The Journal of Systems and Software 94 (2014) 161–185

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

A systematic review of software architecture visualization techniques

Mojtaba Shahin[a,b], Peng Liang[a,c,*], Muhammad Ali Babar[d]

[a] State Key Lab of Software Engineering, School of Computer, Wuhan University, China
[b] Department of Computer Engineering, Neyriz Branch, Islamic Azad University, Iran
[c] Department of Computer Science, VU University Amsterdam, Netherlands
[d] CREST – The Centre for Research on Engineering Software Technologies, The University of Adelaide, Australia

ARTICLE INFO

Article history:
Received 31 August 2013
Received in revised form 4 February 2014
Accepted 22 March 2014
Available online 30 March 2014

Keywords:
Software architecture
Software architecture visualization
Visualization techniques

ABSTRACT

Context: Given the increased interest in using visualization techniques (VTs) to help communicate and understand software architecture (SA) of large scale complex systems, several VTs and tools have been reported to represent architectural elements (such as architecture design, architectural patterns, and architectural design decisions). However, there is no attempt to systematically review and classify the VTs and associated tools reported for SA, and how they have been assessed and applied.
Objective: This work aimed at systematically reviewing the literature on software architecture visualization to develop a classification of VTs in SA, analyze the level of reported evidence and the use of different VTs for representing SA in different application domains, and identify the gaps for future research in the area.
Method: We used systematic literature review (SLR) method of the evidence-based software engineering (EBSE) for reviewing the literature on VTs for SA. We used both manual and automatic search strategies for searching the relevant papers published between 1 February 1999 and 1 July 2011.
Results: We selected 53 papers from the initially retrieved 23,056 articles for data extraction, analysis, and synthesis based on pre-defined inclusion and exclusion criteria. The results from the data analysis enabled us to classify the identified VTs into four types based on the usage popularity: graph-based, notation-based, matrix-based, and metaphor-based VTs. The VTs in SA are mostly used for architecture recovery and architectural evolution activities. We have also identified ten purposes of using VTs in SA. Our results also revealed that VTs in SA have been applied to a wide range of application domains, among which "graphics software" and "distributed system" have received the most attention.
Conclusion: SA visualization has gained significant importance in understanding and evolving software-intensive systems. However, only a few VTs have been employed in industrial practice. This review has enabled us to identify the following areas for further research and improvement: (i) it is necessary to perform more research on applying visualization techniques in architectural analysis, architectural synthesis, architectural implementation, and architecture reuse activities; (ii) it is essential to pay more attention to use more objective evaluation methods (e.g., controlled experiment) for providing more convincing evidence to support the promised benefits of using VTs in SA; (iii) it is important to conduct industrial surveys for investigating how software architecture practitioners actually employ VTs in architecting process and what are the issues that hinder and prevent them from adopting VTs in SA.

1. Introduction

With increasing size and complexity of software-intensive systems, the role of software architecture (SA) as a means of understanding and managing large-scale software intensive systems has been increasingly becoming important. The high level design description of a large system can help a system's stakeholders to understand and reason about the designed architecture with regards to architecturally significant requirements (ASRs) of a software-intensive system (Bass et al., 2012). SA community has been developing various approaches, techniques, and tools for improving software architecture communication and understanding among all the key stakeholders of large-scale software-intensive systems. One of the increasingly popular ways of making software architecture design decisions and their

* Corresponding author at: State Key Lab of Software Engineering, School of Computer, Wuhan University, China. Tel.: +86 27 68776137; fax: +86 27 68776027.
E-mail address: liangp@whu.edu.cn (P. Liang).

http://dx.doi.org/10.1016/j.jss.2014.03.071
0164-1212/

rationale easily understandable is visualizing SA (Shahin and Liang, 2010; de Boer et al., 2009; López et al., 2009; Lee and Kruchten, 2008). Visualization in computer graphics is a technique for creating images, diagrams, or animations to communicate information, which may not be easy to describe and understand in other formats, such as textual (Spence, 2000). Visualization transfers information into visual forms and enhances information understanding in software development (Diehl, 2007).

Software visualization is defined as visual representation of artifacts (such as requirements, design, and program code) related to software and its development process (Diehl, 2007). The main motivation for using software visualization is to help stakeholders to understand and comprehend different aspects of software systems during software development process and reduce the cost of software evolution (Diehl, 2007; Gallagher et al., 2008). SA visualization is defined as a visual representation of architectural models and some or all of the architectural design decisions about the models (Taylor et al., 2009). The importance of visualizing SA has been extensively investigated as SA visualization can be of interest to various stakeholders such as architects, developers, testers, and project managers (Gallagher et al., 2008; Sharafi, 2011; Telea et al., 2010; Shahin and Liang, 2010). SA visualization, e.g., decomposition of a software system's architecture into layers, components, or slices in a structural viewpoint, is critical in understanding and communicating the architecture to a variety of project stakeholders (Cleland-Huang et al., 2013). Due to the recognition of the importance of visualizing SA, an increasing amount of literature describing SA visualization approaches and tools has been published through diverse venues (Gallagher et al., 2008; Sharafi, 2011; Telea et al., 2010). However, there has been no dedicated effort to systematically identify and select, and rigorously analyze and synthesize the SA visualization literature. In order to fill this gap, we decided to carry out a systematic literature review (SLR) (Kitchenham and Charters, 2007) of the SA visualization.

This article reports the design, execution, and findings of the SLR that aimed at systematically identifying, selecting, and summarizing a comprehensive set of SA visualization techniques, associated tools, and the supporting evidence published in the peer-reviewed literature. This SLR enabled us to enumerate a comprehensive set of papers describing SA visualization techniques and tools in order to reveal the key motivators for their development, their evolutionary paths, foundational principles, and assessment mechanisms. For this review, we have systematically identified and rigorously reviewed 53 relevant papers and synthesized the data extracted from those papers in order to answer a set of research questions that had motivated this review. We assert that the results from this SLR can provide important benefits to researchers and practitioners from both software architecture as well as software visualization communities. This review can enable them to gain a better understanding of the available SA visualization techniques, their suitability for different architecting activities, the level of evidence reported for each of them, and the gaps that need further research in this area. The two significant contributions of this paper to the software architecture visualization body of knowledge are:

1. It reports the design, execution, and results of a review aimed at systematically identifying a comprehensive set of relevant papers on SA visualization techniques based on pre-defined selection criteria and rigorously analyzing and synthesizing the reported techniques, associate tools, and reported evidence in an easily accessible format.
2. It structures and classifies the reviewed SA visualization techniques and tools, and the available evidence using different formats that are expected to be useful for practitioners interested in using visualization for communicating and understanding SA design and design decisions. The findings can be used as an evidence-based guide to select appropriate SA visualization techniques and tools based on the required suitability for different activities of the software architecting process. The findings also identify the issues relevant to researchers who are interested in knowing the state-of-the-art of and the areas of future research in SA visualization.

1.1. Background and related work

Software architecture has emerged as an important area of software engineering research and practice over the last two decades. The increasing size of, complexity of, and demand for quality in software systems are some of the most important factors that have resulted in sustained interests in SA research and practice. It is widely recognized that a high level design description can play an important role in successfully understanding and managing large and complex software systems (Bass et al., 2012). SA community has developed several methods, approaches, and tools to help understand and reason about high level architecture design. Software architecture can be described and viewed from multiple perspectives. Two of the most commonly used perspectives of SA are architectural viewpoint and architecting process perspective.

There are two distinct viewpoints on SA, structural and decisional (Poort and van Vliet, 2012): the structural viewpoint expresses SA with components and connectors and considers it as a high-level software structure of a system (Bass et al., 2012). This viewpoint mainly focuses on the end products (e.g., components and connectors) of software architecting process. The decisional viewpoint considers decisions made during architecting as the first class entities and defines SA as a set of design decisions, including their rationale (Jansen and Bosch, 2005). In this SLR, both viewpoints of SA have been considered as visualization techniques can be used to support both kinds of viewpoints of SA. The structural elements (e.g., components and connectors) and decisional elements (e.g., decisions) are generally termed as "architectural elements" or "architectural entities" that are interchangeably used in this paper.

Architecting is a process of conceiving, defining, expressing, documenting, communicating, certifying proper implementation of, maintaining and improving an architecture throughout a system's life cycle (ISo, 2011). From an architecting process perspective, software architecting is composed of a set of general and specific activities (Li et al., 2013; Hofmeister et al., 2007), which can be supported by various visualization techniques and tools. The specific architecting activities cover the entire architecture lifecycle and the general architecting activities provide support to achieve the goals of the specific activities of software architecting. For example, architectural evolution, as a specific architecting activity, copes with correcting faults, responding to new changes, and implementing new requirements in architecture. Architecture recovery, as a general architecting activity, examines existing available sources of a system (such as implementation and documentation of a system) to uncover and extract architecture design and design decisions. Architecture recovery can support architecture evolution by recovering the architecture design and design decisions when architecture documentation is not well documented or unavailable, or architectural design decisions have been lost.

Through this review, we are interested to know how various visualization techniques can facilitate these general and specific architecting activities. It is generally considered that SA visualization techniques can be used to support any stage of the software architecting process, i.e., analyzing, synthesizing, evaluating, implementing, and evolving architecture (Telea et al., 2010). In a decisional viewpoint, visualization of architectural design decisions (ADDs) can improve the understanding of ADDs and their rationale, and this kind of understanding becomes more

important in a collaborative and distributed development environment as it supports better communication of ADDs made during software architecting (Shahin and Liang, 2010). To the best of our knowledge, there have only been three previous secondary studies on VTs for SA, however, none of them was aimed at using a systematic approach (i.e., SLR) to carry out the reported review studies. We provide brief overviews of these studies as the related work here.

Gallagher and his colleagues have provided a criteria-based framework to evaluate six architecture visualization tools (Gallagher et al., 2008). The framework is composed of 31 criteria and each criterion might be relevant to one or some stakeholders of SA (such as developer, maintainer, architect, and tester). None of existing six tools can support all criteria, but the framework can be employed as a template to develop comprehensive SA visualization tools. The framework also suggests the areas that were open for further research in the existing six tools. Sharfi proposed a methodology to evaluate the effectiveness and usability of architecture visualization techniques (Sharafi, 2011). To achieve this, the methodology starts with classifying the visualization techniques based on different audiences of architecture visualization and their goals. The audiences include both researchers who are interested in new software visualization techniques and developers who are interested in better understanding of a system using visualization techniques. Then the methodology determines what functional and non-functional requirements are needed for architecture visualizations. Telea and colleagues conducted a review and evaluation of architecture understanding visualization tools and techniques from a stakeholder perspective (Telea et al., 2010). The perception of the added value of a visualization tool for stakeholders is essential to show the usefulness of that tool. To achieve this, Telea *et al.* (Telea et al., 2010) distinguish three stakeholder types (i.e., technical user, project manager, and consultant) and determine what functionalities these stakeholders expect from architecture visualization tools. They conclude that the existing architecture visualization tools are more suitable for technical users and less suitable for consultants (Telea et al., 2010).

The remainder of the paper is structured as follows. Section 2 describes the systematic literature review method used in this work and defines the review protocol. Section 3 presents the demographic information and the quality assessment of the included studies, and answers the research questions through the analysis of selected studies. The threats to validity are discussed in Section 4 with further discussion of the review results in Section 5. Finally, we present our conclusions in Section 6.

2. Research method

As previously stated, we used Systematic Literature Review (SLR) that is one of the most widely used research methods of Evidence-Based Software Engineering (EBSE). The SLR research method provides a well-defined process for identifying, evaluating, and interpreting all available evidence relevant to a particular research question or topic (Kitchenham and Charters, 2007). A SLR evaluates existing studies on a specific phenomenon fairly and creditably. For this review, we followed Kitchenham and Charters's guidelines (Kitchenham and Charters, 2007) that involve three main phases: defining a review protocol, conducting the review, and reporting the review. Our review protocol consists of these elements: (i) research questions, (ii) search strategy, (iii) inclusion and exclusion criteria, (iv) study selection, (v) study quality assessment, and (vi) data extraction and synthesis. These steps are further discussed in the following subsections.

2.1. Research questions

This SLR aimed to summarize and provide an overview of the current research on "*what software architecture visualization techniques have been reported in the peer-reviewed literature*?" To obtain this goal, we formulated a set of research questions (RQs) to be answered through this SLR. Table 1 shows the research questions as well as the motivations we asked them. The answers to these research questions can be directly linked to the objective of this SLR: an understanding of the available SA visualization techniques in literature and their industrial applications (RQ1.1, RQ1.3, RQ4, and RQ5), their suitability for various architecting activities (RQ1.2 and RQ1.2.1), the purposes of using them in SA (RQ3), and the level of evidence reported for each of them (RQ2). Investigation of these research questions can provide a systematic overview of SA visualization techniques which is beneficial for researchers to identify the missing gap in this area and practitioners to use evidence-based best practices of SA visualization.

Table 1
Research questions of this SLR.

Research question	Motivation
RQ1: What visualization techniques are used in software architecture?	This research question has been broken down into four sub-RQs (RQ1.1, RQ1.2, RQ1.2.1, and RQ1.3)
RQ1.1: Which types of visualization techniques are used in architecting process?	To get an overview of different types of visualization techniques used in various architecting activities, and identify the visualization techniques that is mostly used and least used in the literature.
RQ1.2: Which architecting activities are supported by visualization techniques?	To gain an understanding of the distribution and the gaps of the application of visualization techniques over various architecting activities.
RQ1.2.1: What is the relationship between different visualization techniques and architecting activities?	To investigate whether there is a correlation between visualization techniques and architecting activities, for example, a specific type of visualization technique is most frequently employed in an architecting activity.
RQ1.3: What architecture visualization tools are available to support visualization techniques?	To investigate whether tool support is available for a proposed visualization technique since tool support is critical for practical applicability of the visualization technique.
RQ2: How much evidence is available to support the adoption of different visualization techniques in software architecture?	To gain an understanding of the maturity of proposed visualization techniques in software architecture. This research question is of interest for both practitioners and researchers when they want to adopt or further evaluate existing visualization techniques. The maturity of the visualization techniques is measured based on the evidence levels described in data item D10 in Section 2.5
RQ3: What are the different purposes of using visualization techniques in software architecture?	To identify the major purposes and merits of applying visualization techniques in software architecture, which are especially useful for practitioners to know what they can do with SA visualization.
RQ4: What are the domains in which architecture visualization techniques have been applied to support architecting activities?	The use of architecture visualization techniques in architecting activities may take place in various application domains. Through this research question, we want to know the application domains in which the architecture visualization techniques have been applied and how often the architecture visualization techniques have been applied to each domain. This information can help practitioners to identify the application domains that have gained more interest in applying architecture visualization.
RQ5: Which are the architecture visualization techniques mostly used in industry?	To understand which architecture visualization techniques are more popularly used in industry, so that industrial practitioners can consider them with a high priority. This research question identifies the application domains that receive more attention in applying architecture visualization techniques in industry.

Table 2
Electronic databases for the automatic search included in this SLR.

#	Electronic Database	Search terms are matched with	Web address
ED1	ACM Digital Library	Paper title, keywords	http://portal.acm.org
ED2	IEEE Explore	Paper title, keywords, abstract	http://www.ieee.org/web/publications/xplore
ED3	ScienceDirect	Paper title, keywords, abstract	http://www.elsevier.com
ED4	SpringerLink	Paper title, keywords	http://www.springerlink.com
ED5	Wiley InterScience	Paper title, keywords	http://www3.interscience.wiley.com
ED6	EI Compendex	Paper title, keywords, abstract	http://www.engineeringvillage.com
ED7	ISI Web of Science	Paper title, keywords, abstract	http://www.webofknowledge.com

2.2. Search strategy

For a SLR, it is an important pre-requisite to define a search strategy that can help researchers to retrieve as many relevant studies as possible (Kitchenham and Charters, 2007; Zhang et al., 2011). The search strategy used for this review was composed of the following elements: search methods, search terms, and data sources. The search starts with following initial steps in order to identify the relevant data sources and search terms:

a. Preliminary search to retrieve existing reviews and potential relevant studies (e.g., Gallagher et al., 2008; Sharafi, 2011; Telea et al., 2010).
b. Check the research results published in leading software architecture conferences and workshops such as WICSA, ECSA, QoSA, and SHARK.
c. Trial search using various combinations of the initial search terms derived from the research questions.
d. Consult with the experts and authors who have experiences in applying the visualization techniques in SA to get related sources (e.g., conferences and journals) and literature on software architecture visualization.
e. Use our own experience and knowledge on visualization of SA.

2.2.1. Search method

Our search strategy used two search methods: manual search and automatic search. These search methods are complementary to each other (Zhang et al., 2011). With the manual search, we checked all of the papers published in the venues enlisted in Table 3. For the automatic search, we searched the electronic data sources listed in Table 2 using the search terms mentioned in Section 2.2.2.

2.2.2. Search terms

The search terms are used to match with paper titles, keywords, and abstracts in the electronic data sources during the automatic search, and the exceptional cases are ACM, Springer, and Wiley databases (see Table 2) in which search terms are only matched with paper titles and keywords since these databases return an excessively large number of papers when including abstracts. According to the guidelines provided in (Kitchenham and Charters, 2007), we used the following strategies for forming the most relevant search terms for the automatic search:

1. Derive major terms according to the study topic and research questions.
2. List the keywords mentioned in the articles (primary studies) we already knew about.
3. Identify synonyms and related terms for major terms.
4. Use the Boolean "AND" to join major terms.
5. Use the Boolean "OR" to join alternative terms and synonyms.

The resulting search terms are composed of the synonyms and related terms about "architecture" AND "visualization". We used the following search terms:

(architecture **OR** architectural **OR** architecting **OR** structure) **AND** (visual **OR** visualize **OR** visualization **OR** visualizing **OR** diagram **OR** picture **OR** graphic **OR** graphical)

It is worth noting that in the initial set of search terms, we also used "model", "modeling", "representation", and "illustration" as related terms to "visualization", but we found that these terms were too general in architecture studies and led to many irrelevant search results during the trial searches and it was not feasible for checking them all. Therefore, we decided to exclude these terms from the final search terms. The search terms also did not include "software" for two reasons: (1) many studies on software architecture did not really use "software" in the title, especially in venues like WICSA, ECSA, and SHARK because authors assumed that the audiences are from the "software" architecture community. For example, most architectural knowledge researchers talk about "architectural knowledge" instead of "software architectural knowledge". (2) It was meant to avoid the situations where relevant studies may not be retrieved because authors were targeting system level architecture therefore they did not use "software" in the title.

2.2.3. Data sources

Table 2 shows that we used seven electronic databases. These are considered the primary sources of literature for potentially relevant studies on software architecture, software visualization, and software engineering in general (Chen and Babar, 2010). We did not include Google Scholar in the data sources since it could produce search results with low precision, generate many irrelevant results, and have considerable overlap with ACM and IEEE on software engineering literature (Chen and Babar, 2010). Table 3 enlists relevant and representative journals, conferences, and workshops on software architecture and software visualization, which publish high quality papers. The selection of journals, conferences, and workshops for the manual search may not be comprehensive since we regarded the manual search as a supplementary, but not exhaustive, source to the database (automatic) search. The conferences and workshops such as C9, C10, C11, and W1 are considered high quality and most relevant venues for software architecture literature. The journal such as J7 and the conferences such as C13, C16, and C19 and the workshop such as W3 are the venues relevant for the literature on visualization techniques in software domain. Other journals, conferences, and workshops are related to the general topics of software engineering in which 10 of them are considered as leading venues in software engineering (Sjoberg et al., 2005).

2.3. Inclusion and exclusion criteria

The inclusion and exclusion criteria are used for selecting relevant primary studies to answer the research questions in a SLR.

Table 3
Target venues for the manual search included in this SLR.

#	Venue	Acronym
J1	Information and Software Technology	IST
J2	Journal of Systems and Software	JSS
J3	IEEE Transactions on Software Engineering	TSE
J4	IEEE Software	IEEE SW
J5	ACM Computer Surveys	CSUR
J6	ACM Transactions on Software Engineering and Methodologies	TOSEM
J7	IEEE Transactions on Visualization and Computer Graphics	TVCG
J8	Software: Practice and Experience	SPE
J9	Journal of Software Maintenance and Evolution: Research and Practice	SMPR
J10	Empirical Software Engineering Journal	ESE
J12	Software and Systems Modeling	SoSyM
J11	IEE Proceedings Software (now IET Software)	IET Software
C1	ACM/IEEE International Conference on Software Engineering	ICSE
C2	ACM International Symposium on Foundations of Software Engineering	FSE
C3	IEEE/ACM International Conference on Automated Software Engineering	ASE
C4	Working Conference on Reverse Engineering	WCRE
C5	IEEE International Conference on Software Maintenance	ICSM
C6	International Symposium on Empirical Software Engineering and Measurement	ESEM
C7	Australian Conference on Software Engineering	ASWEC
C8	Asia-Pacific Software Engineering Conference	APSEC
C9	Working IEEE/IFIP Conference on Software Architecture	WICSA
C10	Conference on the Quality of Software Architectures	QoSA
C11	European Conference on Software Architectures	ECSA
C12	European Conference on Software Maintenance and Reengineering	CSMR
C13	IEEE/ACM Symposium on Software Visualization	SoftVis
C14	Annual International Computer Software and Applications Conference	COMPSAC
C15	International Conference on Quality Software	QSIC
C16	IEEE Symposium on Visual Languages and Human-Centric Computing	VLHCC
C17	International Workshop/Conference on Program Comprehension	IWPC
C18	International Conference on Engineering of Complex Computer Systems	ICECCS
C19	IEEE Pacific Visualization Symposium	PacificVis
W1	Workshop on Sharing and Reusing Architectural Knowledge Architecture	SHARK
W2	IEEE International Software Engineering Workshop	SEW
W3	IEEE International Workshop on Visualizing Software for Understanding and Analysis	VISSOFT

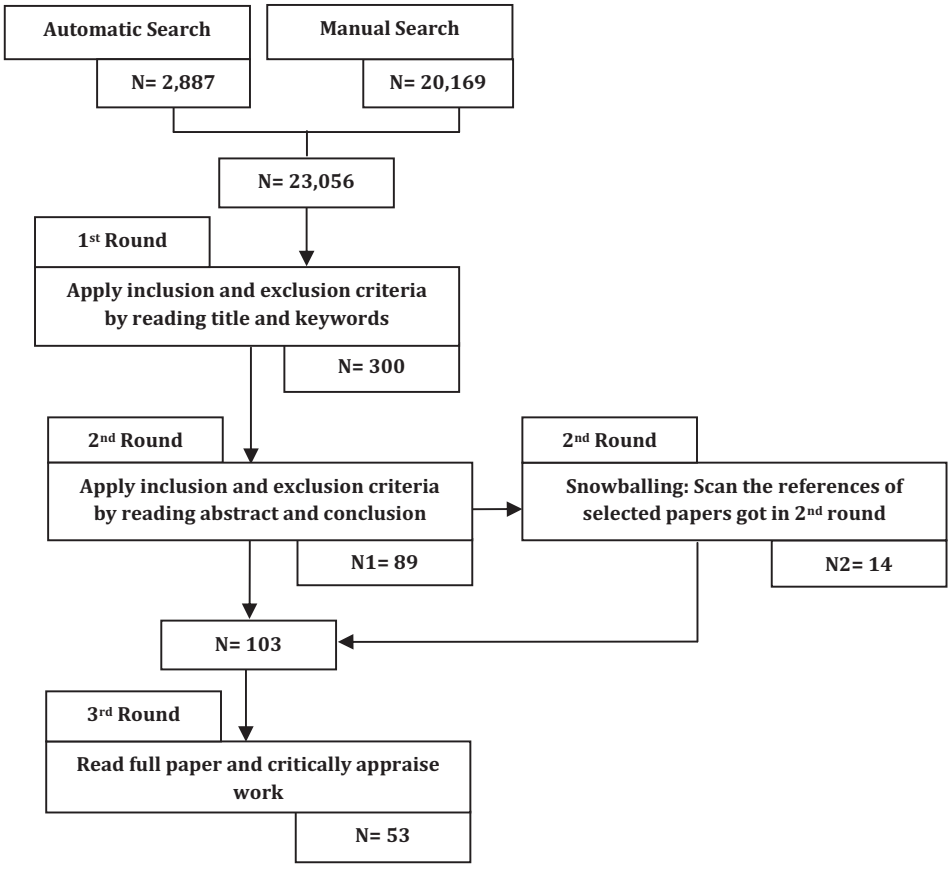


Fig. 1. Stages of the search process and number of selected studies in each stage.

Table 4
Inclusion and exclusion criteria of this SLR.

Inclusion criteria	
I1	A study that is peer-reviewed and available in full-text
I2	A study that introduces a visualization technique used in software architecture
Exclusion criteria	
E1	Editorials, position papers, keynotes, reviews, tutorial summaries, and panel discussions
E2	A study that is not written in English
E3	A study that introduces visualization techniques on other software development activities instead of software architecting (e.g., program comprehension)
E4	Duplicated studies
E5	A study that has no validation (i.e., no evidence)

The inclusion and exclusion criteria are applied to all retrieved studies from databases and target venues at different steps (see Fig. 1). The inclusion and exclusion criteria used in our SLR are specified in Table 4. We limited the time period of our search to studies published between 1 February 1999 and 1 July 2011 because the first flagship software architecture conference WICSA (Working IEEE/IFIP Conference on Software Architecture) was held in 1999. The selected studies should have case studies or empirical data that can validate the proposed techniques and the papers that only present approaches or research plan without any validations should be excluded. The evidence levels are used to show how a study validates its proposed visualization techniques. One of our inclusion criteria was that a study should have at least provided a so-called "toy example/demonstration" to support the claims made for the presented technique and tool (see Section 2.5). If two papers addressed the same topic and were published in different venues (e.g., in a conference and a journal respectively), the less mature one is excluded. All duplicated studies found from different sources were to be detected and removed.

2.4. Study selection

The number of studies selected at each stage of this SLR has been shown in Fig. 1. During the search phase, we retrieved 2887 and 20,169 papers through automatic and manual searches respectively. Because a large number of papers were retrieved during this step, we performed the first round of paper selection when conducting the manual and automatic searches. When there were some papers that we could not decide by reading the titles and keywords, these papers were retained to the next round of selection for further investigation. At the end of the second step, we found that 89 papers met all the inclusion criteria. Then we followed a snowballing technique (Budgen et al., 2008) to scan the references of these 89 selected papers in order to find more potential primary studies. We found 22 potentially relevant papers by title from the references of these 89 papers. We applied the inclusion and exclusion criteria on the abstracts and conclusions of those 22 potentially relevant papers and selected 14 papers for the next phase. It is worth noting that we excluded the studies published before 1 February 1999 got by snowballing technique. We also recorded the reasons of inclusion or exclusion decision for each of the papers, which were used for further discussion and reassessment whether a paper was to be included or not. The selection results in the second round were verified by two of the authors and any disagreements were resolved. Like the first round of selection, when there were some papers that we could not decide through the abstracts and conclusions, these papers were retained for the next round of selection. In the last (third) round of selection, if a paper satisfied all the inclusion criteria, this paper was considered as a primary study to be included in this SLR. Finally, we selected 53 papers for this review.

2.5. Data extraction and synthesis

The data extraction step of SLR purports to identify the relevant information that should be extracted from each of the primary studies in order to answer the research questions of a SLR. Table 5 presents the data items for which the relevant information was extracted from the selected papers. It also shows the research question(s) (described in Section 2.1) that were supposed to be answered using different pieces of the extracted data. A record of the extracted information from a study was stored in a spreadsheet for subsequent analysis. Some of the extracted data items are described below:

- D10 evaluates the evidence level of the reported visualization technique. The obtained results can help researchers to assess the maturity of a visualization technique. We used a five level scheme for assessing the reported evidence as proposed in (Alves et al., 2010). These evidence levels are listed below:

1. Toy examples/demonstration: Evidence obtained from demonstration or working out with toy examples.
2. Expert opinions/observations: Evidence obtained from expert opinions or observations.
3. Academic studies: Evidence obtained from academic studies (e.g., controlled lab experiments)
4. Industrial studies: Evidence obtained from industrial studies (e.g., causal case studies)
5. Industrial practice: Evidence obtained from industrial practice. According to (Alves et al., 2010), "this evidence level indicates that a method has already been approved and adopted by an industrial organization".

- D12 records the level of tool support, which is classified as automatic, semi-automatic, and manual. Automatic refers to the tool support level when human intervention is not necessary or minimal for using a particular visualization technique. If partial human intervention is needed, the tool level will be considered as semi-automatic. If there is no tool support or using the reported visualization technique completely depends on human intervention, the level is rated as manual. The level of tool support is obtained based on the description of the underlying tool support in the study or by referring to the relevant tool website.
- D13 documents the type of applications used as a case study with aim of validating the reported visualization technique. For example, a type of application can be an embedded system, a distributed system, or an information management system. This type of information can help practitioners to identify and understand the domain specific implications of a particular technique for visualizing architecture before deciding to use that technique.

3. Results

In following subsections, we are going to report the results from the synthesis and analysis of the data extracted from the primary studies to answer the research questions. Most of the results presented in this section are based on synthesizing the data directly collected from the reviewed studies. We have limited our own interpretations of the results to a minimum in order to mainly keep our focus on what has been reported in the reviewed primary studies. However, some of the reviewed primary studies might not have provided enough information for answering some RQs, hence, interpretations and inferences to some extent were unavoidable. In these situations, we have tried to examine other available resources about the reviewed studies (e.g., authors homepage and tool training movie) to make our interpretations and inferences as reliable as possible in this kind of effort.

Table 5
Data items extracted from each study and related research questions.

#	Data item	Description	Research questions (Section 2.1)
D1	Author(s)	The author(s) of the paper.	
D2	Year	In which year was the study published?	Overview of studies
D3	Title	The title of the paper.	
D4	Venue	Publication type of the study: including Journal, Conference, Workshop, etc.	Overview of studies
D5	Citation count (Google Scholar)	The citation count of the study in Google Scholar.	RQ2, Overview of studies
D6	Architecting activity(ies)	What architecting activity(ies) are supported by the proposed visualization technique(s) in the study?	RQ1.2, RQ1.2.1
D7	Visualization technique(s)	The visualization technique(s) proposed and/or used in the study.	RQ1.1, RQ1.2.1, RQ5
D8	Solved problem(s)	What problem(s) in software architecting do the visualization technique(s) try to solve?	RQ3
D9	Merit	Advantages and benefits of the proposed visualization technique(s) in comparison with other techniques.	RQ3
D10	Evidence level	The evidence level of visualization technique used.	RQ2, RQ4, RQ5
D11	Study findings	The major findings about using visualization technique(s) in software architecting.	RQ3
D12	Level of tool support	The level of tool support for architecture visualization provided in the study: automatic, semi-automatic, or manual.	RQ1.3
D13	Application domain	The type of application used as a case study for validating visualization technique(s).	RQ4, RQ5
D14	Viewpoint	The architecture viewpoint (i.e., structural, decisional, or both) that is supported by the proposed visualization technique(s).	RQ1.1, RQ1.3, RQ3

3.1. Demographic data

Before reporting the results from the synthesis and analysis of the relevant data extracted from the studies included in this SLR, we report the demographic information about the included studies: the publication venues and types, the citation status and the chronological view. All of the included studies are listed in Appendix A.

3.1.1. Publication venues and types

An attempt to identify the types and venues for publications of a particular topic/theme can potentially be very useful for researchers who may be interested in conducting research on a relevant topic. That is why one of the silent reporting elements of a SLR is the demographic information about the papers included in that SLR. Table 6 shows how the 53 primary studies are distributed over 26 publication venues including journals, conferences, and workshops. It is clear from Table 6 that ECSA, IWPC, and VISSOFT can be considered the leading venues for publishing work on architecture visualization techniques as they have published 11.3% (6 studies), 9.4% (5 studies), and 9.4% (5 studies) of the selected studies respectively. There are 13 venues with only one paper selected and 49.05% (26 studies) of the selected studies were published in only 6 venues including ECSA, IWPC, VISSOFT, WICSA, SoftVis, and WCRE. 14 studies (26.4%) have been published in relevant venues on software architecture (i.e., ECSA, WICSA, QoSA, and SHARK) and 12 studies (22.6%) have been published in the venues that are relevant to applying visualization techniques in software domain (i.e., VISSOFT, SoftVis, VLHCC, PacificVis, VL, and VisSym). Most of studies have been published in conferences (39 studies, 73.5%) and workshops (8 studies, 15%), while only 6 studies (11.3%) have been published in journals. Five venues (the last five in Table 6) which are not listed in the target venues for manual search in Table 3, published 5 studies (9.4%) that were retrieved through automatic search, including SCP (Journal of Science of Computer Programming), SEKE (International Conference on Software Engineering and Knowledge Engineering), SAC-SE (ACM Symposium on Applied Computing - Software Engineering Track), VisSym (IEEE Symposium on Visualization), and VL (IEEE Symposium on Visual Languages). The selected studies have been published in both general software engineering venues and specific venues on SA and software visualization. This finding demonstrates that this research topic has been broadly considered by software architecture, software visualization, and software engineering researchers with different research interests.

3.1.2. Citation status and chronological view

The citation status of a paper can partially show the quality of the reported work and also the maturity of the proposed techniques in a study. It can also show the impact of a study on the reviewed topic. Table 7 provides a general overview of the citation status including citation counts, citation counts excluding self-citations, and average citation count per year (i.e., citation counts excluding self-citations/[2013-publication year]) of the included studies. These numbers have been obtained from Google Scholar on 12 July 2013. As shown in Table 8, 15 studies (28.3%) have been cited by more than 20 sources and meanwhile 31 studies (58.4%) have been cited by less than 10 sources, considering 19 out of these 31 papers were published between 2009 to 2011, a low citation count is understandable. Note that 7 studies (13.2%) got 0 citation count excluding self-citations and this finding is not a surprise as all these 7 studies were published in the last two years of the review period, i.e., 2010 and 2011. The top 5 studies that got the highest citation counts in range of 77–186 are [S32, S38, S41, S43, S49], in which four out of them were published in conferences and only one [S32] was published in journal. Three studies [S32, S38, S41] in the top 5 studies that were published after 2006 are all about visualization of architecture decisional viewpoint (e.g., architectural design decisions and rationale), which shows that this new trend to focus on design decisions in SA is gaining significant popularity in SA community.

Fig. 2 presents the number of the selected studies published per year within the review period from 1999 to 2011. Note that for year 1999 and 2011, the review only covers the studies published after 1 February 1999 and before 1 July 2011, which are not a full year. Trend line in Fig. 2 shows the number of selected studies per year has been increasing since 1999. This trend has more ascending slope since 2005, and we noticed that 35 studies (66%) were published during the last 5 years which means the role and application of visualization techniques to support software architecting process are gaining increasing interest and attention.

3.2. Study quality assessment

The 53 primary studies were evaluated by two of the authors against a set of study quality assessment questions listed in Table 9. These questions were adopted and adjusted from (Dybå and Dingsøyr, 2008). In contrast to the study quality assessment described in (Kitchenham and Charters, 2007), these questions were not used for the study selection but they were employed for

Table 6
Distribution of the selected studies on publication venues.

Pub. Venue	#	%	Pub. Venue	#	%	Pub. Venue	#	%
ECSA	6	11.3	ICSM	2	3.7	PacificVis	1	1.8
IWPC	5	9.4	QoSA	2	3.7	SEW	1	1.8
VISSOFT	5	9.4	CSMR	2	3.7	ESE	1	1.8
WICSA	4	7.5	SHARK	2	3.7	SCP	1	1.8
SoftVis	3	5.6	ICECCS	1	1.8	SEKE	1	1.8
WCRE	3	5.6	ASE	1	1.8	SAC-SE	1	1.8
JSS	2	3.7	APSEC	1	1.8	VisSym	1	1.8
SMPR	2	3.7	QSIC	1	1.8	VL	1	1.8
ICSE	2	3.7	VLHCC	1	1.8			

Table 7
An overiew of citation status of the selected studies.

Studies ID	Citation counts	Citation counts excluding self-citations	Average citation count per year	Studies ID	Citation counts	Citation counts excluding self-citations	Average citation count per year
S1	0	0	0.0	S28	1	0	0.0
S2	1	1	0.3	S29	2	1	0.5
S3	4	0	0.0	S30	10	10	2.5
S4	0	0	0.0	S31	5	4	1.0
S5	0	0	0.0	S32	124	111	18.5
S6	17	13	2.2	S33	10	8	4.0
S7	31	26	3.7	S34	4	1	0.5
S8	6	2	0.7	S35	7	1	0.3
S9	4	1	0.3	S36	5	5	2.5
S10	4	2	0.5	S37	23	21	4.2
S11	34	28	4.0	S38	214	186	26.6
S12	21	12	2.0	S39	4	4	1.3
S13	14	13	2.2	S40	29	22	2.2
S14	2	2	1.0	S41	101	88	14.7
S15	55	48	3.4	S42	10	7	0.9
S16	18	16	2.0	S43	109	103	9.4
S17	9	9	0.8	S44	34	29	5.8
S18	6	6	1.5	S45	24	23	3.3
S19	1	1	0.2	S46	8	8	1.1
S20	8	6	1.0	S47	2	2	0.3
S21	44	36	4.5	S48	64	59	6.6
S22	0	0	0.0	S49	77	77	6.4
S23	17	15	2.5	S50	9	8	2.7
S24	0	0	0.0	S51	43	35	2.7
S25	6	2	0.3	S52	5	4	0.3
S26	6	5	0.6	S53	6	4	0.8
S27	11	10	1.7				

Table 8
Citation counts of the selected studies excluding self-citations.

Cited by	<10	10–20	20–30	30–40	40–50	50–60	>60
No. of studies	31	7	6	2	1	1	5

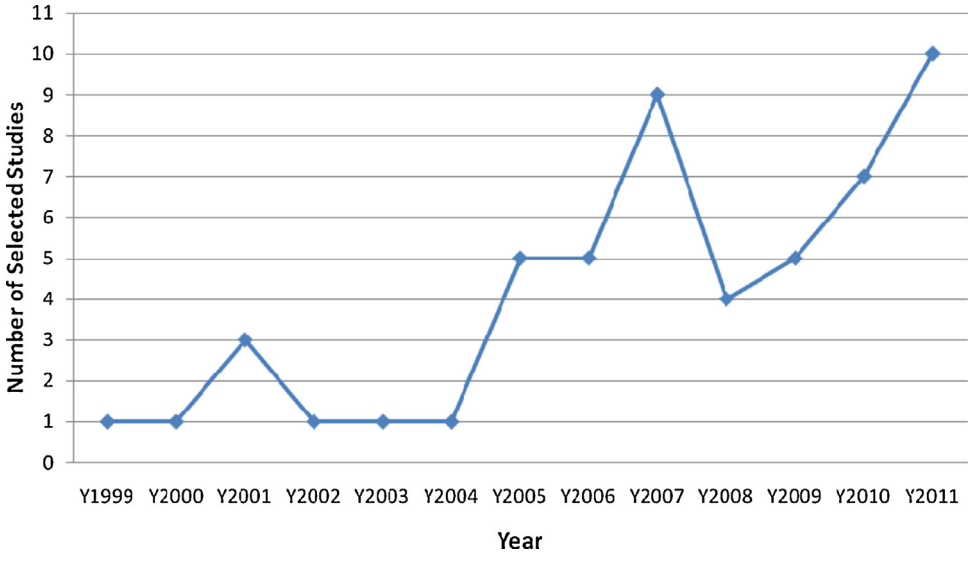


Fig. 2. Number of selected studies published per year.

Table 9
Study quality assessment questions.

#	Study quality assessment question	Yes	Partially	No
Q1	Are the aims and objectives of the study clearly specified?	34 (64.1%)	19 (35.8%)	0 (0.0%)
Q2	Is the context of the study clearly stated?	34 (64.1%)	18 (33.9%)	1 (1.8%)
Q3	Does the research design support the aims of the study?	39 (73.5%)	14 (26.4%)	0 (0.0%)
Q4	Has the study an adequate description of the technique for visualization?	40 (75.4%)	13 (24.5%)	0 (0.0%)
Q5	Is there a clear statement of findings by applying visualization technique in software architecture?	27 (50.9%)	26 (49%)	0 (0.0%)
Q6	Do the researchers critically examine their potential bias and influence to the study?	2 (3.7%)	2 (3.7%)	49 (92.4%)
Q7	Are limitations of the study discussed explicitly?	4 (7.5%)	23 (43.3%)	26 (49%)

validating the results of the selected studies. Each question could be answered according to a ratio scale: "Yes", "No", and "Partially" during the data extraction process (see Section 2.5). The answers for each study show the quality of a selected study and the credibility of the study's results. The result of the quality assessment of the included studies can reveal the potential limitations of the current research and guide the future research in the field (Kitchenham and Charters, 2007; Dybå and Dingsøyr, 2008).

As shown in Table 9, all studies state the aims and objectives of the conducted research (Q1) and provide research design to achieve these objectives (Q3). The Q2 was answered positively by all studies (except one), which means the reviewed studies have an adequate description of context (e.g., industry and laboratory setting) in which the research was carried out. The answers of Q4 and Q5 can reflect the accuracy of the data extraction results (Li et al., 2013). Concerning Q4, 40 out of 53 studies (75.4%) described their proposed visualization techniques adequately and 13 studies (24.5%) could address this issue to some extent. About Q5, more than half of the studies explicitly discussed what findings had been achieved by applying the reported visualization techniques in SA; and others explained the findings partially. It is clear from Table 9 that, the Q6 was answered with "No" by a majority of the studies, and only two studies (3.7%) examined the researchers' bias and influence on the outcomes of the study. Almost half of the studies (49%) did not discuss any limitations, drawbacks, or required improvement in the proposed visualization techniques in software architecture.

3.3. Types of visualization techniques used in architecting process

To better answer RQ1.1, we did not employ a pre-defined classification of visualization techniques (Novais et al., 2013). Instead, we employed thematic analysis (Braun and Clarke, 2006), a qualitative analytic method for identifying, analyzing, and reporting patterns (themes) within collected data (i.e., data item D7 in Table 6). We followed the five steps of the thematic analysis method as detailed below: (1) familiarizing with the data: in this step, we tried to read and examine the extracted data item D7 "visualization technique(s)" to be well acquainted with the VTs used in each study; (2) generating initial codes: in this step the initial list of employed VTs in selected studies were extracted from collected data; (3) searching for VTs (i.e., themes): we tried to combine different initial VTs generated from the second step to form an overarching VT since the studies may use different terms to propose and describe their employed VTs, which might be essentially the same; (4) reviewing and refining VTs: the VTs identified from the previous step were checked against each other. For example, two apparently separate VTs might form one VT; (5) defining and naming VTs: in this step, we defined the VTs to describe the essence of each VT and a clear and concise name for each VT was provided. By using the thematic analysis method, a classification with four types of VTs (i.e., themes in thematic analysis) was recognized.

Fig. 3 shows the status of various visualization techniques that are currently used in software architecting over time. The number in each column of Fig. 3 reveals that how many studies employ a specific visualization technique in each year. There is a significant difference in terms of the popularity of these visualization techniques. Note that some studies may use more than one visualization techniques. That is why the sum in Fig. 3 (57) exceeds the number of the reviewed studies (53). The visualization techniques (VTs) in SA reported in the reviewed studies are described below:

3.3.1. Graph-based visualization

The most popular VT used in software architecture is the graph-based technique as it has been reported in 26 studies (49.1%). As shown in Fig. 3, the graph-based visualization has been used steadily except years 2000 and 2004. The graph-based visualization uses nodes and links to represent the structural relationships between architecture elements and it puts more emphasis on the overall properties of a structure than the types of nodes. For example, Su and colleagues [S10] report an approach to capturing and visualizing architecture document elements and their relationships produced by an attribute-driven design method using a graph that results in non-linear exploration and better finding needed information. The work reported in [S22] uses a graph to represent both hierarchal and usage-relation (dependency) of software entities

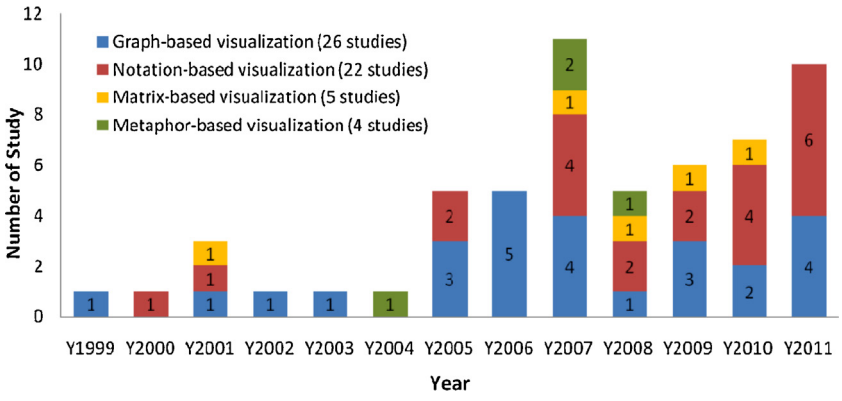
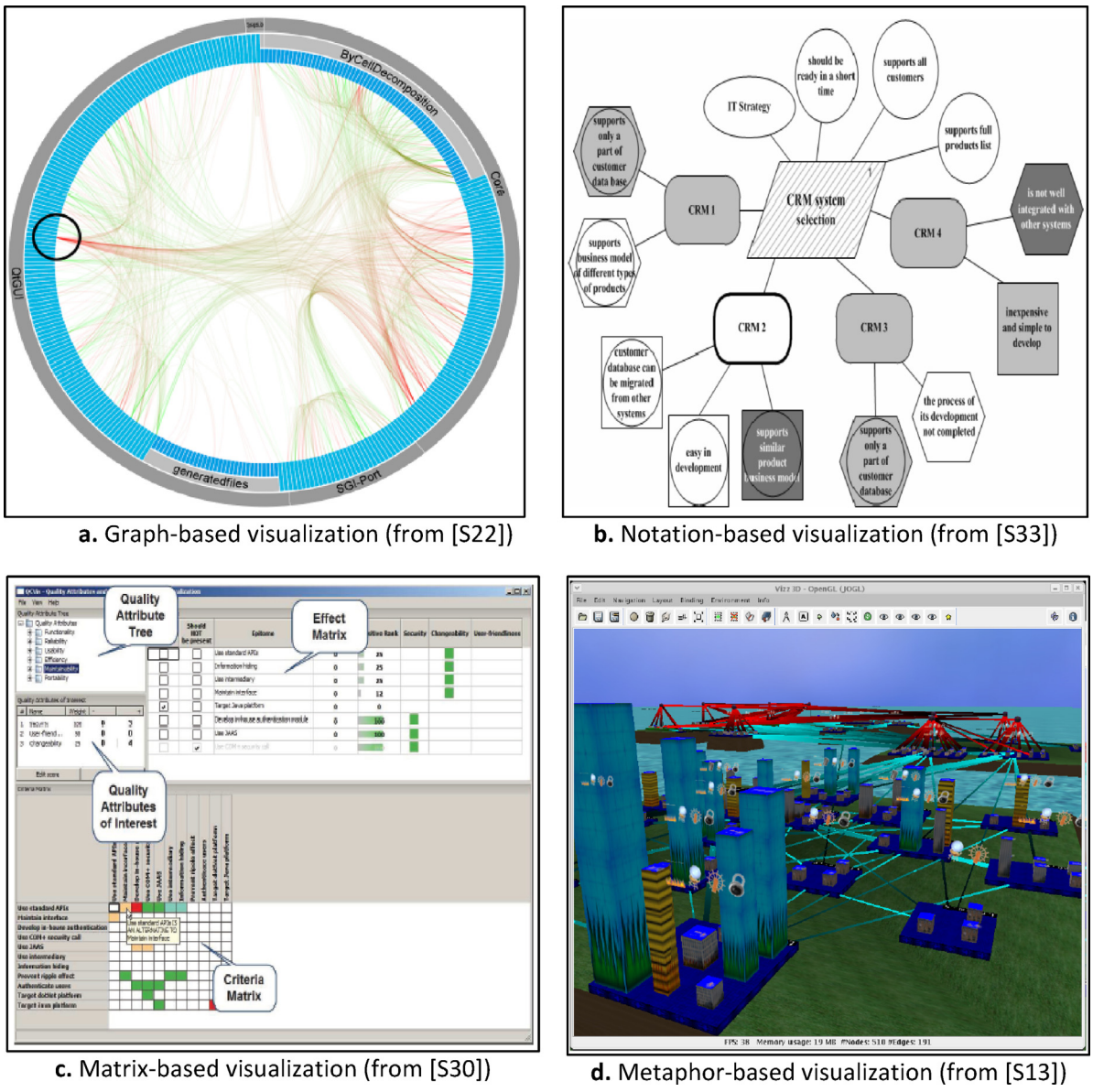


Fig. 3. Types of visualization techniques employed and their distribution in primary studies over time period.



a. Graph-based visualization (from [S22])

b. Notation-based visualization (from [S33])

c. Matrix-based visualization (from [S30])

d. Metaphor-based visualization (from [S13])

Fig. 4. Examples of four visualization techniques used in software architecture.

(e.g., modules, packages, classes, and files) in one view. Software entity types and usage-relations are indicated by different color schemes and color gradients. Gray and blue nodes represent modules and files respectively and a color gradient (e.g., from green to red) represents a dependency direction between software modules. Fig. 4.a provides a screenshot of a graph-based VT used for SA visualization [S22]. Tree is a special kind of graph in which there is no loop between nodes (Diehl, 2007). It was employed in [S9, S11, S12, S29, S43]. The studies [S9, S11, S12, S43] use a tree-based visualization to represent hierarchical decompositions of a system. The work reported in [S29] visualizes users' exploitation paths in software architecture documents as tree-based visualization.

3.3.2. Notation-based visualization

This category is a combination of three modeling techniques: SysML (systems modeling language), UML (unified modeling language), and specific notation-based visualization. SysML is a general-purpose modeling language for specification, analysis, design, verification, and validation of complex systems, including software, hardware, and information systems (OMG System, 2007). UML is an industry standard and general-purpose visual modeling language to specify, design, construct, and document software-intensive systems (OMG, 2007). SysML is built on UML and reuses and extends it for system engineering applications (OMG System, 2007). Tsadimas and colleagues [S4] provides only study that uses SysML and extends SysML profile to visualize and evaluate SA. Kamal and Avgeriou [S2] create a specific UML profile including stereotypes and constraints to express and model architectural pattern variants. The specific-notation refers to new notations developed for special purposes in software architecting. These notations are customized notations, but not standardized as SysML and UML. To give an example, Zalewski and colleagues [S33] define two architecture diagrams: architectural decisions relationship diagram (ADRD) and architecture decision problem map (ADPM). The former visually represents architectural decisions, their possible states (e.g., defined and solved), and the relationships between them; while the latter represents the internal structure of architectural decision problem. Both diagrams use customized notations. A screenshot of the ADPM diagram is shown in Fig. 4b. There are 22 studies (41.5%) that employ this kind of VT, including

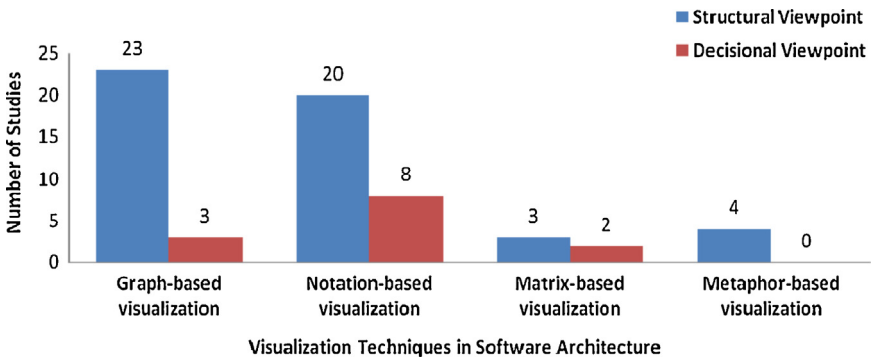


Fig. 5. Relationship between visualization techniques and architectual viewpoints.

UML, SysML, and specific notations. Fig. 3 reveals that 81.8% (18 out of 22) studies that employ the notation-based VT were published in the last 5 years, which means notation-based VTs were gaining significant popularity to visualize SA over the last five years. The graph-based and notation-based VTs are dominant VTs for visualizing SA except year 2004. The notation-based VT tries to present the relationship between and role of elements in a structure and various notations are provided to represent different elements and relationships, and that is why the name and type of the nodes (elements) and links (relationships) are important pieces of information. Conversely, the graph-based VT intends to show the overall properties of a structure, hence, the name and label of the nodes and links are not as important as in notation-based VT.

3.3.3. Matrix-based visualization

Matrix-based visualization can act as a complementary representation of a graph. In comparison with graph-based visualization, a matrix-based representation is less intuitive but it provides complementary information when the graph is large or dense. For example, Beck and Diehl [S8] use matrix to compare SA descriptions and Lungu and Lanza [S12] use matrix to display detailed dependency between two modules. de Boer and colleagues [S30] employ matrix to support auditor in architectural decision-making process during assessment of product quality. To achieve this, the study employs architectural decisions in software product audits as quality criteria and provides two matrices: effective matrix and criteria matrix. Effective matrix shows quality criteria that have positive or negative effect on quality attributes of interest in the current audit. The auditor can employ effective matrix in "trade-off analysis" and "if-then" scenarios in decision-making process. The criteria matrix shows the relationships between quality criteria so it can assist the auditor in determining the impact of a decision. Fig. 4c shows both effective and criteria matrices used in [S30]. Tables as a kind of matrix are used in [S17, S37]. Bril and Postma [S17] use tabular representation to visualize the decomposition of architecture entities of a system and the architectural connectivity metrics between the entities. Architectural connectivity metrics indicate the degree of connectivity between or within architectural entities in a system. Lee and Kruchten [S37] use table view to show architecture design decisions and their relationships.

3.3.4. Metaphor-based visualization

This category uses familiar physical world contexts (e.g., cities) to visualize SA entities and their relationships (Wettel and Lanza, 2007). The use of metaphors makes the visualization process particularly intuitive and effective (Balzer et al., 2004). The works reported in [S6, S13, S44] use city metaphor and [S48] uses landscape metaphor to represent SA entities and their relationships. To give an example, the study reported in [S6] represents source files as basic modules in SA by different textures of building. For instance, a source file with more than 200 LOC (lines of code) is represented as "office building" and the number of floors in each building indicates the number of global variables declared in the file. Header file is represented as "city hall" and the number of functions in the header file determines the height of the "city hall". Panas and colleagues [S13] use single-view metaphor visualization (see Fig. 4d) to represent multiple aspects of large-scale software systems, which contributes to reduced visual complexity.

3.3.5. Visualization techniques and architectural viewpoints

To show how VTs are related to structural and decisional viewpoints of software architecture, we examined the data items D7 (visualization technique) and D14 (viewpoint) in Table 5 simultaneously, which is shown in Fig. 5. Note that some studies may support both structural and decisional viewpoints as well as use more than one visualization techniques. That is why the sum in Fig. 5 (63) exceeds the number of the selected studies (53). From Fig. 5, we can see that (1) VTs in SA are mostly used to visualize structural architectural viewpoint (50 studies), which is supported by all VTs; (2) decisional viewpoint is mostly visualized by notation-based visualization; and (3) metaphor-based visualization has never been used to visualize decisional architectural viewpoint.

3.4. The architecting activities supported by the visualization techniques

This section presents the findings to answer RQ1.2. These findings are based on our examination of the results about the architecting activities, general and specific, that are being supported by various VTs discussed in Section 3.3 To achieve this, the data item D6 from Table 5 were analyzed. Note that some studies introduce the VTs that support several architecting activities and this is the reason that the sum in Table 10 (87) exceeds the number of the selected studies (53). We adopted the classification of architecting activities proposed in (Li et al., 2013), in which general architecting activities include architecture recovery, architecture description, architecture understanding, change impact analysis,

Table 10
Distribution of architecting activities supported by visualization techniques.

Architecting activity	Number of studies
Architecture recovery (AR)	25
Architectural evolution (AEV)	16
Architectural evaluation (AE)	11
Change impact analysis (CIA)	10
Architectural analysis (AA)	9
Architectural synthesis (AS)	7
Architectural implementation (AI)	5
Architecture reuse (ARU)	4

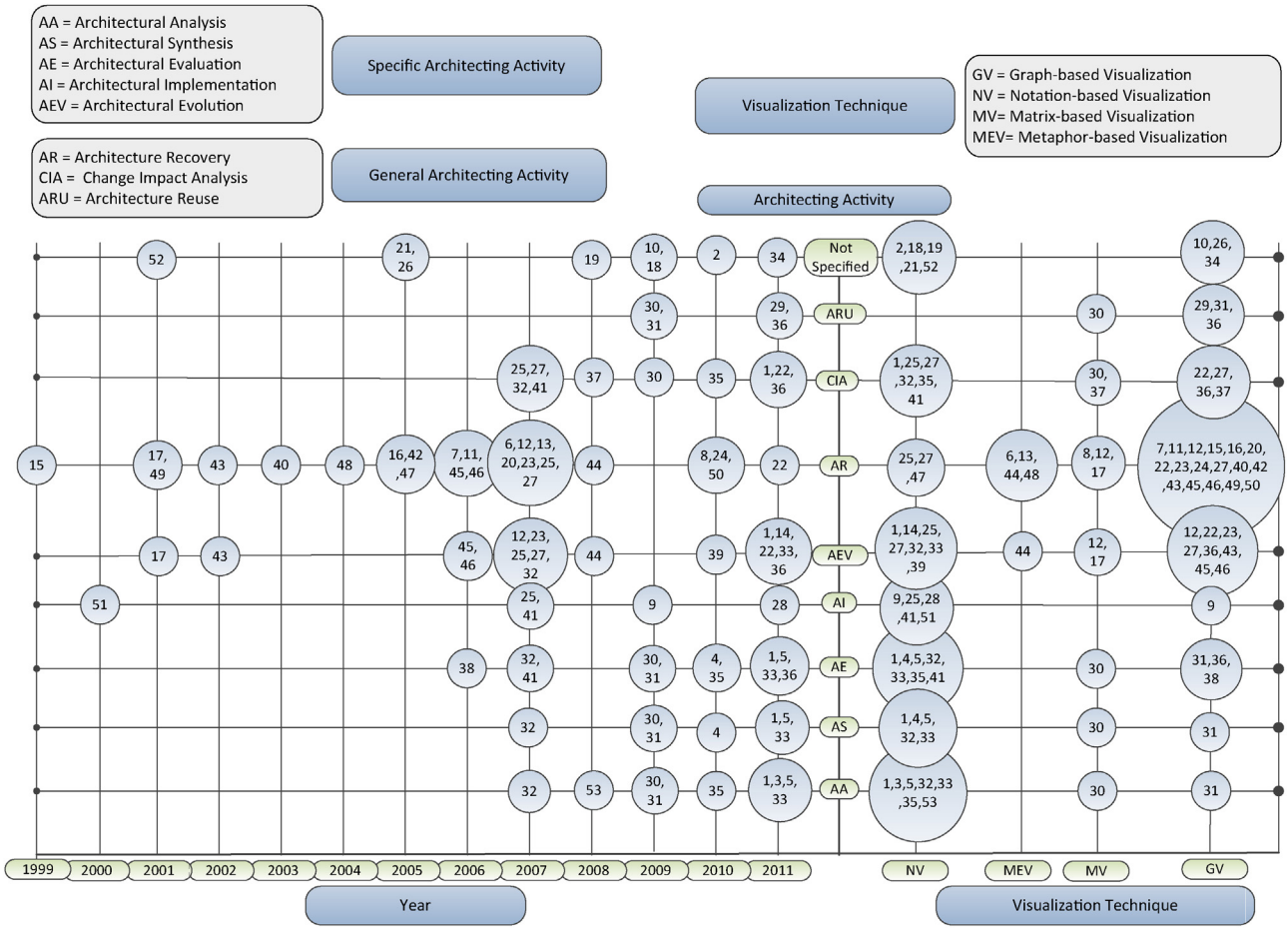


Fig. 6. Studies distribution over the range of architecting activities, visualization techniques, and time period.

and architecture reuse, and specific architecting activities are architectural analysis, architectural synthesis, architectural evaluation, architectural implementation, and architectural evolution. The specific architecting activities construct the entire architecture lifecycle, and are supported by the general architecting activities (Li et al., 2013). Note that two general architecting activities "architecture understanding" and "architecture description" are not taken into account in this review because of the following reasons. The dominant function of visualization is to improve understanding therefore we ignore "architecture understanding" activity because visualization techniques by default are expected to always support this activity. Meanwhile, architecture visualization is a kind (part) of architecture description, therefore "architecture description" activity is also not explicitly considered in this review.

We found that architectural implementation (AI) has seldom been supported by using VTs in SA that only 5 studies [S9, S25, S28, S41, S51] support this architecting activity. Buchgeher and Weinreich [S9] developed a tool suite to support the design, analysis, visualization, and implementation of component-based software systems. The architectural structure diagrams visualized by the tool suite are used to synchronize between architecture description and implementation. Demirli and Tekinerdogan [S28] propose an approach with an accompanying tool to define, model, and visualize executable architectural views based on domain specific language, and further support architectural implementation with model-driven development. Architecture reuse (ARU) is supported by VTs reported in 4 studies (Shahin and Liang, 2009; Kruchten and Lago, 2006) [S29, S30, S31, S36]. The work reported in [S30, S31] provides functionality to reuse architectural knowledge (i.e., decision, rationale and quality criteria) through architectural knowledge visualization. Su and colleagues [S29] capture and visualize the users' exploration paths in architecture documents and allows searching and reusing the analyzed exploration paths. Ciraci and colleagues [S36] represent possible architectural evolution paths using graph and provides guidelines to select the best evolution path, and further provides functionality to reuse architecture changes during the architectural evolution. Architecture recovery (AR) is the most popular architecting activity for applying VTs with 25 studies. Architectural evolution (AEV) in 16 studies, architectural evaluation (AE) in 11 studies, change impact analysis (CIA) in 10 studies, architectural analysis (AA) in 9 studies, and architectural synthesis (AS) in 7 studies have been supported by VTs respectively. The distribution of various architecting activities supported by VTs has been presented in Table 10.

3.5. Relationships between visualization techniques and architecting activities

To answer RQ1.2.1, we examined the results obtained from RQ1.1 and RQ1.2. Fig. 6 shows the correlation between VTs and architecting activities. A bubble on the left side of Fig. 6 indicates one study or several studies over an architecting activity (*y*-axis) in a specific year (left *x*-axis) and a bubble on the right side specifies one study or several studies applying a certain VT (right *x*-axis) for an architecting activity (*y*-axis). The numbers in a bubble denote the identification numbers of the studies (see Appendix A). Because of limited space in the figure, we have excluded "S" in the study identification numbers in Fig. 6.

Table 11
Number and percentage of papers associated to each visualization technique and architecting activity.

	Graph-based	Notation-based	Matrix-based	Metaphor-based
Architecture recovery	17 (32.1%)	3 (5.6%)	3 (5.6%)	4 (7.5%)
Architectural evolution	8 (15.1%)	7 (13.2%)	2 (3.7%)	1 (1.8%)
Architectural evaluation	3 (5.6%)	7 (13.2%)	1 (1.8%)	0 (0.0%)
Change impact analysis	4 (7.5%)	6 (11.3%)	2 (3.7%)	0 (0.0%)
Architectural analysis	1 (1.8%)	7 (13.2%)	1 (1.8%)	0 (0.0%)
Architectural synthesis	1 (1.8%)	5 (9.4%)	1 (1.8%)	0 (0.0%)
Architectural implementation	1 (1.8%)	5 (9.4%)	0 (0.0%)	0 (0.0%)
Architecture reuse	3 (5.6%)	0 (0.0%)	1 (1.8%)	0 (0.0%)

As we can see in Table 11, graph-based VT is mostly employed in AR and AEV activities with 17 (32%) and 8 (15%) studies respectively. However, this technique is less popular in AA, AS, and AI with only one study for each. Graph-based VT is the only one VT which has been applied to all architecting activities. Metaphor-based VT is only used for AR and AEV activities. AA, AS, and AE are the most popular architecting activities for applying notation-based VT, but notation-based technique is not used for supporting ARU. Fig. 6 reveals that AR and AEV are supported by all types of VTs, but there is no VT that can support all architecting activities. Note that 8 studies [S2, S10, S18, S19, S21, S26, S34, S52] did not explicitly specify what architecting activities have been supported by the reported VTs. Hence, they have been mapped as "not specified" in the axis of architecting activity in Fig. 6.

3.6. Level of tool support

In order to answer RQ1.3, we analyzed the data item of D12 from Table 5 to get an understanding of the level of tool support for the visualization techniques reported in the primary studies reviewed in this SLR. An investigation of tool support can be considered an indicator of the limitations of the proposed SA visualization techniques. It can also indicate whether the reported techniques are theoretical (e.g., proposal or plan) or practical (i.e., implemented in appropriate tools), which can be interesting information for practitioners. Fig. 7 shows that 6 studies (11.3%) provide manual support including 4 studies [S1, S2, S19, S34] that do not provide any tools and 2 studies [S5, S39] provide tools, which completely depend on human effort. There are 25 studies (47.1%) that provide semi-automatic tools, and 22 studies (41.5%) have automatic tool support. Some tools have been developed from scratch and others have been extended from existing tools, for example, Eclipse visualization plug-in. Since most of the studies (49, 92.4%) provide various kinds of tools (including two tools that provide manual support), it can be concluded that tool support is a very important concern of stakeholders, who use visualization techniques in SA.

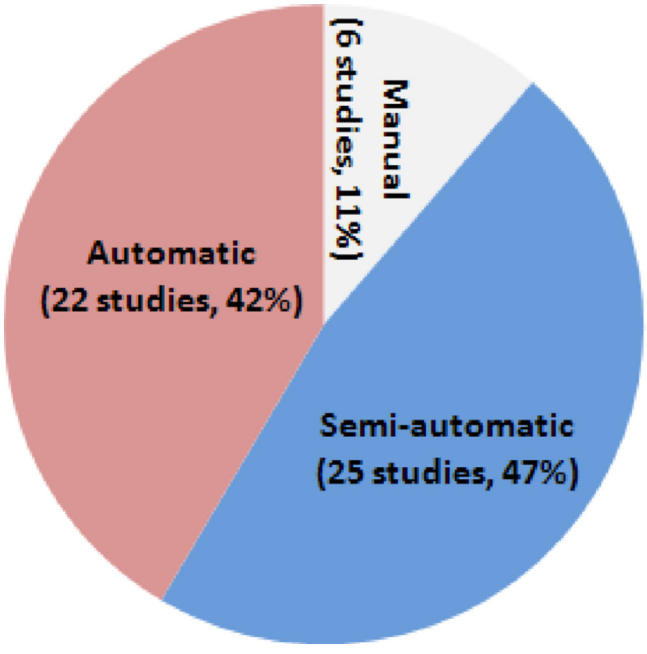


Fig. 7. Status of tool support provided by selected studies.

3.7. The available evidence for adoption of visualization techniques in software architecture

To answer RQ2 (the evidence available to adopt a proposed VT), we analyzed the data collected for data item D10 in Table 5. Table 12 presents the distribution of the studies according to the five levels of evidence as described in Section 2.5. As shown in Table 12, the main method for evaluating architecture visualization techniques is industrial study. There are 26 studies (49%) that have been evaluated in industrial software projects (e.g., medium or large projects) as case studies. The so-called "toy example and demonstration" have been used by 23 studies to demonstrate how to use the reported VTs. According to Table 12, little attention has been paid to academic study (e.g., controlled experiment), which is also revealed in general software engineering research (Sjoberg et al., 2005) and software evolution visualization research (Novais et al., 2013). There are only 3 studies [S5, S53, S37] that used controlled experiments in academic settings and expert opinions respectively to evaluate their proposed techniques. [S23] and [S35] used both expert opinions and industrial case study to validate the reported techniques, but we chose the higher one (i.e., industrial study) as their evidence level because the objective we analyzed the evidence levels of the selected studies was to investigate the maturity of the reviewed studies. None of the selected studies employed industrial controlled experiments. The possible reasons can be that it is difficult to find motivated industrial participants to do experiments (Cornelissen et al., 2011) as well as conducting the controlled experiment needs an excessive amount of effort and resources (Sjoberg et al., 2005); on the contrary, industrial partners try to use the VTs directly in their work (i.e., most studies have the evidence level of "industrial study"). The proposed VT in [S50] is ranked as industrial practice, in which the technique has been adopted by several industrial organizations. Table 12 reveals that most of the studies (27 studies, 50.9%) have used industrial study and industrial practice to evaluate the proposed VTs. The practitioners are always faced

Table 12
Evidence available in the selected studies to adopt the proposed visualization techniques.

Evidence levels	Number of studies	%
Toy example/demonstration	23	43.3
Expert opinion/observation	1	1.8
Academic study	2	3.7
Industrial study	26	49.1
Industrial practice	1	1.8

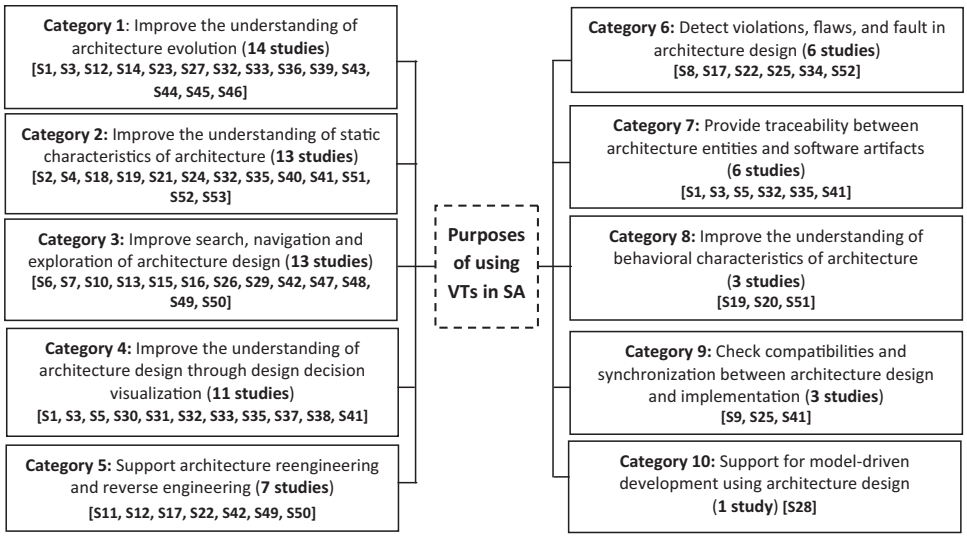


Fig. 8. Categorization of the purposes of using visualization techniques in software architecture.

with challenges whether to adopt a new technology, because there is little objective evidence to confirm its suitability, limits, qualities, costs, and inherent risks (Dyba et al., 2005). This can potentially lead to less industrial acceptance. However, the high evidence levels of selected studies reported in this SLR with the moderate quality (i.e., obtained from quality assessment questions in Section 3.2) and the high percentage of tool support (in Section 3.6) improve the practical applicability of the reported results and enable practitioners to use and apply the results into daily software architecting practice.

3.8. Different purposes of using visualization techniques in software architecture

This section reports the results from analyzing the data item D8 (solved problem(s)), D9 (merit), and D11 (study findings) for answering RQ3, "*What are the different purposes of using visualization techniques in software architecture*?" These data items were also analyzed using the thematic analysis method (Braun and Clarke, 2006), as describe in Section 3.3, for identifying and synthesizing the purposes of applying visualization techniques to support software architecting process. Our analysis resulted in the identification of ten categories of the purposes of using VTs in SA (i.e., themes in thematic analysis), which are shown in Fig. 8. We provide detailed descriptions of the categories of the identified purposes of applying visualization techniques in software architecting.

3.8.1. Category 1: Improve the understanding of architecture evolution

This category of the reviewed studies report the use of visualization to help stakeholders to understand the changes during an architecture's evolution [S1, S3, S12, S14, S23, S27, S32, S33, S36, S39, S43, S44, S45, S46]. Orlic and colleagues [S1] have developed a set of specific notations as building blocks for architectural reasoning diagrams (AR-diagram) to capture and visualize architectural reasoning process. Their AR-diagram has four structural layers: *stakeholder layer*, *requirement layer*, *analysis and decision layer*, and *architecture layer*. The key element in an AR-diagram is an architectural decision, which is specified as a special case of requirement. An architectural artifact is considered as a specific kind of decision. The main objective of an AR-diagram is to supplement existing architectural description methods. It supports architectural evolution and change impact analysis. The VTs used in [S12] and [S23] can represent how the relations (e.g., dependency edges) between modules or subsystems get evolved over time. Lungu and Lanza [S12] have introduced a "filmstrip" VT to display the evolution of inter-module relations through multiple versions of a system. According to this technique, a pattern of architecture evolution showing the relationships between modules are extracted. A visual representation of this information can be used to filter and reduce the number (information overload) of visible inter-module relations depending on the type of design activity a user is doing. Hindle and colleagues have reported a VT that can be used to show how dependency between two subsystems has changed before and after changes made [S23].

The approaches reported in [S45, S46] use version history to recover structure of a software system and visualize co-changes of software entities (e.g., modules and files) over time. Both approaches shows how structural dependencies have changed over time; however, the work reported in [S45] emphasizes more on the *actual* structural dependencies that are the reasons for co-changes. In both studies, the size of a node indicates how many co-changes have happened to that node. Fischer and Gall [S45] state that if two file-pairs (i.e., files are changed simultaneously) have a high co-change rate, they should be placed close to each other. The approach reported in [S46] asserts that if two subsystems have been changed together, they will be placed close to each other because they are likely to be more inter-dependent.

3.8.2. Category 2: Improve the understanding of static characteristics of architecture

This category includes the studies that report VTs purport to help understand the static aspects of SA [S2, S4, S18, S19, S21, S24, S32, S35, S40, S41, S51, S52, S53]. The static characteristics of SA do not change during execution of a system, e.g., topologies, assignment of components and connectors (Taylor et al., 2009). The work reported in [S2] proposes to reuse architectural primitives in combination with an extended UML meta-model using UML profiles. This approach explicitly represents architectural patterns for a better understanding of variability in architectural patterns. To integrate enterprise information system (EIS) architecture design with architectural solutions evaluation, the research reported in [S4] extends SysML to incorporate solution evaluation view in EIS design process. The proposed solution view can help an architect to better understand the effects of his/her re-design and requirement readjustment decisions. Software metric information is useful for analyzing SA (Fenton and Pfleeger, 1996), but most architecture

modeling tools do not support correlation analysis between metric and architecture diagrams.

The studies reported in [S18, S21] attempt to visualize metric data of architecture diagrams. The work reported in [S21] has developed *MetricView* tool that uses special icons to visualize software metrics of UML architecture diagrams for making UML diagrams (e.g., class diagrams and collaboration diagrams) more clear and understandable. The *MetricView's* limitation of just displaying the relations of a metric with a single element of architecture has been addressed by another tool reported in [S18]; this tool visualizes the grouping of metrics and the related architecture elements, also called area of interest (AOI). It also displays the architecture elements that are located in an AOI.

The research reported in [S52] uses a formal language LePUS, a symbolic and visual language, to specify and visualize object-oriented architectures. This reported purports to improve the understanding of patterns used in an architecture.

3.8.3. Category 3: Improve search, navigation, and exploration of architecture design

All the studies (except [S10, S29]) reports VTs approaches that visualize architecture/structure of software systems by extracting the software architecture/structure from mainly source code. The studies reported in [S6, S48] use metaphor-based VTs to explore and navigate software architecture/structure. Another study [S6] provides navigable visualization of architecture and metrics information together in one view. Balzer and colleagues [S48] attempt to balance between information density and comprehensibility in order to reduce visual complexity and information loading. The work reported in [S10, S29] uses visualization for better navigation and exploration in software architecture documents (SAD), mostly produced using productivity applications such as MS Word and Excel Spreadsheet. By this visualization, users can get a better understandability and readability of SAD. In The VTs presented in both studies structure and present architecture information in a way that can enable users to quickly search/retrieve/navigate.

Su and colleagues [S10] report *KaitoroBase* tool that provides non-liner navigation and visualization of structured information in SAD based on a visual Wiki. To facilitate information retrieval in SAD, the work reported in [S29] introduces "chuck" concept that is defined as a group of related architectural information. The proposed approach and a supporting tool, called *KaitoroCap*, [S29] can capture, retrieve, analyze, and finally visualize the exploration paths by SAD users in a hierarchical tree view, which is used to support the identification of chunks in SAD. The work reported in [S47] has developed a tool, *software architecture browser* (*SAB*), to visually explore software systems structure written in Java. In comparison to UML tools, *SAB* can show the desired level of detail for each class in a class diagram (i.e., customizable class diagram), allows users to declare hierarchical layout, and supports layered architectures.

3.8.4. Category 4: Improve the understanding of architecture design through design decision visualization

The studies placed in this category [S1, S3, S5, S30, S31, S32, S33, S35, S37, S38, S41] report VTs aimed at visualizing architectural design decisions (ADDs) to help concerned stakeholders to better understand and communicate architecture design. Shahin and colleagues [S5] have reported a controlled experiment aimed at investigating the usefulness of visualization of ADDs. This work asserts that the presented approach to visualizing ADDs and their rationale can improve the quality of SA design. The visualization approach proposed in [S30] purports to support software product auditors by providing a facility for reusing quality criteria (i.e., a particular type of design decision) in their early decision making process, especially in two main tasks: trade-off analysis and if-then scenarios. This approach is expected to help auditors to understand how architecture design meets the quality criteria with ADD visualization.

An approach combining non-functional requirements (NFRs) and design rationale (NDR) ontology for visualizing architecture rationale has been reported in [S31]. The method and an associated tool reported in [S31] can support architects in (1) understanding architectural decisions changes and (2) reviewing, comparing, and reusing architectural rationale of ADDs.

The work reported in [S38] has constructed an ontology to represent architectural knowledge and related artifacts, which are essential to construct an architectural knowledge repository. To determine how an architectural repository can be used and who will use it, a use case model (i.e., architecture tasks) has been introduced. A commercial ontology-based visualization tool, called *Aduna Cluster Map Viewer*, has been used to support the proposed use case model and its underlying ontology. However, only some use cases can be handled by the tool. The ontology-based architectural knowledge visualization can help architects to understand architecture design. Another VT approach reported in [S41] explicitly represents architectural decisions as first class entities in an architecture design model. The visualization approach proposed in [S41] helps architects to assess the consequence of ADDs and perform change impact analysis of ADDs on architecture design.

3.8.5. Category 5: Support architecture reengineering and reverse engineering

An architecture reverse engineering process examines an existing software system and represents its software components and the relationships between those components at different levels of abstraction (Harris and Reubenstein, 1995). Software engineering literature uses the terms like architecture recovery, architecture reverse engineering, and architecture reconstruction interchangeably. However, an architecture recovery is more specific than other two terms and it can be considered as a subset of architecture reverse engineering (Ducasse and Pollet, 2009). The architecture reengineering process focuses on re-architecting existing systems to satisfy new changes at architecture level and improve quality attributes such as maintainability and performance (Bengtasson and Bosch, 1998). In reengineering process, the current system must be understood (i.e., using reverse engineering process) and then its structure/architecture and implementation are transformed into a new form (Chikofsky and Cross, 1990). Reverse engineering is a sub-activity in reengineering. The studies [S11, S12, S17, S22, S42, S49, S50] report approaches to support reverse architecting by which SA is recovered first and then some facilities are provided for exploration and navigation. The work reported in [S11, S12, S49, S50] represents recovered architecture using dependency graph, but the studies like [S11, S12] focus more on the arrows in the graph and the work reported in [S49, S50] pay more attention to nodes in the graph.

Lungu and colleagues [S11] provide a set of package patterns that guide reverse engineers during exploration of package decomposition to decide whether or not all the packages of a view are relevant for architecture of a system. These package patterns, extracted automatically from packages' structure and their relationships with other packages, provide information to help reverse engineers to decide about the packages that should be expanded. The work reported in [S17] introduces a new architectural connectivity metric, called "directed connectivity", and an existing *module architecture browser* (*MAB*) tool was extended to visualize this new metric. The directed connectivity provides a means to measure how many connections exist between an architecture entity to other entities and supports incremental re-architecting in situations when existing architectural connectivity metrics of cohesion and coupling are insufficient.

Another work reported in [S22] has developed a tool that represents software structure hierarchy and inner dependency in a unified view. The tool enables an architect to visually perform reengineering tasks such as checking the possibility of a reengineering task and estimating the effort needed for doing that task. An architect can reorganize software structure by using the drag and drop capability in the tool and can estimate the potential impact of the desired change on the planned architecture. The VTs reported in [S49, S50] have developed two tools for static structure visualization, *SHriMP* and *Rigi.* Both of the tools can support reverse engineering software systems. *SHriMP* provides multiple views for enabling users to visualize, explore, browse, and document software structure in both low-level and high-level abstractions. *Rigi* tool facilitates understanding and re-documenting structure of large-scale software systems by visualizing their structure. In re-documenting, the low-level entities are grouped into high-level entities until appropriate abstraction level is achieved. *Rigi* tool uses interactive graph editor including selecting, layouting, filtering, and grouping functionality to represent software structure that reduces visual complexity.

3.8.6. Category 6: Detect violations, flaws, and faults in architecture design

The studies placed in this category [S8, S17, S22, S25, S34, S52] report VTs that provide visualization clues to architects to easily detect violations, flaws, and faults in architecture decompositions, architecture specifications, and architecture design. The work reported in [S8] presents an approach to visually compare architecture descriptions based on software decomposition and code dependencies. The approach represents similarities and differences between architecture descriptions that could reveal architectural drifts and violations and also shows how much two decompositions are (not) matched to each other at an appropriate level of detail. The studies reported in [S22, S25] use visualization to identify architectural violations in SA design. As discussed in **Category 5** (Section 3.8.5), the work reported in [S22] has developed a tool that represents software structure hierarchy and inner dependency in a unified view. The tool can help an architect to find what architectural violations exist against an intended architecture and assess how much effort is needed to solve these violations. The work reported in [S25] can prevent architecture from degenerating by identifying architectural violations between planned and actual architecture. The study described in [S25] shows architectural deviations on architecture diagrams to inform architects about the parts of the planned and actual architecture that are not compliant and provide guidelines to resolve it. The work reported in [S34] introduces an approach to integrate software system and software process artifacts. Thus, the study defines SA as architecture of software system and process. These artifacts are considered architectural artifacts and organized into a graph-based model. The approach provides some transformation models on the graph that architects can perform bad smell detection, pattern discovery, and architecture refactoring. The work reported in [S52] employs a formal language LePUS, which provides both symbolic and visual notations, to visually compare and symbolically verify the conformance between architecture design and architecture patterns.

3.8.7. Category 7: Provide traceability between architecture entities and software artifacts

The studies classified in this category are aimed at capturing and visualizing traceability links between requirements, design decisions and architecture artifacts [S1, S3, S5, S32, S35, S41]. Decision-centric-traceability (DCT) meta-model proposed in [S3] provides visual traceability links between quality goals/requirements and decisions, and also between decisions and architectural elements in design and implementation. DCT results in fewer traceability links and facilitates understanding the impact of a decision. DCT meta-model extracts traceability information from architectural analysis documents and transforms it into a reusable format. The work reported in [S5] uses ADD visualization to record traceability links between problem space and solution space, i.e., from requirement to architecture design or vice verse. By this way, the origins and targets of an ADD and its rationale are traceable. The work reported in [S32] provides traceability between design concerns, design decisions, and design outcomes. The design concerns are inputs or the causes of a decision, and design outcomes are the results or effects of a decision. Design concerns and outcomes are represented by architectural entity (AE) and decisions are shown by architectural rationale (AR). In other words, the proposed approach always keeps links between an input AE, a decision AR, and a resulting AE. The work reported in [S35] supports traceability visualization from architecture design to requirements or design decisions with an accompanying tool. Traceability links are visualized in architecture diagrams and in source code editors (i.e., Eclipse IDE) as well. The traceability visualization provided by [S41] can capture traces between requirements, decisions, and architectural components by means of formal and informal relationships. Formal relationships, which are defined in Archium meta-model, determine the impact of a decision, and also represent which components in architecture are related to a decision. Informal relationships represent other types of relationships between a decision and a requirement, and are defined as decision properties in text descriptions.

3.8.8. Category 8: Improve the understanding of behavioral characteristics of architecture

The work reported in [S19, S20, S51] focuses on visualization of behavioral characteristics of SA. This category emphasizes visualizing the characteristics of running components and monitor inter-component communication during runtime. The proposed notations in [S19] are used for describing both structural and behavioral properties of architecture. The behavioral properties focus on events and state changes of architectural elements. The work reported in [S20] developed a tool, *DiffArchViz*, to visualize dynamic SA. This tool was used to analyze the runtime profiles of a data storage server operating system against a set of storage server performance benchmarks. The work reported in [S20] does not explicitly discuss what architecting activities are supported by the dynamic architecture visualization, but it helps users to understand more details about SA, such as showing which components are the busiest and most active during specific benchmark runs. Besides visualization of static SA described in **Category 2** (Section 3.8.2), the VT reported in [S51] also visualizes the dynamic aspects of SA to support consistency between static and dynamic aspects of software architecture in one tool. The specific visualization notations proposed in [S51] can visualize dynamic architecture at various levels of abstractions. Software architects and developers can view the processes created at architecture level and can request information about the inter-process communication of a selected set of architectural elements.

3.8.9. Category 9: Check compatibilities and synchronization between architecture design and implementation

It is very well known that architecture design and implementation may not be compatible, especially when implementation evolves over time. The studies [S9, S25, S41] in this category focus on using visualization to verify designed architecture against implementation. The work reported in [S9] provides a visualization tool, *LISA*, with underlying meta-model to support continuous synchronization between architecture and implementation based on a defined meta-model. It means the tool provides real-time updates on the changes made at the architecture or implementation levels.

When architecture description and implementation are changed, the tool will annotate corresponding implementation elements and architectural elements that should be updated respectively. The tool always ensures compatibility between component-level abstraction and low-level architectural elements. The *SAVE (software architecture visualization and evaluation)* tool [S25] aims at checking the compliance between a planned architecture (i.e., planned architecture is defined based on the knowledge of a current system and it includes architectural goals and rationale) and an actual architecture (i.e., extracted from source code). The results of compliance checking are visualized on architecture diagrams. The *SAVE* tool claims to help keep an alignment between actual architecture and planed architecture during software evolution. The *Archium* tool [S41] can help architects to examine whether or not the agreed architecture decisions have been implemented. If the made architecture decisions are ignored, the *Archium* tool will warn the concerned stakeholders about the violations of architecture decisions.

3.8.10. Category 10: Support for model-driven development using architecture design

This category includes VTs aimed at supporting model-driven development (MDD) through architecture visualization, e.g., executable architectural view. Existing architectural views with viewpoints cannot be characterized and interpreted as executable models because they lack precision. Therefore, they cannot support MDD. The work reported in [S28] defines architectural viewpoints based on a domain specific language that provides executable architectural views. The *SAVE* tool developed in [S28] supports both textual and visual modeling of executable architecture views. The executable viewpoints and views in *SAVE* are not just documentation but represent executable models that can be interpreted, processed, and mapped onto other software models. Hence, these approaches support MDD based on architectural views.

3.9. Domain based architecture visualization techniques

The last two questions of our SLR, RQ4 and RQ5, were specifically designed for providing potentially useful information for practitioners. The results are expected to be quite beneficial to those who are interested in the domain specific aspects of the architecture visualization techniques. To answer RQ4, "*What are the domains in which architecture visualization techniques have been applied to support architecting activities*?", we analyzed the data item D13 in Table 5. Table 13 shows the application domains in which the reviewed VTs can be placed. As shown in Table 13, 11 studies did not explicitly report the application domains for which the architecture visualization techniques have been developed and/or employed. The remaining studies have been classified in 18 application domains. Note that, the VT introduced in one study can be applied in more than one application domains with several cases (e.g., the VT reported in [S40] has been applied in six different domains such as graphics software (including both image editing software and visualization tools), information management system, operating system, web server software, and email server software) and if one study uses more than one system as a case study, then we count this study N (number of systems) times in Table 13 and 11 (e.g., the work reported in [S8] uses two distributed systems and two general software libraries as case studies, and x represents the number of cases in S8(x)). It is clear from our findings that graphics software, distributed system, and information management system have gained more attention in applying VTs to their SA. For example, the studies [S3] and [S45] use NASA crew exploration vehicle system and Mozilla web browser software as case studies to validate their proposed VTs.

Table 14 shows that how VTs have been employed for what purposes (i.e., the 10 purposes of using VTs categorized and discussed in Section 3.8) and in which application domains. The results reported in Table 14 have been extracted and combined from Fig. 8 and Table 13 in Sections 3.8 and 3.9. According to Table 13, the VTs reported in [S6, S7, S16] use architecture visualization in "*web browser software*" domain, and according to Fig. 8 and Section 3.8, the studies [S6, S7, S16] use VTs to "improve search, navigation, and exploration of architecture design", therefore, we put "3" in Table 14, which means 3 studies have applied architecture visualization techniques in "web browser software" domain to "improve search, navigation, and exploration of architecture design". It is clear from Table 14 that the objectives "improve search, navigation, and exploration of architecture design" and "support architecture reengineering and reverse engineering" of using architecture visualization have been supported in most of application domains. The VTs applied in "distributed system" domain support all objectives except two (i.e., "improve the understanding of behavioral characteristics of architecture" and "support for model-driven development using architecture design").

3.10. Most frequently used visualization techniques in industry

Our RQ5, "*Which are the architecture visualization techniques mostly used in industry?*", was aimed at revealing the number of times a particular VT had been applied to an industrial environment. Table 15 presents the primary studies along with the domain and the number of times a particular VT has been applied to different domains. It becomes clear from Table 15 that the reported industrial practices have been validated in 16 application domains. Only two application domains ("telecommunication system" and "aerospace system") in Table 13 do not have any validations of using VTs in SA in an industrial setting. The "graphics software" domain has gained the most attention in applying VTs to architecture in industry. One possible reason can be that "graphics software" has more interest to visualize architecture design because of its nature. A VT proposed in a study represented as [X] in Table 15 (for example, S18 [NV] means study [S18] employs a notation-based VT. Because of the limited space in the table, we use the abbreviations of the visualization techniques proposed in Fig. 6). By investigating the proposed VTs reported in the studies in Table 15, it is clear that graph-based VT (38 cases) is the dominant and most popular architecture visualization technique used in industry, and matrix-based VT (9 cases), notation-based VT (7 cases), and metaphor-based VT (6 cases) are used but less popular in industry. The results reveal that the distribution of VTs in SA used in industry is different from the overall distribution of VTs in the selected studies described in Section 3.3. As shown in Fig. 3, the notation-based VT is in the second place in terms of popularity with 22 studies, but as shown in Table 15, the notation-based VTs are applied to 6 industrial cases, which places the notation-based VTs in the last place (i.e., graph-based, matrix-based, and metaphor-based VTs are used in 38, 9, and 6 industrial cases respectively) in industrial applications. Both distributions have some similarities, for example, graph-based VT is still dominant in both industrial and overall applications, because graph-based VT intends to show the overall properties of a structure, which is useful for all types of projects to get an overview of the architecture.

4. Threats to validity

This SLR has two main threats to validity: bias in the study selection and bias in the data extraction. Because the study selection and data extraction are quite subjective, researchers' bias could affect the results of this SLR. To reduce the potential bias in the

Table 13
Distribution of application domains of the selected studies.

Application domain	No. of cases	Cases
Not specified	11	S13, S14, S19, S21, S28, S29, S33, S34, S37, S47, S51
Graphics software	12	S2, S18, S22, S27(2), S39, S40(2), S44(2), S48, S49
Distributed system	9	S1, S8(2), S11, S12, S24(2), S41, S50
Information management system	8	S4, S11(2), S30, S31, S35, S36, S40
General software library	5	S8(2), S42(2), S52
Web browser software	4	S6, S7, S16, S45
Operating system	4	S15, S16, S20, S40
Web server software	4	S8, S16, S40, S53
Aerospace system	3	S3, S25, S38
Text editor	3	S8, S11, S16
Financial software	3	S9, S22, S32
Software modeling tool	3	S11, S12, S44
Database system	3	S23, S46, S50
Embedded software	2	S5, S10
Email server software	2	S11, S40
Compiler software	2	S26, S43
Software development tool	2	S22, S48
Telecommunication system	1	S17
Spreadsheet software	1	S16

study selection as much as possible, we developed and validated the review protocol in the following way. This protocol was initially defined by two researchers and was further evaluated by a third researcher on software architecture who had an extensive experience of research and practice on empirical software engineering and systematic literature review. After finalizing the review protocol, we strictly followed it for conducting the review. To further mitigate the effect of researchers' bias in the study selection, we chose a three-step (round) study selection process so that there is less likely chance of excluding the relevant studies unintentionally. The first author performed the study selection process and the second author examined all the included and excluded studies from the second step of the selection process as described in Section 2.4. For the input of the first round of paper selection, we got 23,056 papers (as shown in Fig. 1), and we decided to select papers based on the title and keywords only because reading abstract of all the papers will cost a huge amount of effort, but the threat is that we may have excluded some relevant papers. In the second step of the study selection process, the reasons for inclusion or exclusion decisions for each paper were recorded and any disagreements were resolved through discussions (there were 18 papers that the two authors had different opinions on whether including or excluding them and finally the disagreements were resolved through discussions between the two authors). A cross-check using a random number of the selected papers was performed by the third author. The search strategy includes both automatic and manual searches. The automatic search was limited to title, abstract, and keywords of papers to get a reasonable number of potentially relevant papers. Search terms were improved iteratively based on the trial search and were tested carefully before executing the review. Despite this, it is possible that some relevant papers might not have been retrieved by the used search terms. We employed two ways to mitigate this threat: (1) a manual search was conducted to complement the results from the automatic search; (2) a snowballing technique was employed in the second round of the study search process (see Fig. 1) to identify related studies. According to the data sources for automatic and manual searches listed in Section 2.2.3, we might have two threats, which may cause the missing of related studies: (1) we did not include Google Scholar as a data source for the automatic search; (2) the list of target venues for the manual search in Table 3 might not be comprehensive due to the limitation of our knowledge. The later threat is partially mitigated by including the general intervention terms "architecture" and "visualization" in the search terms for the automatic search.

To reduce the threat of inaccuracies in the data extraction, we created a data extraction form to extract and analyze the needed data consistently for answering the research questions of this SLR. To reduce the bias, the data extraction process was conducted by two researchers, the first author acted as the data extractor, and the second author as the data checker of the selected studies. Any disagreements among all the researchers were resolved through discussions. Because of the page limitation of several included studies, especially those published in conferences and workshops, they have not provided enough and detailed information about the data items to be extracted. In this kind of situations, a subjective interpretation about the extracted data by researchers was needed. It might have had an effect on the data extraction outcomes. For example, some studies did not discuss the level of tool support, and some studies did not explicitly discuss what architecting activities and in which application domains the VTs had been applied. To reduce the researchers' bias in interpretation of the results, in the case of the level of tool support, besides reading the given study, we also referred the tool's website and any training movie (if it was available) to get more reliable information. In the case of architecting activities and application domains, we did not have any appropriate interpretation unless we were sure that the proposed VT supports certain architecting activities and has been applied in a specific application domain. The study quality assessment presented in Section 3.2 also helped us to increase the correctness and accuracy of the data extraction results because as we discussed in Section 3.2, the quality assessment ensures that the extracted data are from credible studies.

5. Discussions

There has been an increasing interest in approaches and tools for understanding and reasoning about software architecture of that system. Visualization approaches are increasingly becoming popular for communicating and understanding software architecture of a system. Several dozens of approaches and tools have been developed and reported for visualizing architectural elements (i.e., components, connectors, design patterns, and rationale) of a software intensive system. It is equally important to systematically review and thoroughly document the reported software architecture visualization approaches and tools in form that can help understand their nature and potential areas of applications and identify the areas of future research direction. This study was motivated to address the abovementioned needs that were described as the five key research questions. The previously section presented

Table 14
Number of studies assigned to each application domain and purpose of using visualization techniques.

Purpose of using visualization techniques	Application domain								
	Graphics software	Distributed system	Information management system	General software library	Web browser software	Operating system	Aerospace system	Text editor	Web server software
Category 1: Improve the understanding of architecture evolution	3	2	1		1		1		
Category 2: Improve the understanding of static characteristics of architecture	3	2	3	1		1			2
Category 3: Improve search, navigation, and exploration of architecture design	2	1		1	3	2		1	1
Category 4: Improve the understanding of architecture design through design decision visualization		2	3				2		
Category 5: Support architecture reengineering and reverse engineering	2	3	1	1				1	
Category 6: Detect violations, flaws, and faults in architecture design	1	1		2			1	1	1
Category 7: Provide traceability between architecture entities and software artifacts		2	1				1		
Category 8: Improve the understanding of behavioral characteristics of architecture						1			
Category 9: Check compatibilities and synchronization between architecture design and implementation		1					1		
Category 10: Support for model-driven development using architecture design									

Purpose of using visualization techniques	Application domain									
	Financial software	Software modeling tool	Database system	Embedded software	Email server software	Compiler software	Software development tool	Telecommunication system	Spreadsheet software	Not specified
Category 1: Improve the understanding of architecture evolution	1	2	2			1				1
Category 2: Improve the understanding of static characteristics of architecture	1				1					1
Category 3: Improve search, navigation, and exploration of architecture design			1	1		1	1		1	3
Category 4: Improve the understanding of architecture design through design decision visualization	1			1						2
Category 5: Support architecture reengineering and reverse engineering	1	2	1		1		1	1		
Category 6: Detect violations, flaws, and faults in architecture design	1						1	1		1
Category 7: Provide traceability between architecture entities and software artifacts	1			1						
Category 8: Improve the understanding of behavioral characteristics of architecture										2
Category 9: Check compatibilities and synchronization between architecture design and implementation	1									
Category 10: Support for model-driven development using architecture design										1

Table 15
Frequencies of the reported VTs applied to different domains.

Application domain	No. of cases	Cases
Graphics software	9	S18 [NV], S22 [GV], S27(2) [NV, GV], S40(2) [GV], S44(2) [MEV], S48 [MEV]
Distributed system	7	S8(2) [MV], S11 [GV], S12 [MV, GV], S24(2) [GV], S50 [GV]
Information management system	6	S4 [NV], S11(2) [GV], S31 [GV], S35 [NV], S40 [GV]
General software library	4	S8(2) [MV], S42(2) [GV]
Web browser software	4	S6 [MEV], S7 [GV], S16 [GV], S45 [GV]
Operating system	4	S15 [GV], S16 [GV], S20 [GV], S40 [GV]
Web server software	4	S8 [MV], S16 [GV], S40 [GV], S53 [NV]
Text editor	3	S8 [MV], S11 [GV], S16 [GV]
Software modeling tool	3	S11 [GV], S12 [MV, GV], S44 [MEV]
Database system	3	S23 [GV], S46 [GV], S50 [GV]
Financial software	2	S22 [GV], S32 [NV]
Email server software	2	S11 [GV], S40 [GV]
Software development tool	2	S22 [GV], S48 [MEV]
Telecommunication system	1	S17 [MV]
Compiler software	1	S43 [GV]
Spreadsheet software	1	S16 [GV]

the findings from this literature review with respect to those key research questions. We discuss the findings from this systematic literature review and reflect upon the potential areas for further research in architecture visualization techniques and tools:

(1) **Architecting activities and VTs**: According to the results reported in Section 3.4, the VTs support architecture recovery (AR) and architectural evolution (AEV) activities frequently, and support architectural evaluation (AE), change impact analysis (CIA), and architectural analysis (AA) activities to some extent, but they rarely support architectural synthesis (AS), architectural implementation (AI), and architecture reuse (ARU) activities. We assert that for the architecting activities such as AR, AEV, AE, and CIA, the value of "understanding" of architectural elements including structural and decisional elements is more visible and therefore in these activities architects need more "visualization" support (as we know, the dominant function of visualization is to improve understanding). For example, to evolve a given software architecture (AEV), an architect needs to understand the structure of software, and then she/he can change the architecture in an informed way. To perform AR, as a sub-activity in reverse/re-engineering, it is necessary to get a clear understanding of a system before performing other reverse/re-engineering activities. We argue that the research area (i.e., applying VTs in AA, AS, AI, and ARU activities) should be more investigated in the future, for example, whether these architecting activities can be well-supported by SA visualization? What are the underlying reasons that these architecting activities receive little attention of using SA visualization?

(2) **VTs in SA and tool support**: Table 16 shows what VTs for SA visualization are provided with which level of tool support. To give an example, the study reported in [S34] uses graph-based VT with manual level of tool support in software architecting. Note that the studies, which provide a certain level of tool support to more than one VTs (e.g., S12), are only counted once to the number of "Total Studies" in Table 16. All the four VTs are mainly provided with "semi-automatic" and "automatic" tool support. It shows that (semi-)automatic tool support is an important driver to employ VTs in SA. It is clear from Table 16 that matrix- and metaphor-based VTs do not fall under the manual level of tool support, and manual tools mostly support notation-based VT (5 out of 6 studies). Notation-based VT can be both used in manual and (semi-)automatic tools, e.g., manual ADD visualization using ADD notations has been reported in [S5], and automated UML model construction from source code has been reported in [S25]. We further investigated what differences exist in the functionality provided by semi-automatic tools and automatic tools in SA visualization. We identified three main features that were performed manually in semi-automatic tools compared to automatic tools in SA visualization:

- **User involvement**: in some studies, there is a need of user involvement such as user analysis, user inspection, and user decision that can provide users with the knowledge to produce better results. The user involvement makes architecture visualization clearer, more understandable, and accurate. For example, Sartipi and Kontogianni [S40] provide a user-assisted technique for components clustering to better partition large systems. The study in [S11] has employed a graph-based VT to assist reverse engineers during package decomposition of systems, but the system decomposition process in [S11] is not fully automated because there are some cases in which user decisions are needed to choose which packages should (not) be further decomposed.
- **User configuration**: some semi-automatic architecture visualization tools require users to input initial data and provide

Table 16
Classification of studies by visualization technique and level of tool support.

	Level of tool support			Total studies
Visualization technique	Manual	Semi-automatic	Automatic	
Graph-based	S34 [1 Study]	S7, S10, S11, S12, S20, S23, S24, S29, S31, S38, S40, S43, S50 [13 Studies]	S9, S15, S16, S22, S26, S27, S36, S37, S42, S45, S46, S49 [12 Studies]	26
Notation-based	S1, S2, S5, S19, S39 [5 Studies]	S21, S28, S32, S33, S35, S41, S47, S51, S52, S53 [10 Studies]	S3, S4, S9, S14, S18, S25, S27 [7 Studies]	22
Matrix-based		S12, S30 [2 Studies]	S8, S17, S37 [3 Studies]	5
Metaphor-based		S44 [1 Study]	S6, S13, S48 [3 Studies]	4
Total studies	6	25	22	

initial configurations to be triggered for visualization. In [S38 and S41], the initial data could be ADDs, their attributes, and the relations between them, which are manually defined and provided by architects, and then the tools developed in [S38 and S41] can automatically visualize these ADDs with other features. The work in [S28] developed a tool to visualize executable architectural views for model-driven development, but it requires that architects write a grammar for constructing and visualizing the architectural views.

- **User layout**: In some studies, the layout of architecture visualization output is not supported by visualization tools sufficiently. A user should manually adjust the layout. For example, the semi-automatic tool reported in [S52] provides a basic layout algorithm, therefore, some manual layout is needed.

(3) **Architecting activities and tool support**: Table 17 shows the architecting activities that are supported by the tools reported in the studies included in this SLR. Table 17 also shows the type of support (i.e., manual, semi-automatic, and automatic) for SA visualization provided by each of the tools. To give an example, the work reported in [S7] uses a semi-automatic tool to support the AR activity. Similarly, the studies, which provide a certain level of tool support to more than one architecting activities (e.g., S30), are only counted once to the number of "Total Studies" in Table 17. As shown in Table 17, all the architecting activities are mainly facilitated by "semi-automatic" and "automatic" types of tool support. It has also been shown that (semi-)automatic tools are fundamentally employed by VTs in software architecting. SA visualization for AR and AEV are more facilitated by (semi-)automatic tools, but SA visualization for AI and ARU are less supported by tools. AA, AE, and AS are mostly supported by semi-automatic tools. One reason that AA, AS, and AE are mostly (over 61.5%) supported by semi-automatic visualization tools can be that these architecting activities are mainly human-centric and user involvement is needed, such as inspection in AA, decisions making in AS, and examination in AE.

(4) **Tool support of VTs and evidence level**: Table 18 summarizes the findings from our investigation whether the architecture visualization techniques supported by (semi-)automatic tools have a higher level of evidence (mostly in level 4, industrial studies) than those that only have manual support (mostly in level 1, toy examples/demonstration). The level of tool support provided in a study represented as (*X*) in Table 18 (For example, S9(A) means [S9] employs an automatic tool to support graph-based VT in a toy example, i.e., evidence level 1. The abbreviations used in the Table are as follows: automatic "A", semi-automatic "S", and manual "M"). The studies that

Table 17
Classification of studies by architecting activity and level of tool support.

	Level of tool support			Total studies
Architecting activity	Manual	Semi-automatic	Automatic	
Architecture recovery		S7, S11, S12, S20, S23, S24, S40, S43, S44, S47, S50 [11 Studies]	S6, S8, S13, S15, S16, S17, S22, S25, S27, S42, S45, S46, S48, S49 [14 Studies]	25
Architectural evolution	S1, S39 [2 Studies]	S12, S23, S32, S33, S43, S44 [6 Studies]	S14, S17, S22, S25, S27, S36, S45, S46 [8 Studies]	16
Architectural evaluation	S1, S5 [2 Studies]	S30, S31, S32, S33, S35, S38, S41 [7 Studies]	S4, S36 [2 Studies]	11
Change impact analysis	S1 [1 Study]	S30, S32, S35, S41 [4 Studies]	S22, S25, S27, S36, S37 [5 Studies]	10
Architectural analysis	S1, S5 [2 Studies]	S30, S31, S32, S33, S35, S53 [6 Studies]	S3 [1 Study]	9
Architectural synthesis	S1, S5 [2 Studies]	S30, S31, S32, S33 [4 Studies]	S4 [1 Study]	7
Architectural implementation		S28,S41,S51 [3 Studies]	S9,S25 [2 Studies]	5
Architecture reuse		S29, S30, S31 [3 Studies]	S36 [1 Study]	4
Total studies	6	25	22	

Table 18
Classification of studies by visualization technique, level of tool support, and evidence level.

	Evidence level					Total studies
Visualization technique	Level 1	Level 2	Level 3	Level 4	Level 5	
Graph-based	S9(A), S10(S), S26(A), S29(S), S34(M), S36(A), S38(S), S49(A) [8 Studies]	S37(A) [1 Study]		S7(S), S11(S), S12(S), S15(A), S16(A), S20(S), S22(A), S23(S), S24(S), S27(A), S31(S), S40(S), S42(A), S43(S), S45(A), S46(A) [16 Studies]	S50(S) [1 Study]	26
Notation-based	S1(M),S2(M),S3(A), S9(A),S14(A),S19(M), S21(S),S25(A),S28(S), S39(M),S41(S),S47(S), S51(S),S52(S) [14 Studies]		S5(M), S53(S) [2 Studies]	S4(A), S18(A), S27(A), S32(S), S33(S), S35(S) [6 Studies]		22
Matrix-based	S30(S) [1 Study]	S37(A) [1 Study]		S8(A), S12(S), S17(A) [3 Studies]		5
Metaphor-based	S13(A) [1 Study]			S6(A), S44(S), S48(A) [3 Studies]		4
Total Studies	23	1	2	26	1	

provide a certain level of evidence to more than one visualization techniques (e.g., S37), are only counted once to the number of "Total Studies" in Table 18. Table 18 indicates that a manual level of tool support is mostly provided in studies whose evidential support is based on toy examples (i.e., evidence level 1). Moreover, the tools presented in these studies have never been used in industrial settings. This may indicate that the VTs in SA supported by manual level tools are not interesting for industrial settings. We also found that 38% of the studies reporting (semi-)automatic tool support have provided the evidence based on toy examples and expert opinions/observations (i.e., evidence level 1 and level 2), and there are 59% of the studies with (semi-)automatic tool support that have provided the evidence based on industrial studies and industrial practices (i.e., evidence level 4 and level 5). As shown in Table 18, there is a correlation between the level of tool support and the level of evidence provided, that VTs in SA supported by (semi-)automatic tools receive a higher level of evidential support (i.e., 26 studies with (semi-)automatic tool support were validated with industrial studies, evidence level 4), and toy examples (i.e., 23 studies with evidence level 1) are usually used to demonstrate how to use the VTs in SA.

6. Conclusions

The aim of this work is to report the design, execution, and results of a systematic review of software architecture visualization techniques. We systematically selected and rigorously analyzed a comprehensive set of SA visualization techniques and tools in order to provide an evidential based knowledge about the current state of SA visualization and the potential areas of research. From the initially identified 23,056 papers through manual and automatic searches, we selected 53 papers based on the inclusion and exclusion criteria for this review. The data extraction from the 53 papers and synthesis have enabled us to draw the following conclusions:

(1) The results of this SLR show that the reported software architecture visualization techniques can mainly be classified into four types: graph-based, notation-based, matrix-based, and metaphor-based. The graph-based and notation-based VTs appear to be the most popular ones. The architecting activities including general and specific activities are supported by these VTs with a significant difference in terms of the popularity. Whilst AR and AEV are supported more frequently by the VTs, however, AA, AS, AI, and ARU receive less attention from using VTs. We can also conclude that AR and AEV are more facilitated by (semi-)automatic tools, but SA visualization for AI and ARU are less supported by tools. Our review also reveals that AA, AS, and AE are more dependent on human involvement than other architecting activities, therefore, these activities are mostly supported by semi-automatic tools. Manual tools for architecture visualization mostly employ notation-based VT, but they receive less attention from practitioners and have not been trialed in industrial applications.

(2) Our review enables us to conclude that VTs are mainly employed in SA for ten purposes, in an order of importance: (i) improving the understanding of architecture evolution; (ii) improving the understanding of static characteristics of architecture; (iii) improving search, navigation, and exploration of architecture design; (iv) improving the understanding of architecture design through design decision visualization; (v) supporting architecture reengineering and reverse engineering; (vi) detecting violations, flaws, and faults in architecture design; (vii) providing traceability between architecture entities and software artifacts; (viii) improving the understanding of behavioral characteristics of architecture; (ix) checking compatibilities and synchronization between architecture design and implementation; (x) supporting for model-driven development using architecture design.

(3) VTs have been applied to the SA in a wide range of application domains, and the "graphics software" and "distributed system" domains have received the most attention in industry. This systematic review also shows that graph-based VT is the most popular architecture visualization technique in industry; other VTs are less employed.

(4) A high percentage of papers providing industrial level evidence (i.e., industrial studies and practices as presented in Section 3.7) and tool support (Section 3.6) in the selected studies with the moderate quality (Section 3.2) improves the practical applicability of the reported results and encourages SA practitioners to adopt and employ these VTs in their daily work. But academic and industrial controlled experiments, as one of objective evaluation methods in empirical software engineering, are rarely used in the evaluation of the selected studies. This deficiency has also been demonstrated in secondary study on software evolution visualization (Novais et al., 2013). We encourage researchers and practitioners in SA community to pay more attention to this evaluation method and other high quality empirical methods (Sjoberg et al., 2007) for gathering and sharing more convincing evidence of the potential benefits of using VTs in SA.

(5) A comparative analysis of semi-automatic and automatic SA visualization tools reveals that these tools can be categories based on three types of features: "user involvement", "user configuration", and "user layout", which are manually performed in semi-automatic tools. The findings from this review also enable us to conclude that there is more likelihood to gather and provide higher level of evidence for architecture visualization techniques that have associated (semi-)automatic tools compared with the techniques which needs to be applied manually. One reason for this situation may be that practitioners prefer those VTs which have (semi-)automatic tool support compared with manual tool support that may need much more efforts to use them.

The results of this SLR can have several implications for software architecture and visualization researchers, practitioners, and tool developers. For researchers, this review has identified a number of potentially researchable topics in the area of SA visualization. For example, (i) there is a paucity of research aimed at systematically studying the pros and cons of applying VTs for architectural analysis, architectural synthesis, architectural implementation, and architecture reuse activities (as demonstrated in Table 10); (ii) little attention has been paid to carry out controlled experiments (one of objective evaluation methods used in empirical software engineering) to evaluate and compare the usefulness (e.g., ease of use, effectiveness, efficiency, application cost) of the reported VTs and tools due to the excessive effort and resources needed (Sjoberg et al., 2005). Hence, researchers need to systematically design and rigorously carry out suitable controlled experiment to collect more objective and quantifiable evidence about the usefulness of reported VTs and tools; (iii) as reported in Section 3.2, the majority (49%) of the included studies did not examine the potential researchers bias and influence on the findings; there is also a general lack of discussions on the limitations of the reported techniques and tools in the reviewed papers; we strongly suggest that more attention should be paid to these issues by researchers which is likely to improve the quality and credibility of the reported VTs in SA; (iv) since architectural implementation and architecture reuse activities are less supported by the reported SA visualization tools, researchers

may consider to allocate more research and development effort to provide appropriate techniques and tools for supporting these activities.

The findings from this review also provide potentially useful insights for practitioners and tools developers. For example, (i) since VTs without (semi-)automatic tool support have not been trialed in industrial applications in order to provide strong evidential support, SA visualization tools developers ought to propose VTs with (semi-)automatic tool support in order to encourage industry to try their VTs in architecting practice; (ii) as discussed in Section 5, "user involvement", "user configuration", and "user layout" are three main features that were performed manually in semi-automatic tools compared to automatic tools in SA visualization. We suggest that these features should better be supported or assisted by the future automatic tools. For example, with increasing size and complexity of software systems, layouting SA design properly can be out of the capability of architects. To this end, we suggest that practitioners, who work on developing and using SA visualization tools, employ more efficient layout algorithms in SA visualization; (iii) our dataset revealed that there are only 9.4% of studies (5 out of 53, [S3, S32, S35, S39, S41]) which provide tools to support architecture visualization of both structural and decisional viewpoints in an integrated tool. The possible reasons for this low attention could be that architectural design decision and decisional viewpoint are emerging topics (Jansen and Bosch, 2005; Duenas and Capilla, 2005) in SA community, and documenting architectural design decision necessitates high effort as well as establishing the links between decisions and other software artifacts (e.g., structural view) is time-consuming (Zdun et al., 2013). Since SA practitioners confront both structural and decisional elements of architecture during architecting process, which are intertwined; it is recommended that tools vendor develop the VTs with SA visualization tools that support simultaneously the visualization of structural elements and decisional elements in one tool; (iv) this review has found that only one of the reported VTs, has been employed in industrial practice, which indicates that the application of VTs in SA of real-world industrial systems is quite limited. Accordingly, we suggest that researchers and practitioners can cooperate closely to conduct industrial surveys and collect empirical data on how SA practitioners actually employ VTs in architecting process and what are the issues that hinder and prevent them from adopting VTs in SA; (v) SA practitioners can use the results of this SLR to get information about existing VTs and tools in SA and their features, which may help them to select appropriate VTs and tools for a given application context. Furthermore, the results of the review are also expected to stimulate new research and development efforts for developing better architecture visualization techniques and tools, and facilitating the use of SA visualization in industry.

Acknowledgement

This research has been partially sponsored by the Natural Science Foundation of China (NSFC) under Grant No. 61170025.

Appendix A. Selected studies

See Table 19.

Table 19
Selected studies in the review.

ID	Title	Author(s)	Venue	Year
S1	Concepts and diagram elements for architectural knowledge management	B. Orlic, R. Mak, I. David, J. Lukkien	ECSA Companion	2011
S2	Modeling the variability of architectural patterns	A.W. Kamal, P. Avgeriou	SAC-SE	2010
S3	Transforming trace information in architectural documents into re-usable and effective traceability links	M. Mirakhorli, J. Cleland-Huang	SHARK	2011
S4	Evaluating software architecture in a model-based approach for enterprise information system design	A. Tsadimas, M. Nikolaidou, D. Anagnostopoulos	SHARK	2010
S5	Architectural design decision visualization for architecture design: preliminary results of a controlled experiment	M. Shahin, P. Liang, Z. Li	ECSA Companion	2011
S6	EvoSpaces: 3D visualization of software architecture	S. Alam, P. Dugerdil	SEKE	2007
S7	Visual exploration of function call graphs for feature location in complex software systems	J. Bohnet, J. Döllner	SoftVis	2006
S8	Visual comparison of software architectures	F. Beck, S. Diehl	SoftVis	2010
S9	Tool support for component-based software architectures	G. Buchgeher, R. Weinreich	APSEC	2009
S10	KaitoroBase: visual exploration of software architecture documents	M.T. Su, C. Hirsch, J. Hosking	ASE	2009
S11	Package patterns for visual architecture recovery	M. Lungu, M. Lanza, T. Girba	CSMR	2006
S12	Exploring inter-module relationships in evolving software systems	M. Lungu, M. Lanza	CSMR	2007
S13	Communicating software architecture using a unified single-view visualization	T. Panas, T. Epperly, D. Quinlan, A. Saebjornsen, R. Vuduc	ICECC	2007
S14	Evolve: tool support for architecture evolution	A. McVeigh, J. Kramer, J. Magee	ICSE	2011
S15	Browsing and searching software architectures	S.E. Sim, C.L. Clarke, R.C. Holt, A.M. Cox	ICSM	1999
S16	Browsing software architectures with LSEdit	N. Synytskyy, R.C. Holt, I. Davis	IWPC	2005
S17	An architectural connectivity metric and its support for incremental re-architecting of large legacy systems	R.J. Bril, A. Postma	IWPC	2001
S18	Visualizing metrics on areas of interest in software architecture diagrams	H. Byelas, A. Telea	PacificVis	2009
S19	Visual software architecture description based on design space	Q. Zhang	QSIC	2008
S20	DiffArchViz: a tool to visualize correspondence between multiple representations of a software architecture	A.P. Sawant, N. Bali	VISSOFT	2007
S21	Visual exploration of combined architectural and metric information	M. Termeer, C.F.J. Lange, A. Telea, M.R.V. Chaudron	VISSOFT	2005
S22	A visual analysis and design tool for planning software reengineering	M. Beck, J. Trumper, J. Dollner	VISSOFT	2011
S23	YARN: animating software evolution	A. Hindle, Z.M. Jiang; W. Koleilat, M.W. Godfrey, R.C. Holt	VISSOFT	2007
S24	Reverse engineering object oriented distributed systems	D. C. Cosma, R. Marinescu	ICSM	2010
S25	The SAVE tool and process applied to ground software development at JHU/APL: an experience report on technology infusion	W.C. Stratton, D.E. Sibol, M. Lindvall, P. Costa	SEW	2007
S26	Navigating software architectures with constant visual complexity	W. Li, P. Eades, S.H. Hong	VLHCC	2005
S27	Visualizing software architecture evolution using change-sets	A. McNair, D.M. German, J. Weber-Jahnke	WCRE	2007
S28	SAVE: software architecture environment for modeling views	E. Demirli, B. Tekinerdogan	WICSA	2011

Table 19 (*Continued*)

ID	Title	Author(s)	Venue	Year
S29	KaitoroCap: a document navigation capture and visualization tool	M.T. Su; J. Hosking, J. Grundy	WICSA	2011
S30	Ontology-driven visualization of architectural design decisions	R.C. de Boer, P. Lago, A. Telea, H. van Vliet	WICSA/ECSA	2009
S31	Visualization and comparison of architecture rationale with semantic web technologies	C. López, P. Inostroza, L.M. Cysneiros, H. Astudillo	JSS	2009
S32	A rationale-based architecture model for design traceability and reasoning	A. Tang, Y. Jin, J. Han	JSS	2007
S33	Capturing architecture evolution with maps of architectural decisions 2.0	A. Zalewski, S. Kijas, D. Sokoowska	ECSA	2011
S34	Software Is a directed multigraph	R. Dabrowski, K. Stencel, G. Timoszuk	ECSA	2011
S35	Integrating requirements and design decisions in architecture representation	R. Weinreich, G. Buchgeher	ECSA	2010
S36	Guiding architects in selecting architectural evolution alternatives	S. Ciraci, H. Sözer, M. Aksit	ECSA	2011
S37	A tool to visualize architectural design decisions	L. Lee, P. Kruchten	QoSA	2008
S38	Building up and reasoning about architectural knowledge	P. Kruchten, P. Lago, H. van Vliet	QoSA	2006
S39	Viability for codifying and documenting architectural design decisions with tool support	R. Capilla, J.C. Dueñas, F. Nava	SMPR	2010
S40	A user-assisted approach to component clustering	K. Sartipi, K. Kontogianni	SMPR	2003
S41	Tool support for architectural decisions	A. Jansen, J. van der Ven, P. Avgerious, D. Hammer	WICSA	2007
S42	Exploring relations within software systems using treemap enhanced hierarchical graphs	M. Balzer, O.Deussen	VISSOFT	2005
S43	An integrated approach for studying architectural evolution	Q. Tu, M.W. Godfrey	IWPC	2002
S44	Visual exploration of large-scale system evolution	R. Wettel, M. Lanza	WCRE	2008
S45	EvoGraph: a lightweight approach to evolutionary and structural analysis of large software systems	M. Fischer, H. Gall	WCRE	2006
S46	Evolution storyboards: Visualization of software structure dynamics	D. Beyer, A. E. Hassan	IWPC	2006
S47	SAB- The software architecture browser	N. Erben, K.P. Löhr	SoftVis	2005
S48	Software landscapes: visualizing the structure of large software systems	M. Balzer, A. Noack, O. Deussen, C. Lewerentz	VisSym	2004
S49	SHriMP views: an interactive environment for exploring java programs	M.D. Storey, C. Best, J. Michuand	IWPC	2001
S50	Rigi - An environment for software reverse engineering, exploration, visualization, and redocumentation	H.M. Kienle, H.A. Müller	SCP	2010
S51	High-level static and dynamic visualization of software architectures	J. Grundy, J. Hosking	VL	2000
S52	Visualization of object oriented architectures	A.H. Eden	ICSE Workshop	2001
S53	An experiment on the role of graphical elements in architecture visualization	J. Knodel, D. Muthig, M. Naab	ESE	2008

Appendix B. Abbreviations used in the review

AA	Architectural analysis
ADD	Architectural design decision
AE	Architectural evaluation
AEV	Architectural evolution
AI	Architectural implementation
AR	Architecture recovery
ARU	Architecture reuse
AS	Architectural synthesis
ASR	Architecturally significant requirement
CIA	Change impact analysis
EBSE	Evidence-based software engineering
GV	Graph-based visualization
MEV	Metaphor-based visualization
MV	Matrix-based visualization
NV	Notation-based visualization
RQ	Research question
SA	Software architecture
SAD	Software architecture document
SysML	Systems modeling language
UML	Unified modeling language
VT	Visualization technique

References

Alves, V., Niu, N., Alves, C., Valença, G., 2010. Requirements engineering for software product lines: a systematic literature review. Inform. Softw. Technol. 52 (8), 806–820.

Balzer, M., Noack, A., Deussen, O., Lewerentz, C., 2004. Software landscapes: visualizing the structure of large software systems. In: Proceedings of the 6th Joint Eurographics-IEEE TCVG Conference on Visualization (VisSym), pp. 261–266.

Bass, L., Clements, P., Kazman, R., 2012. Software Architecture in Practice, 3rd ed. Addison Wesley, Boston.

Bengtsson, P., Bosch, J., 1998. Scenario-based software architecture reengineering. In: Proceedings of the 5th International Conference on Software Reuse (ICSR), pp. 308–317.

Braun, V., Clarke, V., 2006. Using thematic analysis in psychology. Qual. Res. Psychol. 3 (2), 77–101.

Budgen, D., Turner, M., Brereton, P., Kitchenham, B., 2008. Using mapping studies in software engineering. In: Proceedings of the 20th Annual Meeting of the Psychology of Programming Interest Group (PPIG), pp. 195–204.

Chen, L., Babar, M.A., Zhang, H., 2010. Towards an evidence-based understanding of electronic data sources. In: Proceedings of the 14th International Conference on Evaluation and Assessment in Software Engineering (EASE), pp. 135–138.

Chikofsky, E.J., Cross, J.H., 1990. Reverse engineering and design recovery: a taxonomy. IEEE Softw. 7 (1), 13–17.

Cleland-Huang, J., Hanmer, R.S., Supakkul, S., Mirakhorli, M., 2013. The twin peaks of requirements and architecture. IEEE Softw. 30 (2), 24–29.

Cornelissen, B., Zaidman, A., van Deursen, A., 2011. A controlled experiment for program comprehension through trace visualization. IEEE Trans. Softw. Eng. 37 (3), 341–355.

de Boer, R.C., Lago, P., Telea, A., van Vliet, H., 2009. Ontology-driven visualization of architectural design decisions. In: Proceedings of the Joint 8th Working IEEE/IFIP Conference on Software Architecture & 3rd European Conference on Software Architecture (WICSA/ECSA), pp. 51–60.

Diehl, S., 2007. Software Visualization Visualizing the Structure, Behaviour, and Evolution of Software. Springer-Verlag, Berlin, Heidelberg.

Ducasse, S., Pollet, D., 2009. A process-oriented software architecture reconstruction taxonomy. IEEE Trans. Softw. Eng. 35 (4), 573–591.

Dueñas, J.C., Capilla, R., 2005. The decision view of software architecture. In: Proceedings of the 2nd European Workshop on Software Architecture (EWSA), pp. 222–230.

Dybå, T., Dingsøyr, T., 2008. Empirical studies of agile software development: a systematic review. Inform. Softw. Technol. 50 (9), 833–859.

Dyba, T., Kitchenham, B.A., Jorgensen, M., 2005. Evidence-based software engineering for practitioners. IEEE Softw. 22 (1), 58–65.

Fenton, N., Pfleeger, S., 1996. Software Metrics: A Rigorous and Practical Approach. International Thomson Computer Press, London.

Gallagher, K., Hatch, A., Munro, M., 2008. Software architecture visualization: an evaluation framework and its application. IEEE Trans. Softw. Eng. 34 (2), 260–270.

Harris, D.R., Reubenstein, H.B., Yeh, A.S., 1995. Reverse engineering to the architectural level. In: Proceedings of the 17th International Conference on Software engineering (ICSE), pp. 186–195.

Hofmeister, C., Kruchten, P., Nord, R.L., Obbink, H., Ran, A., America, P., 2007. A general model of software architecture design derived from five industrial approaches. J. Syst. Softw. 80 (1), 106–126.

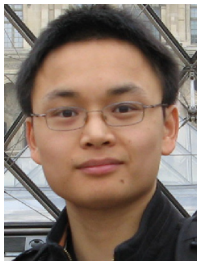
ISO/IEC/IEEE, ISO/IEC/IEEE 42010:2011, 2011. Systems and Software Engineering – Architecture Description.

Jansen, A., Bosch, J., 2005. Software architecture as a set of architectural design decisions. In: Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture (WICSA), pp. 109–120.

Kitchenham, B.A., Charters, S., 2007. Guidelines for Performing Systematic Literature Reviews in Software Engineering, EBSE Technical Report version 2.3, EBSE-2007-01, Software Engineering Group. Keele Univ., Univ of Durham.

Kruchten, P., Lago, P., Vliet, H., 2006. Building up and reasoning about architectural knowledge. In: Proceedings of the 2nd International Conference on Quality of Software Architectures (QoSA), pp. 43–58.
Lee, L., Kruchten, P., 2008. A tool to visualize architectural design decisions. In: Proceedings of the 4th International Conference on Quality of Software Architectures (QoSA), pp. 43–54.
Li, Z., Liang, P., Avgeriou, P., 2013. Application of knowledge-based approaches in software architecture: a systematic mapping study. Inform. Softw. Technol. 55 (5), 777–794.
López, C., Inostroza, P., Cysneiros, L.M., Astudillo, H., 2009. Visualization and comparison of architecture rationale with semantic web technologies. J. Syst. Softw. 82 (8), 1198–1210.
Novais, R.L., Torres, A., Mendes, T.S., Mendonça, M., Zazworka, N., 2013. Software evolution visualization: a systematic mapping study. Inform. Softw. Technol. 55 (11), 1860–1883.
OMG, 2007. Unified Modeling Language (UML) Specification, Version 2.1.2, November.
OMG, 2007. Systems Modeling Language (SysML) Specification, Version 1.0, September.
Poort, E.R., van Vliet, H., 2012. RCDA. Architecting as a risk- and cost management discipline. J. Syst. Softw. 85 (9), 1995–2013.
Shahin, M., Liang, P., Khayyambashi, M.R., 2009. Architectural design decision: Existing models and tools. In: Proceedings of the Joint 8th Working IEEE/IFIP Conference on Software Architecture & 3rd European Conference on Software Architecture (WICSA/ECSA), pp. 293–296.
Shahin, M., Liang, P., Khayyambashi, M.R., 2010. Improving understandability of architecture design through visualization of architectural design decision. In: Proceedings of the 5th ICSE Workshop on SHAring and Reusing architectural Knowledge (SHARK), pp. 88–95.
Sharafi, Z., 2011. A systematic analysis of software architecture visualization techniques. In: Proceedings of the 19th IEEE International Conference on Program Comprehension (ICPC), pp. 254–257.
Sjoberg, D.I.K., Hannay, J.E., Hansen, O., Kampenes, V.B., Karahasanovic, A., Liborg, N., Rekdal, A.C., 2005. A survey of controlled experiments in software engineering. IEEE Trans. Softw. Eng. 31 (9), 733–753.
Sjøberg, D.I.K., Dybå, T., Jørgensen, M., 2007. The future of empirical methods in software engineering research. In: Proceedings of the Future of Software Engineering (FOSE), pp. 358–378.
Spence, R., 2000. Information visualization, 1st ed. Addison Wesley, Boston.
Taylor, R.N., Medvidovic, N., Dashofy, E.M., 2009. Software Architecture: Foundations, Theory and Practice. John Wiley and Sons, Hoboken.
Telea, A., Voinea, L., Sassenburg, H., 2010. Visual tools for software architecture understanding: a stakeholder perspective. IEEE Softw. 27 (6), 46–53.
Wettel, R., Lanza, M., 2007. Visualizing software systems as cities. In: Proceedings of the 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT), pp. 92–99.
Zdun, U., Capilla, R., Tran, H., Zimmermann, O., 2013. Sustainable architectural design decisions. IEEE Softw. 30 (6), 46–53.
Zhang, H., Babar, M.A., Tell, P., 2011. Identifying relevant studies in software engineering. Inform. Softw. Technol. 53 (6), 625–637.

Mojtaba Shahin is an invited researcher in the State Key Lab of Software Engineering (SKLSE), School of Computer, Wuhan University, China, and a lecturer and researcher in the Department of Computer Engineering at Neyriz Branch, Islamic Azad University since September 2010. He got his MS and BS degrees in computer engineering with software engineering major from Sheikh Bahaei University and Shiraz Islamic Azad University in 2010 and 2006, respectively. His current research mainly focuses on software architecture, software architecture visualization, and architectural design decision.



Peng Liang is a professor of software engineering in the State Key Lab of Software Engineering (SKLSE), School of Computer, Wuhan University, China. He is currently a visiting researcher at VU University Amsterdam, the Netherlands. Between 2007 and 2009, he was a post-doctoral researcher at the software engineering and architecture (SEARCH) research group at the University of Groningen, the Netherlands. His research interests concern the area of software architecture and requirements engineering. He has published more than 50 articles in peer-reviewed international journals, conference and workshop proceedings, and books.

Muhammad Ali Babar is a Professor and Chair of Software Engineering in the School of Computer Science, the University of Adelaide, Australia. He also holds an academic position with IT University of Copenhagen, Denmark. Prior to this, he was an Associate Professor (Reader) in Software Engineering at Lancaster University UK. Previously, he worked as a researcher and project leaders in different research centers in Ireland and Australia. His research projects have attracted funding from various agencies in Denmark, UK, Ireland, and Australia. He has authored/co-authored more than 150 peer-reviewed research papers at premier software engineering journals and conferences such as ACM Trans. on Software Engineering and Methods (TOSEM), IEEE Software, and ICSE. He has recently co-edited a book on Agile Architecting published by Morgan Kaufmann, Elsevier. He is a member of the steering committees of several international software engineering and architecture conferences such as WICSA, ECSA, and ICGSE. He regularly runs tutorials and gives talks on topics related to cloud computing, software architecture and empirical approaches at various international conferences. More information on Prof. M. Ali Babar can be found at http://malibabar.wordpress.com.