entities across multiple releases; "Rationale": *EVA* collects relevant data of design decisions from system repositories, displaying them together with the architecture evolution visualization. |
| Other Features = ("ADD Traceability") | *EVA* shows the traceability of architecture design decisions over time |

the architecture, such as S02 solution [34] (using *Bigraphical Reactive System*) and S06 solution [40] (using an ADL visual modelling component). The task *Evaluation* also appears with 33%.

## 6. Discussion

The specific research question SRQ1 is related to the main visual solutions to support the software architecture evolution comprehension. The answer is presented in Table 11, where it is possible to identify their main goals and characteristics of each solution. The specific research question SRQ2 is concerned to different purposes of using the visual solutions to support the software architecture evolution comprehension. The purposes are also presented in Table 11, identified primarily in the *Architectural Tasks* column and also in the *Static Representation*, *Dynamic Representation* and *Other Features* columns. The specific research question SRQ3 focuses on solutions designed to visually support comprehension of software architecture evolution can be classified. The solutions can be classified in the categories *Description*, *Technique*, *Tool* and *Environment*. The Table 11 classify each solution and shows that the *Tool* category has more representants among the selected studies. The specific research question SRQ4 focuses on visual forms used to support comprehension of software architecture evolution. The Table 11 lists in the *Visualization Form* column how the visual forms have been adopted in the visual solutions discussed in the selected studies. From the list, it is possible to identify the following distribution of use of the visual forms: 2D Elements (20 studies), Tree-based (2 studies), Color coding (9 studies), Visual metaphor (4 studies), 3D Visualization (3 studies), Bigraph (1 study), Uml-based (5 studies) and Animation (5 studies).

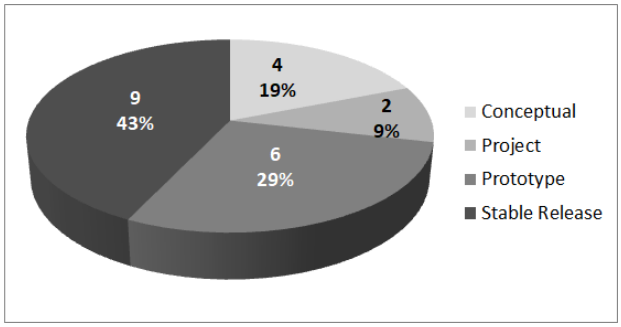


**Figure 9:** Current Stages of Visual Solutions

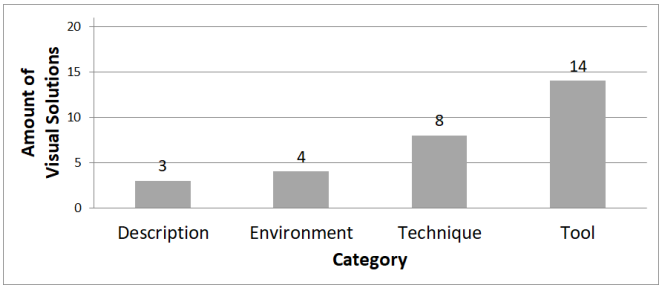


**Figure 10:** Categories of Visual Solutions

Finally, the main research question (RQ) focuses on the evaluation of the usage of visual solutions to support the comprehension of software architecture evolution based on papers published in the peer-reviewed literature. Table 11 presents an up-to-date overview of solutions used for the stated purpose with different characteristics and strategies as has been already explained for the specific research questions. We have identified that based on evidence from the selected studies, features of comprehension and SA evolution visualization are minimum requirements to support software architecture evolution comprehension, as shown in Figure 11. On the other hand, the low number of papers found in

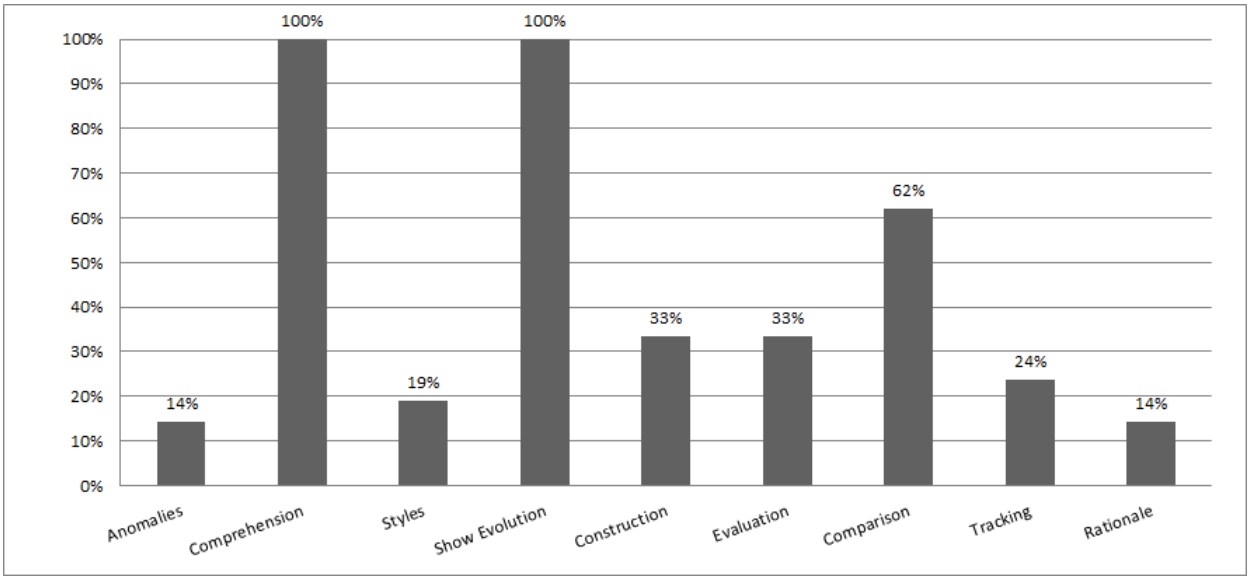


**Figure 11:** Distribution of Architectural Tasks in the Selected Papers

this systematic mapping study, suggests that visual solutions to support SA evolution comprehension is an area that needs expand in terms of studies and options available for practitioners and researchers.

## 7. Conclusion

The aim of this work is to report the design, execution and results of a systematic mapping study of visual solutions to support the comprehension of software architecture evolution. We performed a SMS according to the plan described in Section 4. Initially, the applied search strings retrieved 211 papers from the selected electronic databases. This number was reduced to 21 after applying all selection procedures criteria. From these 21 selected studies, the identified visual solutions to support the comprehension of SA evolution were classified as follows: 14% categorized as *Description*, 38% as *Technique*, 67% as *Tool* and 19% as *Environment*. All of them support the architectural tasks *Comprehension* and *Show Evolution*.

Besides the identification of visual solutions from the literature, another contribution of this study is a taxonomy to classify these solutions. The taxonomy contains six dimensions: category, stage, visualization form, static representation, dynamic representation and architectural tasks. These dimensions and their attributes was explained in Section 5, SubSection 5.1. Each visual solution was classified according to this taxonomy, generating a table of characterization of visual solutions to SA evolution, presented in Table 11. The assignment of the taxonomy attributes value to the visual solution characteristics shows a current overview of the available visual solutions in peer-reviewed literature.

Moreover, this study also shows that visual solutions to support SA evolution comprehension usually present features to support analysis tasks to improve the SA comprehension and also provide facilities to exhibit the SA evolution. This study also concludes that, due to a few number of papers found in this SMS, the studies may consider to allocate more research and development effort to provide effective visual solutions to support SA evolution comprehension, improving the acquired knowledge in this area.

As a future work, we recommend extending the research to establish a methodology or process, based on the taxonomy proposed, to define projects to build visual solutions of SA evolution comprehension. Another possibility for future work is the improvement of the proposed taxonomy, aiming to generate a new framework to objectively evaluate solutions like that ones discussed in this SMS.

## References

[1] H. P. Breivold, I. Crnkovic, M. Larsson, A systematic review of software architecture evolution research, Information and Software Technology 54 (2012) 16 – 40.

[2] L. Yu, S. Ramaswamy, J. Bush, Symbiosis and software evolvability, IT Professional 10 (2008) 56–62.

[3] D. Nam, Y. K. Lee, N. Medvidovic, Eva: A tool for visualizing software architectural evolution, in: 2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion), 2018, pp. 53–56.

[4] Q. Tu, M. W. Godfrey, An integrated approach for studying architectural evolution, in: Proceedings 10th International Workshop on Program Comprehension, 2002, pp. 127–136.

[5] M. Lungu, M. Lanza, Exploring inter-module relationships in evolving software systems, in: 11th European Conference on Software Maintenance and Reengineering (CSMR'07), 2007, pp. 91–102.

[6] R. Taylor, N. Medvidovic, E. Dashofy, Software Architecture: Foundations, Theory and Practice, Hoboken, New Jersey, 2009.

[7] D. Garlan, J. M. Barnes, B. Schmerl, O. Celiku, Evolution styles: Foundations and tool support for software architecture evolution, in: 2009 Joint Working IEEE/IFIP Conference on Software Architecture European Conference on Software Architecture, 2009, pp. 131–140.

[8] M. Shahin, P. Liang, M. A. Babar, A systematic review of software architecture visualization techniques, Journal of Systems and Software 94 (2014) 161 – 185.

[9] M. Shahin, P. Liang, M. R. Khayyambashi, Improving understand-

ability of architecture design through visualization of architectural design decision, in: Proceedings of the 2010 ICSE Workshop on Sharing and Reusing Architectural Knowledge, SHARK '10, 2010, pp. 88–95.

[10] R. L. Novais, M. G. de Mendonca Neto, Computer Systems and Software Engineering: Concepts, Methodologies, Tools, and Applications, Chapter: Software Evolution Visualization: Status, Challenges, and Research Directions, IGI Global, 2018.

[11] Y. Ghanam, S. Carpendale, A survey paper on software architecture visualization Technical report (2008).

[12] B. A. Price, R. M. Baecker, I. S. Small, A principled taxonomy of software visualization, Journal of Visual Languages Computing 4 (1993) 211 – 266.

[13] J. Werther, G. Carneiro, R. Maciel, A systematic mapping on visual solutions to support the comprehension of software architecture evolution, in: Proceedings of the 25th International DMS Conference on Visualization and Visual Languages, DMSVIVA '19, 2019, pp. 63–74. doi:10.18293/DMSVIVA2019-008.

[14] D. Garlan, D. E. Perry, Introduction to the special issue on software architecture, IEEE Transactions on Software Engineering 21 (1995) 269–274.

[15] D. Garlan, Software architecture: A travelogue, in: Proceedings of the on Future of Software Engineering, FOSE 2014, 2014, pp. 29–39.

[16] D. Garlan, Software architecture: a roadmap, in: Proceedings of Conference on the Future of Software Engineering, Limerick, Ireland, 2000, pp. 91–101.

[17] N. Medvidovic, R. Taylor, D. Rosenblum, An architecture-based approach to software evolution, in: Proceedings of International Workshop on the Principles of Software Evolution, 1998.

[18] L. A. Belady, M. M. Lehman, A model of large program development, IBM Systems journal 15 (1976) 225–252.

[19] C. F. Kemerer, S. Slaughter, An empirical approach to studying software evolution, IEEE Transactions on Software Engineering 25 (1999) 493–509.

[20] D. Rowe, J. Leaney, D. Lowe, Defining systems evolvability - a taxonomy of change, in: Proceedings of International Conference and Workshops on Engineering of Computer-Based Systems (ECBS), 1998, Jerusalem, Israel, 1998.

[21] E. Gamma, et al., Design patterns: elements of reusable object-oriented software, Addison-Wesley, 1995.

[22] K. Gallagher, A. Hatch, M. Munro, Software architecture visualization: An evaluation framework and its application, IEEE Transactions on Software Engineering 34 (2008) 260–270.

[23] J. Cleland-Huang, R. S. Hanmer, S. Supakkul, M. Mirakhorli, The twin peaks of requirements and architecture, IEEE Software 30 (2013) 24–29.

[24] V. Basili, G. Caldiera, H. Rombach, The goal question metric paradigm, Encyclopedia of Software Engineering 2 (1994) 528–532.

[25] A. Telea, L. Voinea, H. Sassenburg, Visual tools for software architecture understanding: A stakeholder perspective, IEEE Software 27 (2010) 46–53.

[26] M. Sulir, M. Bacikova, S. Chodarev, J. Poruban, Visual augmentation of source code editors: A systematic mapping study, Journal of Visual Languages Computing 49 (2018) 46 – 59.

[27] C. Wohlin, et al., Experimentation in Software Engineering, Springer-Verlag, 2012.

[28] V. R. Basili, H. D. Rombach, The tame project: towards improvement-oriented software environments, IEEE Transactions on Software Engineering 14 (1988) 758–773.

[29] T. Dyba, T. Dingsoyr, Empirical studies of agile software development: A systematic review, Information and Software Technology 50 (2008) 833 – 859.

[30] D. Moher, A. Liberati, J. Tetzlaff, D. G. Altman, P. Group, et al., Preferred reporting items for systematic reviews and meta-analyses: the prisma statement, PLoS medicine 6 (2009) e1000097.

[31] M. Jazayeri, On architectural stability and evolution, in: J. Blieberger, A. Strohmeier (Eds.), Reliable Software Technologies — Ada-Europe 2002, Springer Berlin Heidelberg, Berlin, Heidelberg, 2002, pp. 13–23.

[32] D. Hirsch, U. Montanari, Two graph-based techniques for software architecture reconfiguration, Electronic Notes in Theoretical Computer Science 51 (2002) 177–190.

[33] K. Dunsire, T. O'Neill, M. Denford, J. Leaney, The abacus architectural approach to computer-based system and enterprise evolution, in: 12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'05), 2005, pp. 62–69. doi:10.1109/ECBS.2005.66.

[34] Z. Chang, X. Mao, Z. Qi, An approach based on bigraphical reactive systems to check architectural instance conforming to its style, in: First Joint IEEE/IFIP Symposium on Theoretical Aspects of Software Engineering (TASE '07), 2007, pp. 57–66.

[35] W. C. Stratton, D. E. Sibol, M. Lindvall, P. Costa, Technology infusion of save into the ground software development process for nasa missions at jhu/apl, in: 2007 IEEE Aerospace Conference, 2007, pp. 1–15. doi:10.1109/AERO.2007.352763.

[36] W. C. Stratton, D. E. Sibol, M. Lindvall, P. Costa, The save tool and process applied to ground software development at jhu/apl: An experience report on technology infusion, in: 31st IEEE Software Engineering Workshop (SEW 2007), 2007, pp. 187–193.

[37] A. McNair, D. M. German, J. Weber-Jahnke, Visualizing software architecture evolution using change-sets, in: 14th Working Conference on Reverse Engineering (WCRE 2007), IEEE, 2007, pp. 130–139.

[38] A. Hindle, Z. M. Jiang, W. Koleilat, M. W. Godfrey, R. C. Holt, Yarn: Animating software evolution, in: 2007 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis, IEEE, 2007, pp. 129–136.

[39] L. Schrettner, P. Hegedus, R. Ferenc, L. J. Fulop, T. Bakota, Development of a methodology, software – suite and service for supporting software architecture reconstruction, in: 2010 14th European Conference on Software Maintenance and Reengineering, 2010, pp. 190–193.

[40] A. McVeigh, J. Kramer, J. Magee, Evolve: Tool support for architecture evolution, in: Proceedings of the 33rd International Conference on Software Engineering, ICSE '11, 2011, pp. 1040–1042.

[41] A. Dragomir, H. Lichter, Model-based software architecture evolution and evaluation, in: 2012 19th Asia-Pacific Software Engineering Conference, volume 1, 2012, pp. 697–700.

[42] T. Khan, H. Barthel, A. Ebert, P. Liggesmeyer, ecity: A tool to track software structural changes using an evolving city, in: 2013 IEEE International Conference on Software Maintenance, 2013, pp. 492–495. doi:10.1109/ICSM.2013.80.

[43] A. Dragomir, H. Lichter, Run-time monitoring and real-time visualization of software architectures, in: 2013 20th Asia-Pacific Software Engineering Conference (APSEC), volume 1, 2013, pp. 396–403.

[44] T. Khan, H. Barthel, L. Guzman, A. Ebert, P. Liggesmeyer, ecity: Evolutionary software architecture visualization–an evaluation, in: Building Bridges: HCI, Visualization, and Non-formal Modeling, Springer, 2014, pp. 201–224.

[45] T. Khan, S. R. Humayoun, K. Amrhein, H. Barthel, A. Ebert, P. Liggesmeyer, ecity+: A tool to analyze software architectural relations through interactive visual support, in: Proceedings of the 2014 European Conference on Software Architecture Workshops, ECSAW '14, 2014, pp. 36:1–36:4.

[46] A. Nicolaescu, H. Lichter, A. Göringer, P. Alexander, D. Le, The aramis workbench for monitoring, analysis and visualization of architectures based on run-time interactions, in: Proceedings of the 2015 European Conference on Software Architecture Workshops, ACM, 2015, p. 57.

[47] D. Escobar, D. CÃąrdenas, R. Amarillo, E. Castro, K. GarcÃĺs, C. Parra, R. Casallas, Towards the understanding and evolution of monolithic applications as microservices, in: 2016 XLII Latin American Computing Conference (CLEI), 2016, pp. 1–11.

[48] D. Faitelson, R. Heinrich, S. S. Tyszberowicz, Supporting software architecture evolution by functional decomposition., in: MODELSWARD, 2017, pp. 435–442.

[49] D. Faitelson, S. Tyszberowicz, Improving design decomposition, in:

International Symposium on Dependable Software Engineering: Theories, Tools, and Applications, Springer, 2015, pp. 185–200.

[50] K. Rostami, J. Stammel, R. Heinrich, R. Reussner, Architecture-based assessment and planning of change requests, in: Proceedings of the 11th International ACM SIGSOFT Conference on Quality of Software Architectures, ACM, 2015, pp. 21–30.