A Systematic Mapping Study on Text Analysis Techniques in Software Architecture

Article *in* Journal of Systems and Software · July 2018

DOI: 10.1016/j.jss.2018.07.055

CITATION
1

READS
198

4 authors:

Tingting Bi
Wuhan University
7 PUBLICATIONS **13** CITATIONS
SEE PROFILE

Peng Liang
Wuhan University
110 PUBLICATIONS **1,120** CITATIONS
SEE PROFILE

Antony Tang
Swinburne University of Technology
77 PUBLICATIONS **1,595** CITATIONS
SEE PROFILE

Chen Yang
IBO Technology (Shenzhen) Co., Ltd.
9 PUBLICATIONS **58** CITATIONS
SEE PROFILE

The Journal of Systems & Software 144 (2018) 533–558

Contents lists available at ScienceDirect

The Journal of Systems & Software

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

A systematic mapping study on text analysis techniques in software architecture

Tingting Bi[a,b], Peng Liang[a,*], Antony Tang[b], Chen Yang[a,c]

[a] State Key Lab of Software Engineering, School of Computer Science, Wuhan University, 430072 Wuhan, China
[b] Faculty of Science, Engineering and Technology, Swinburne University of Technology, VIC 3122 Melbourne, Australia
[c] IBO Technology (Shenzhen) Co., Ltd, 518057 Shenzhen, China

ARTICLE INFO

Keywords:
Software architecture
Text analysis technique
Systematic mapping study

ABSTRACT

Context: Information from artifacts in each phase of the software development life cycle can potentially be mined to enhance architectural knowledge. Many text analysis techniques have been proposed for mining such artifacts. However, there is no comprehensive understanding of what artifacts these text analysis techniques analyze, what information they are able to extract or how they enhance architecting activities.
Objective: This systematic mapping study aims to study text analysis techniques for mining architecture-related artifacts and how these techniques have been used, and to identify the benefits and limitations of these techniques and tools with respect to enhancing architecting activities.
Method: We conducted a systematic mapping study and defined five research questions. We analyzed the results using descriptive statistics and qualitative analysis methods.
Results: Fifty-five studies were finally selected with the following results: (1) Current text analysis research emphasizes on architectural understanding and recovery. (2) A spectrum of text analysis techniques have been used in textual architecture information analysis. (3) Five categories of benefits and three categories of limitations were identified.
Conclusions: This study shows a steady interest in textual architecture information analysis. The results give clues for future research directions on improving architecture practice through using these text analysis techniques.

1. Introduction

Software Architecture (SA)[1] is a set of structures comprising software elements, the relationships among them, and the properties of the elements and relationships (Bass et al., 2012). SA has become a crucial research and practice in software development since the complexity and size growth of modern systems (Bass et al., 2012). Every software system has an architecture (Kruchten et al., 2006), which plays a vital role in discussions on design, development, evolution, and maintenance throughout the development lifecycle (Bass et al., 2012), for example, an architectural change affects the fundamental ways in which architectural components interact with each other and may trigger changes in many parts of the system (Taylor et al., 2010). SA is also regarded as multiple overlapping design rule spaces, which are formed by one or more structural or evolutionary relationships (Xiao et al., 2014). SA can be characterized by a number of different perspectives and viewpoints which helps to improve development and support software system evolution. These abstract concepts of architecture need to be represented, captured, analyzed, and eventually used.

Textual artifacts such as requirements documents, architecture documents, source code and comments, and bug reports contain a large amount of information related to architecture, which can assist stakeholders to comprehend different aspects of a system (Moreno et al., 2015). Architecture documents provide architecture knowledge and shape software architecture (Jansen et al., 2009). The importance of architectural knowledge management for software development has been claimed for decades (Babar et al., 2009). There are various needs for capturing architectural knowledge, such as architectural knowledge sharing, compliance, and traceability (Capilla et al., 2016). In addition, architecture knowledge provides a main source to enable architectural evaluation, for example, without the captured architecture knowledge; it is difficult for stakeholders to conduct architecture analysis (Tang and

* Corresponding author.
E-mail addresses: bi_tingting@whu.edu.cn (T. Bi), liangp@whu.edu.cn (P. Liang), atang@swin.edu.au (A. Tang), c.yang@ibotech.com.cn (C. Yang).
[1] For readability and clarity, we list all the abbreviations used in this paper in Appendix B for reference.

https://doi.org/10.1016/j.jss.2018.07.055
Received 28 January 2018; Received in revised form 21 May 2018; Accepted 16 July 2018
Available online 19 July 2018

Lau, 2014).

The creation and maintenance of architecture documents in practice is typically often overlooked and architectural knowledge is scattered in informal and semi-structured artifacts, such as meeting notes and e-mails (Babar and Gorton, 2007). Architecture knowledge often exists in multiple documents, which makes finding relevant architecture knowledge difficult (Jansen et al., 2009). Sometimes architectural knowledge is described in a combination of textual and graphical representation (Malavolta et al., 2013). Sometimes architecture knowledge is tacit and not documented at all (Capilla et al., 2016). Especially with the increase in the complexity and size of systems, architecture knowledge management becomes even more challenging, and stakeholders need to find the right knowledge and find them efficiently. Searching and retrieving relevant architectural knowledge from textual artifacts is a challenging task. With a large volume of documents and texts, applying text analysis techniques can greatly facilitate the tasks of finding the desired architectural knowledge. Many works have experimented with text analysis techniques to analyze various software development artifacts for extracting such information.

Text analysis techniques have been widely investigated by researchers. Various methods and tools employing text analysis techniques have been proposed in SA, e.g., architecture information extraction using text mining (Vico and Calegari, 2015). Whilst there are text analysis methods and tools, how these text analysis methods, tools, and techniques can help architecting activities has not been mapped in a holistic way. As such, in order to understand the state of the art of text analysis techniques in SA, we conduct a Systematic Mapping Study (SMS). We aim to identify and analyze the application of *text analysis techniques in software architecture in the context of software architecting activities*.

Systematic Literature Review (SLR) aims at identifying, evaluating, and interpreting all available research works that investigate a topic area or phenomenon of interest with a set of research questions. On the other hand, SMS focuses on providing an overview of a domain, identifying research activities on a research topic. One of the main differences between an SLR and SMS is that SLRs focus on particular research questions, while SMSs have a broader scope to provide an overview of the research landscape (Arksey and O'Malley, 2005). To be specific, software architecture and text analysis techniques cover a broad topic with many different aspects (e.g., architecting activities and software artifacts). To this end, we plan to conduct an SMS instead of an SLR (Kitchenham et al., 2009). This study aims to identify and analyze the use of text analysis techniques in software architecture. We select and analyze research works from major journals, conferences, and workshops.

The structure of the paper is organized as follows: The context of this study (i.e., architecting activities and text analysis techniques) is introduced in Section 2. The execution process of our research methodology is described in Section 3. The results to the research questions are provided in Section 4. The results and their implications are further summarized and discussed in Section 5. The threats to validity are presented in Section 6, and this SMS is concluded in Section 7.

2. Research context

Software architecting activities involve requirements, design, and implementation. Much of the knowledge is captured in a wide variety of documents. Texts used in SA documentation can spread across different stages of software development as well as across different kinds of media such as specification, emails, meeting minutes, posts and even codes. In the following, we discuss architecting activities and how architecting activities can be supported by the knowledge extracted from text analysis techniques. In the field of software architecture, we collected ten architecting activities from related studies in Section 2.1, and we introduce seven categories of common text analysis techniques in Section 2.2.

2.1. Architecting activities

Architects conduct different architecting activities for different purposes. According to Li *et al.*, architecting activities can be classified as specific activities and general activities (Li et al., 2013). The specific activities are practical end-to-end coverage of the entire life cycle of architecture development (Tang et al., 2010), while the general architecting activities can be used to support the specific architecting activities. The explanations of these activities are listed below:

- ***Architectural Analysis (AA)*** aims the issues that an architecture intends to solve. A series of architecturally significant requirements (ASRs) are generated through conducting this activity (Hofmeister et al., 2007).
- ***Architectural Synthesis (AS)*** provides candidate architectural solutions to address the ASRs generated in AA. The AS activity moves from the problem space to the solution space in architecture design (Hofmeister et al., 2007).
- ***Architectural Evaluation (AE)*** is used to assess the architectural solutions provided in AS in terms of the ASRs, and makes sure the architecture decisions made are appropriate ones (Hofmeister et al., 2007).
- ***Architectural Implementation (AI)*** builds the architecture through implementing the architecture decisions in detail design (Tang et al., 2010).
- ***Architectural Maintenance and Evolution (AME)***. Architecture must evolve during its lifetime to respond to changing requirements (Postma et al., 2004). Architectural maintenance is to adjust the architecture according to changes and faults (Hofmeister et al., 2007). It is considered a phase after architecture implementation. In this study, we regard architecture evolution and architecture maintenance as one activity where the architecture is changed either to carry out new requirements or to correct faults.

As mentioned above, the five specific architecting activities run through the whole architecture life cycle, and there are some other activities (we call them general architecting activities) identified in (Li et al., 2013):

- ***Architectural Recovery (AR)*** makes the existing architecture explicit and is used to deal with architectural erosion (Medvidovic and Jakobac, 2006). AR can be conducted through a bottom-up approach from source code, documentation, or physical file organizations (Ducasse and Pollet, 2009).
- ***Architectural Documentation (AD)*** is an activity to specify or document an architecture using a set of architectural elements (e.g., model, concern) (ISO, 2011). The output of AD presents a blueprint for software systems by addressing the components and their dependencies that can facilitate the expression, communication, and evolution of the system.
- ***Architectural Understanding (AU)*** is an activity conducted by architects and related stakeholders to comprehend the architecture design, including the underlying decisions (Li et al., 2013), the structure, behavior, properties, and relationships with other artifacts (Javed and Zdun, 2014).
- ***Architectural Impact Analysis (AIA)*** intends to identify the elements in architecture affected by a change scenario. The identified elements include the components affected directly as well as the components affected indirectly by the change scenario (Bengtsson et al., 2004).
- ***Architectural Reuse (ARu)*** is to reuse the existing architectural designs, decisions, patterns, styles, and so on (IEEE, 2010).

2.2. Text analysis techniques

Text analysis techniques have received increasing attention in

software engineering in recent years because retrieving and analyzing textual information are vital to supporting software engineering tasks, for example, bug triage, program comprehension, traceability links recovery, and software redocumentation (Haiduc et al., 2012), (Haiduc et al., 2016). Text analysis and text mining are two umbrella terms that describe a series of techniques for mining, processing, and analyzing semi-structured and unstructured textual data (Miner et al., 2012). Many software engineering tasks can be addressed through using various text analysis techniques (Haiduc et al., 2013) For example, Maalej et al., introduced several probabilistic techniques that automatically classify user reviews into bug reports, feature requests, user experience, and ratings (Maalej et al., 2016). Their method can help to filter user reviews, and assign them to the appropriate stakeholders, such as developers. Zhang et al., proposed a method EmbTE to extract topics from source code based on word embedding techniques. Their method can identify the most contributory terms from source code (Zhang et al., 2016). In this SMS, we employed a pre-defined classification of text analysis techniques defined in (Miner et al., 2012), including Search and Information Retrieval (SIR), Natural Language Processing (NLP), Clustering (Ct), Web Mining (WM), Classification (Cf), Information Extraction (IE), and Concept Extraction (CE). Each text analysis technique contains several specific topics (Miner et al., 2012), and we provide detailed descriptions of the seven text analysis techniques and their specific topics in Table 1.

In this study, we collect and analyze text analysis techniques and tools that assist architecting activities.

3. Research methodology

This SMS was conducted based on the guidelines in (Brereton et al., 2007). The SMS was conducted in three phases: (1) planning, (2) mapping study execution, and (3) writing the report. We described former two phases in Section 3.1 and Section 3.2, respectively.

3.1. Planning

Before we conducted the SMS, we prepared a research protocol during the planning phase that focuses on the specific objectives of using text analysis techniques in enhancing architecting activities. We specified the following research questions and their rationale in Section 3.1.1. The search scope, strategy, databases, and search terms were defined in Section 3.1.2.

3.1.1. Research questions

We conducted this SMS in order to understand the state of the art of the application of text analysis techniques in SA. We defined five research questions (RQs), which concern what text analysis techniques have been used, where they have been applied, and how effective they are in helping SA. Table 2 articulates our RQs in details.

3.1.2. Study search and selection

In the literature review protocol, we defined the search scope, search strategy, and selection criteria before conducting the SMS. This protocol specifies the process and the conditions for selecting research papers in this SMS (Brereton et al., 2007).

Search scope:

1. Time period: We planned to search as many relevant research papers as possible to get a comprehensive understanding on this topic. Therefore, we did not define the start time in order to reduce risks of omitting some literatures, and the end time was set on December 2017.
2. Electronic databases: This mapping study focuses on the literatures that applying text analysis techniques in software architecture, and Table 3 lists seven most popular databases, these seven databases are considered suitable to retrieve related potential literatures (Chen et al., 2010). Google Scholar was not included in this study since it will produce a number of unrelated results, and the retrieved literatures have overlaps with the seven indexed databases.

Search strategy: Search strategy influences the quality of the retrieved literatures, and determines the time and effort required to search the literatures. The search strategy in this mapping study was divided into two steps.

Step 1: In this mapping study, we defined the search terms based on the topics (i.e., software architecture and text analysis techniques). We defined the search terms by leveraging Population, Intervention,

Table 1
Text analysis techniques and related specific topics (Miner et al., 2012).

Text analysis technique	Description	Topics
Search and Information Retrieval (SIR)	SIR is used for retrieval and storage of information through search engines. The predefined search keywords that matches particular users' requirements (Chowdhury, 2010).	Keyword search Inverted index Question answering
Natural Language Processing (NLP)	NLP employs highly-scalable techniques based on statistics to processing, understanding and search text efficiently. NLP can be divided into low-level tasks (e.g., sentence boundary, tokenization, POS tagging) and high-level tasks (e.g., named entity recognition) (Kao and Poteet, 2005).	Part of speech tagging Tokenization Named entity recognition
Clustering (Ct)	Clustering is an unsupervised learning technique, which is used to divide data elements into groups. Each group of similar elements that are called clusters. The elements of one cluster should be similar to one another and dissimilar to the elements of other clusters (Xu et al., 2003).	Document clustering Document similarity
Web Mining (WM)	WM focuses on discovering and analyzing desired information resources on the Web. There are several important WM techniques, including web content mining, web usage mining (Cooley et al., 1997).	Web crawling Link analytics
Classification (Cf)	Classification is a supervised or semi-supervised learning technique. Cf can be used to classify terms, paragraphs, or documents using classification methods (e.g., Support Vector Machine) based on trained labeled data (Manevitz and Yousef, 2001).	Dimensinality reduction Feature selection
Information Extraction (IE)	IE is used to extract information or structured data (e.g., entity) from unstructured text, and identifying the relationships between the entities (Mooney and Bunescu, 2005), (Cowie and Lehnert, 1996).	Entity extraction Link extraction
Concept Extraction (CE)	CE is a process to group words and phrases into semantically similar groups (Miner et al., 2012).	Word clustering Synonym identification

Table 2
Research questions and their rationale.

Research questions	Rationale
RQ1: What text analysis techniques have been used in mining, recovering, and analyzing textual architecture information?	SA is heavily related to textual representation, for example, 62% practitioners use both textual and visual representations to describe SA (Malavolta et al., 2013) and 89% open source software projects use natural language to document their architectures (Ding et al., 2014). The answer to this RQ can tell us about the existing text analysis techniques that are employed to find/retrieve/analyze textual architecture knowledge, and how they are used.
RQ2: In which architecting activities have text analysis techniques been applied to and what do the techniques do to enhance architecting activities?	Various architecting activities may require specific needs on different text analysis techniques. The answers to this RQ can help us identify which architecting activities use text analysis techniques to generate or consume textual architecture information.
RQ3: What software artifacts related to architecture are analyzed and by which text analysis techniques?	There are many types of software artifacts related to architecture, e.g., architecturally significant requirements, architectural design decisions in architecting activities (Bass et al., 2012). The answers to this RQ can tell us what types of architectural information are recorded in textual artifacts, and what text analysis techniques have been used to extract knowledge from these architecture-related software artifacts.
RQ4: What tools can be used to facilitate the analysis of textual architecture information?	The successful application of text analysis techniques requires tools. The answers to this RQ can tell us what tools can be used for textual architecture information analysis.
RQ5: What are the benefits and limitations of using textual architecture information analysis?	Applying textual architecture information analysis will lead to certain benefits as well as costs. The answers to this RQ can help researchers and practitioners understand the benefits and limitations of the current techniques and tools.

Table 3
Seven electronic databases.

#	Electronic database	Search terms used in
ED1	Science direct	Paper title, keywords, abstract
ED2	Wiley InterScience	Paper title, abstract
ED3	ACM digital library	Paper title, abstract
ED4	ISI web of science	Paper title, keywords, abstract
ED5	EI compendex	Paper title, abstract
ED6	Springer link	Paper title, abstract
ED7	IEEE explore	Paper title, keywords, abstract

Comparison and Outcome (PICO) criteria (Kitchenham and Charters, 2007). The population in this mapping study is software architecture. We excluded the word "software" in the software architecture search terms for the purpose of retrieving as many as studies as possible. We included "software design" in the search terms, since software design may cover software architecture design and "design" itself is a general word, which will generate lots of irrelevant results. The intervention is text analysis techniques. In this mapping study, we made use of the seven text analysis techniques suggested in (Miner et al., 2012), and used search terms that are highly related to the seven text analysis techniques.

Step 2: We conducted a pilot search and selection using various types of combinations of search terms based on our knowledge related to software architecture and text analysis techniques. The pilot search was conducted in the IEEE database (between 2006 and 2016), and two sets of search terms were used. One set (Set A) contains the general words of "text" ("*text*" **OR** "*textual*" **OR** "*document*") **AND** ("*architecture*" **OR** "*architecting*" **OR** "*architectural*" **OR** "*software design*"). The other set (Set B) includes the words of the commonly-used text analysis techniques ("*information retrieval*" **OR** "*information extraction*" **OR** "*natural language processing*" **OR** "*classification*" **OR** "*clustering*" **OR** "*concept extraction*" **OR** "*web mining*" **OR** "*semantic analysis*") **AND** ("*architecture*" **OR** "*architecting*" **OR** "*architectural*" **OR** "*software design*"). The pilot search results of using Set A were 5647 papers, and 8 papers were finally selected. The pilot search results of using Set B were 1956 papers, and 12 papers were finally selected. We found that selected studies using search terms Set B, which can cover the selected studies using search terms Set A. The results show that the search terms related to the commonly-used specific text analysis techniques are more effective for retrieving the relevant studies than the general search terms related to "text". In addition, "*text*", "*textual*", and "*document*" are very general terms in Computer Science. The combinations (e.g., "*text*" and "*architecture*") will lead to a very large proportion of irrelevant results, as well as an enormous number of retrieved papers in the study search phase. To balance the value of the SMS and the effort needed (i.e., reproduce or update this SMS in the future), we decided to exclude the general search terms related to "text" (i.e., "*text*", "*textual*", and "*document*") and used the search terms of the commonly-used specific text analysis techniques (i.e., Set B) in the formal search. Note that, concept extraction techniques aim at understanding the "meaning" of the text, and concept-level text analysis remains the semantics associated between words (Miner et al., 2012). The term "*concept extraction*" is a broad definition that can contain a set of specific text analysis techniques, and it is difficult to cover the applications of "*concept extraction*" only with this term (Miner et al., 2012). As such, we added the specific text analysis technique and most relevant term "*semantic analysis*" in the search terms (Cambria and Hussain, 2015).

The following keywords were kept to build the search strings. The search terms identified in software architecture and text analysis techniques were joined by using Boolean operator **OR**. Software architecture and text analysis techniques were jointed with each other using Boolean operator **AND**. The query designed to search relevant studied is as follows:

("*architecture*" **OR** "*architecting*" **OR** "*architectural*" **OR** "*software design*") **AND** ("*information retrieval*" **OR** "*natural language processing*" **OR** "*classification*" **OR** "*clustering*" **OR** "*information extraction*" **OR** "*concept extraction*" **OR** "*web mining*" **OR** "*semantic analysis*")

Selection criteria: The following inclusion and exclusion criteria were defined for study selection. Note that, we did not exclude short papers (i.e., less than four pages) but we excluded technical reports which are not peer-reviewed.

I1: A study that investigates the application of text analysis techniques in software architecture.
I2: A study that is peer-reviewed and available in full-text.
I3: A study that is written in English.
The exclusion criteria are also considered as follow:
E1: If two studies about the same work were published in different venues (e.g., workshop and conference), we exclude the less mature one.
E2: If a study uses an approach that employs text analysis techniques, but this approach is not used for enhancing architecting activities, we exclude this study.
E3: If a study uses an approach that was applied in architecting activities, but this approach does not employ text analysis techniques, we exclude this study.

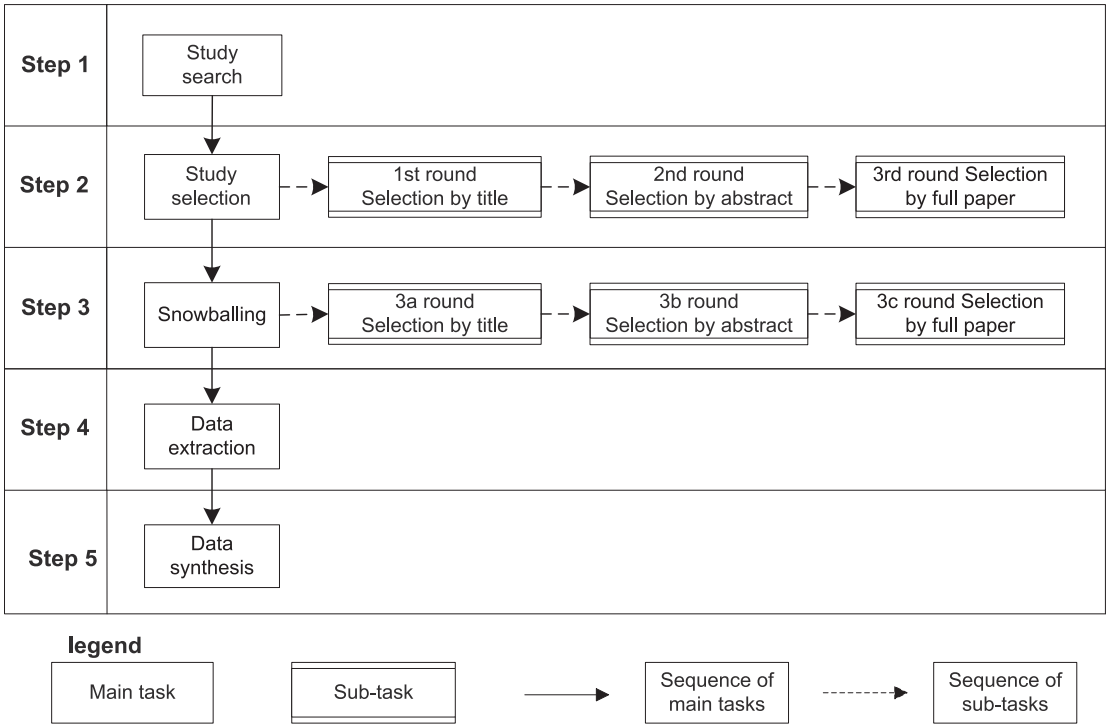

Fig. 1. Execution procedure of this systematic mapping study.

3.2. Mapping study execution

The execution process of this SMS and the allocation of reviewers to particular activities are described in this section, which follows the guidelines proposed in (Kitchenham and Charters, 2007). We executed five main tasks stretching across five steps (see Fig. 1). Study selection contains two parts (i.e., three rounds study selection and snowballing). To make the execution of the mapping study more efficient, we carried out several sub-tasks simultaneously in **Step 2** and **Step 3**.

Step 1: The first author conducted the study search using the search terms (defined in Section 3.1.2) in seven databases (listed in Table 3).
Step 2: The study selection was conducted in this step. The *first round* of study selection (by titles) was performed subsequently by the first author based on the search results. In the *second round* study selection, the first author read the abstracts of studies retained in the first round, and selected the potential studies based on the selection criteria (see Section 3.1.2). In the *third round* study selection, the first and last author selected studies by reading the full paper of the studies left in the second round, and any studies with controversial opinions were discussed and resolved among all the authors.
Step 3: We conducted Snowballing in this step, which contains three rounds (3a, 3b, and 3c) of selection, which are the same as the study selection in **Step 2**.
Step 4: We extracted data according to the data items listed in Table 4 from the selected studies in this step.
Step 5: We synthesized the extracted data in order to answer the RQs defined in Section 3.1.1.

The description of the five steps was detailed in Section 3.2.1 to Section 3.2.3.

3.2.1. Study search

We used the search terms and search criteria defined in Section 3.1.2 to search the related studies in seven electronic databases (see Table 3). The result of the study search is 16245 (see Fig. 2). The seven databases provide different way of retrieval methods. For example, in Science Direct, IEEE Explore, and ISI Web of Science, the search can be set to include titles, keywords, or abstracts of the studies to match the search terms. While, in the rest of the databases (see Table 3), i.e., Wiley InterScience, ACM Digital Library, and EI Compendex, the search terms can be only matched with the title and keywords of the studies. Based on the search result, we conducted the study selection as described in Section 3.2.2.

3.2.2. Study selection

First round study selection: We defined study selection criteria in the planning phase (see Section 3.1.2). Study selection was then conducted based on the selection criteria. In the first round, the first author read the titles of the retrieved studies and selected potential studies, and also kept the studies she could not decide for the second round.

Second round study selection: In the second round of study selection, the first author read the abstracts of the retained studies from the first round, and selected studies based on the selection criteria (see Section 3.1.2). The studies she could not judge by reading the abstracts were kept for the third round.

Third round study selection: The first and last author read the full papers that retained from the second round independently, and decided whether a study should be finally selected or not. For a controversial study, all the authors discussed together and reached a consensus about whether the study should be finally included or not.

Snowballing: We followed a snowballing technique (Wohlin, 2014) to collect and scan the references of the selected studies in order to find more potential primary studies Snowballing is an iterative process (note that we only conducted backward snowballing). The first author read the references of the studies that are retained in the third round of study selection in the first iteration; then checked the referenced papers of newly selected studies of the last iteration in the subsequent iterations. When there is no newly selected paper, the iterative process is complete. Each iteration has a three round selection process as shown in Fig. 1. Snowballing also goes through selection that is based on filtering research papers by their titles, abstracts, and full contents. We included the final selection of papers from snowballing in the final selected studies.

Table 4
Data items extracted from each selected study.

#	Data item	Description	Relevant RQ
D1	Index	The ID of the study.	Overview
D2	Title	The title of the study.	Overview
D3	Year	The published year of the study.	Overview
D4	Source	The published venue of the paper.	Overview
D5	Author list	The full names of all the authors.	Overview
D6	Authors' type	Researcher or practitioner or both.	Overview
D7	Summary	The main idea of the study.	Overview
D8	Text analysis technique (s)	What specific text analysis technique (s) is conducted in the study to address the problems in architecting activity/ activities?	RQ1
D9	Architecting activity/activities	What architecting activities are mentioned in the paper?	RQ2
D10	Artifact (s)	What type of architecture-related software artifacts in the study are processed as inputs by text analysis techniques?	RQ3
D11	Tool support	What tool is used in the study to support the application of text analysis techniques in architecting activity/activities?	RQ4
D12	Benefits	What are the benefits of applying text analysis techniques in textual architecture information analysis?	RQ5
D13	Limitations	What are the costs of applying text analysis techniques in textual architecture information analysis?	RQ5

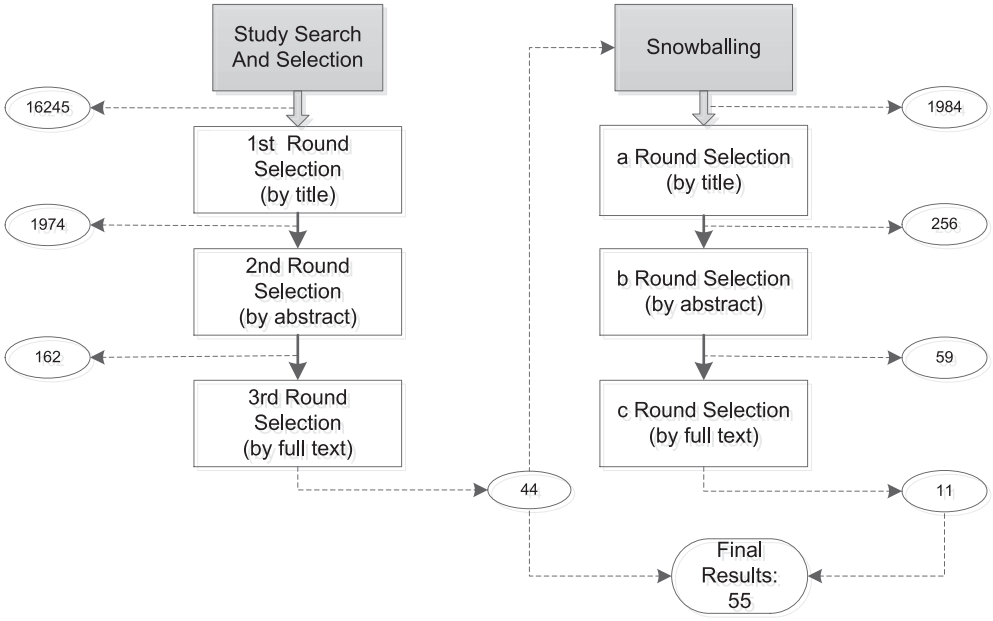

Fig. 2. Results of study search and selection.

3.2.3. Data extraction

The relevant data items are defined (see Table 4) for answering the RQs in Section 3.1.1. Since the extracted data is critical for data synthesis, the first author conducted a pilot data extraction from ten studies, and the rest of the authors reviewed and discussed the pilot data extraction results and reached a consensus on the extracted data when disagreement exists. In the formal data extraction, the first author conducted data extraction from the selected studies, and the second author checked the data extraction results. Any ambiguous extraction data was discussed by all the authors, and finally an agreement on the data extraction results was reached. D1-D7 provide the general information of the selected studies, including the titles of the studies, names of authors, publishing year, etc. D9-D13 contribute to the RQs. Finally, the extracted data were recorded in an Excel file for data synthesis.

3.2.4. Data synthesis

We extracted thirteen types of data items from the selected studies (See Table 4) for our analysis. We then applied descriptive statistics and qualitative analysis to analyze these thirteen types of data.

Descriptive statistics was used to analyze text analysis techniques (i.e., D8 in Table 4), architecting activities (i.e., D9), artifacts (i.e., D10), and tools (i.e., D11). The data can be used to answer RQ1, RQ2, RQ3, and RQ4, respectively. For example, we extracted the employed text analysis techniques from each selected study, and classified these techniques into the seven categories (see Section 2.2) to answer RQ1.

In order to answer RQ2, we used qualitative analysis to analyze the categories of architecting activities (i.e., D9). Qualitative analysis was also used to analyze the benefits and limitations (i.e., D12 and D13) of applying text analysis techniques in mining, recovering, and analyzing textual architecture information. The results of data synthesis can be used to answer RQ5. The process of qualitative analysis consists of three steps: (1) Initial coding was performed to classify the topics and themes of analyzed data (e.g., benefits and limitation of textual architecture information analysis). (2) Line-by-line selective coding is then carried out to identify the subtypes of topics and themes. (3) The disagreements of the coding results were discussed and resolved among all the authors.

4. Results

The results of study search and selection with the statistical information are presented in Section 4.1. In Section 4.2–4.6, we analyzed the results in terms of the five RQs defined in Section 3.1.1 respectively.

4.1. Overview

The results of the search, selection, and snowballing phases were reported in this section. After three selection rounds, fifty-five studies (listed in Appendix A) were found.

4.1.1. Search and selection results

A summary of the papers found in each round is shown in Fig. 2. In the study search, we collected 16245 studies based on the search terms. In the study selection, 1974 studies were retained after the 1st round, 162 studies were kept after the 2nd round, and 44 studies were finally selected after the 3rd round. Then, we conducted three iterations of snowballing (note that we only conducted backward snowballing). In each round of iteration in the snowballing, we conducted three rounds of study selection (i.e., by title, by abstract, and by full paper). To be specific, we initially collected 1984 references, retained 256 studies after the "a" round, kept 59 studies after the "b" round, and ultimately selected 11 studies. In total, 55 studies are finally selected (see the full information of the 55 selected studies in Appendix A).

4.1.2. Studies distribution

In this section, we illustrate the statistical information of the selected studies (including study classification by publication venue, publication year, and author type). Fig. 3 presents the results in a map by analyzing the data of selected studies that presents the studies distributed over the dimensions of architecting activities, text analysis techniques, and time period. In the top of Fig. 3, we illustrated ten architecting activities and seven categories of text analysis techniques. The relationships of the three elements are shown in the lower half part of the map. The left part reflects the relationship between publication year and architecting activities of the selected studies, and the number in bubbles represents the corresponding selected studies (listed in Appendix A) in specific architecting activities. The right part shows the relationship between architecting activities and text analysis techniques. The number in the bubble signifies the number of studies that employing a certain text analysis technique in specific architecting activity.

In Table 5, we illustrated the publication venues of the selected studies (e.g., the information of publication name, type, and count). The 55 selected studies are distributed over the 39 different publication venues. The leading venues for the topic (i.e., applying text analysis techniques in SA) are ICSE, CSMR, WICSA (4 studies for each venue), SCP, TSE, SHARK (3 studies for each venue), IST and ESE (2 studies for each venue). Journal and conference are the two main publication types, which account for 45.5 % (29 out of 55 studies) and 29.1% (16 out of 55 studies), respectively. Workshop papers constitute 18.2% (i.e., 10 out of 55 studies) of the total selected papers.

Fig. 4 illustrates the number of selected papers fluctuating over the last 16 years (i.e., 2002-2017). One study was published in 2002 and there were no studies published in 2003 and 2004. From 2006 onwards, the number of studies applying text analysis techniques in enhancing SA increased gradually, and reached its peak at 2016. Research works in this area continue when we cut off our literature search.

We also collected the information about whether the authors come from academia or the industry: authors who are academics made up the largest proportion (85.4%, i.e., 47 out of 55 studies), authors who are

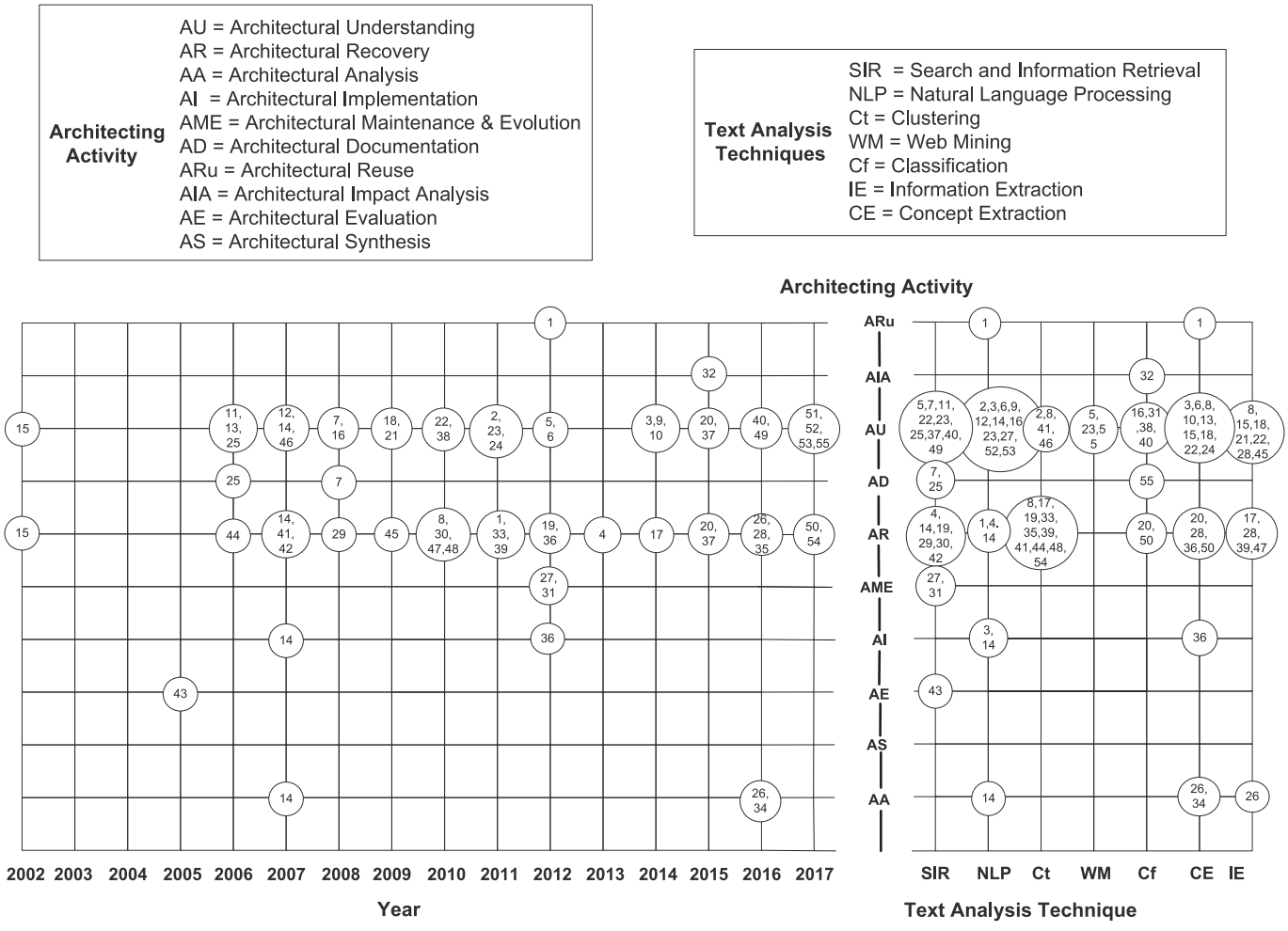

Fig. 3. Bubble chart over text analysis techniques, architecting activities, and time period.

Table 5
Number and proportion of the selected studies over publication venues.

Publication venue	Venue type	No.	%
International Conference on Software Engineering (ICSE)	Conference	4	7.2
European Conference on Software Maintenance and Reengineering (CSMR)	Conference	4	7.2
Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture (WICSA/ECSA)	Conference	4	7.2
Workshop on SHAring and Reusing Architectural Knowledge (SHARK)	Workshop	3	5.4
IEEE Transactions on Software Engineering (TSE)	Journal	3	5.4
Information and Software Technology (IST)	Journal	2	3.6
Science of Computer Programming (SCP)	Journal	2	3.6
Empirical Software Engineering (ESE)	Journal	2	3.6
Journal of Systems and Software (JSS)	Journal	1	1.8
Knowledge-based Systems	Journal	1	1.8
International Journal Multimedia and Image Processing (IJMIP)	Journal	1	1.8
Software Practice and Experience (SPE)	Journal	1	1.8
Journal of Software Engineering Research and Development (JSERD)	Journal	1	1.8
CLEI Electronic Journal	Journal	1	1.8
Software Practice and Experience (SPE)	Journal	1	1.8
IEEE International Conference on Software Architecture (ICSA)	Conference	1	1.8
Argentine Symposium on Software Engineering (ASSE)	Conference	1	1.8
Australian Conference on Software Engineering (ASWEC)	Conference	1	1.8
European Semantic Web Conference (ESWC)	Conference	1	1.8
Working Conference on Reverse Engineering (WCRE)	Conference	1	1.8
International Conference on Program Comprehension (ICPC)	Conference	1	1.8
International Conference on Software Maintenance and Evolution (ICSME)	Conference	1	1.8
International Conference on Information Society (i-Society)	Conference	1	1.8
International Conference on Business Information Systems (BIS)	Conference	1	1.8
International Conference on Fuzzy System and Knowledge Discovery (FSKD)	Conference	1	1.8
International Conference on Software Engineering and Formal Method (SEFM)	Conference	1	1.8
International Multitopic Conference (INMIC)	Conference	1	1.8
International Conference on Intelligent Computing and Cognitive Informatics (ICICCI)	Conference	1	1.8
International Conference on Advances in New Technologies, Interactive Interfaces and Communicability (ADNTIIC)	Conference	1	1.8
International Workshop on Semantic Matchmaking and Resource Retrieval (SMRR)	Workshop	1	1.8
International Workshop on the Twin Peaks of Requirements and Architecture (TwinPeaks)	Workshop	1	1.8
Annual IEEE International Conference and Workshop on Engineering of Computer Based System (ECBS)	Workshop	1	1.8
International Workshop on Eternal Systems (EternalS)	Workshop	1	1.8
International Workshop on Empirical Software Engineering in Practice (IWESEP)	Workshop	1	1.8
International Conference on Software Architecture Workshops (ICSAW)	Workshop	1	1.8
Asia-Pacific Software Engineering Conference Workshops (APSECW)	Workshop	1	1.8
International Symposium on Web Site Evolution (WSE)	Workshop	1	1.8
ACM Symposium on Applied Computing (SAC)	Workshop	1	1.8

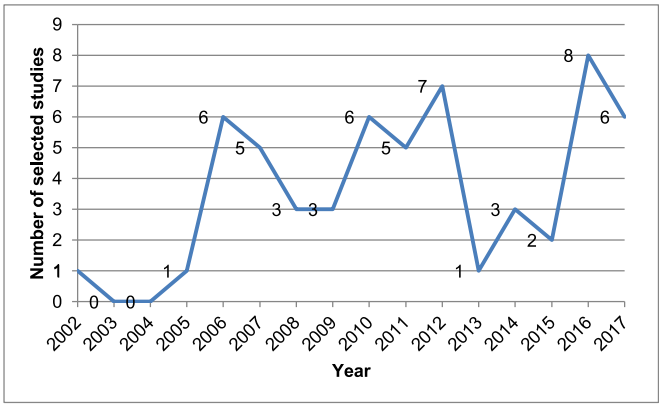

Fig. 4. Number of selected studies over time period.

industry practitioners accounted for 1.8% (i.e., 1 out of 55 studies) of all studies, 12.7% of authors (i.e., 7 out of 55 studies) are from both the academia and the industry.

4.2. RQ1: Text analysis techniques for mining recovering, and analyzing textual architecture information

We classified text analysis techniques into seven categories (i.e., search and information retrieval, clustering, classification, web mining, information extraction, natural language processing, and concept extraction) which are proposed in (Miner et al., 2012). Fig. 5 shows the distribution of text analysis techniques classified by mining, recovering, and analyzing textual architecture information. The seven categories of text analysis techniques have considerable overlaps, since many text analysis tasks sit at the intersection of multiple practice areas (Miner et al., 2012). For example, classification and clustering techniques are two classical techniques used in pattern recognition, which are both based on data mining and statistics. In this SMS, one selected study may make use of two or more text analysis techniques, and consequently the number of overall text analysis techniques (80) is greater than the number of the selected studies (55). In most of the selected studies, text analysis technique can be subdivided into specific techniques. For example, the specific text analysis technique used in [S22] is topic modeling, which belongs to Concept Extraction category. The specific technique employed in [S30] is link analysis that also can be classified into Concept Extraction category. We extracted specific text analysis techniques from the selected studies (see Table 6) to provide details of text analysis techniques used by the selected research works.

The seven categories of text analysis techniques (SIR, NLP, IE, CE, Ct, Cf, and WM) used in the selected studies are shown according to their popularity in Fig. 5. The applied text analysis techniques and their specific techniques are detailed below:

Search and Information Retrieval (SIR) is the most frequently used text analysis technique in processing textual architecture information (i.e., 17 out of 80 techniques, 21.2%). We found that two types of specific techniques can cover the most SIR topics of the selected studies: For example, in [S19], the authors analyzed software entities and calculated the dissimilarity between them using LSA-based measure, then applied K-means algorithm to group similar software entities. This method is effective for architectural view recovery. In [S42], the

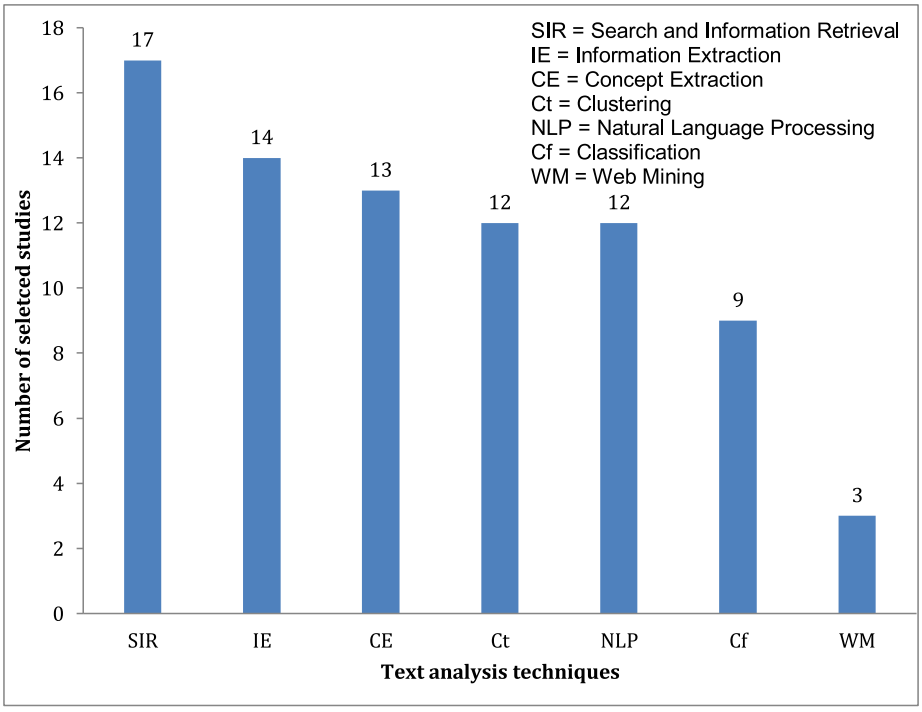

Fig. 5. Number of selected studies over text analysis techniques.

authors applied LSA to group similar vocabularies of source code and software documents in order to find the intention of the code. To be specific, LSA is used to calculate the similarities between entities (e.g., packages and classes) of source code documents, and cluster the entities based on the similarities. As such, a system can be divided into linguistic topics that represent groups of the documents. The results of the case studies show that the clustered linguistic topics can be related to application concepts or architectural components. While, keyword search facilities indexing, searching, and retrieving documents from large text databases based on the keyword queries. In [S43], the authors proposed a knowledge management tool (i.e., PAKME) to process architecture knowledge. The tool contains four components: user interface, knowledge management, search, and reporting. One of the functions in the search component is keyword-based search, which can explore desired artifacts in the repository using the keywords attached to each artifact. This tool can be used for searching and managing architecture knowledge.

Information Extraction (IE) techniques rank second in terms of popularity (14 out of the 80 techniques, 17.5%) amongst text analysis techniques. IE is used to automatically extract structured information from unstructured data source. The outputs are entities, relationships between entities, and attributes describing entities (Mooney and Bunescu, 2005). We classified the IE techniques used in the selected studies into two types: entity extraction and relationship extraction. For example, in [S34], the authors proposed an approach based on an IE technique (i.e., entities extraction) and knowledge representation (i.e., ontology) to automatically analyze architecture patterns considering specific quality attributes (e.g., performance). To be specific, an ontology that contains two sub-ontologies was predefined. One is English grammar-based ontology, which is further categorized into promotes

Table 6
The specific text analysis techniques employed in the selected studies.

Category	Specific text analysis technique	Studies
SIR	Keyword search	[S4][S5][S7][S11][S12][S14][S23][S43] [S49][S51]
	LSA	[S7][S19][S25][S29][S30][S37][S42]
IE	Entity extraction	[S1][S3][S6][S10][S14][S18][S24][S26][S34][S36]
	Entity linking	[S13][S15][S27]
CE	Link analysis	[S6][S8][S21][S22][S31][S37][S47]
	Word clustering	[S17][S22][S28][S46][S50]
Ct	K-means	[S2][S17][S19][S30][S45]
	HAC	[S8][S39]
	WCA, LIMBO	[S41][S48]
	K-Medoid	[S35]
	MST	[S8]
	Modularization quality clustering, Edge clustering, Design structure matrix clustering	[S45]
NLP	POS tagging	[S2][S6][S15][S16]
	Tokenizer	[S1][S8][S9]
	Stemming	[S9]
	Parsing	[S12]
Cf	SVM	[S16][S38][S40]
	NB	[S20][S40]
	DT, BLR, AdaBoost, Bagging Ensemble rule learning	[S38][S40]
WM	Semantic web search	[S5][S23][S55]

verb, modal verb etc. The other is performance ontology that defines performance-specific concepts (e.g., security and throughput). IE technique (i.e., entity extraction) and the ontology were used to identify the relationships between architectural patterns and quality attributes in architectural pattern descriptions. For example, a sentence of a certain architecture pattern description "… *such a configuration can further increase system performance and throughput*" can be transformed into "… *such a configuration no_doubt_modal_verb_of_possibility* (*can*) *conjunction* (*further*) *promotes_verb* (*increase*) *system promotes_metric* (*performance*) *conjunction* (and) *promotes_metric* (*throughput*)". The results of experiment show that the approach is helpful for inexperienced architects to select architecture patterns through knowing whether specific quality attributes are promoted or inhibited. As mentioned above, IE can be used to identify a set of relationships or links among various entities. For example, in [S15], the authors introduced a source model extraction tool that based on IE technique to recover architectural concepts embedded in the source code. This work can help engineers quickly understand particular architectural aspects that are expressed in source code patterns.

Concept Extraction (CE) techniques rank third (13 out of 80 techniques, 16.2%). We classified CE techniques of the selected studies into two types: link analysis and word clustering. As examples of link analysis, in [S21], the authors presented an approach that aims at identifying modules for the purposes of architecture recovery and understanding of object oriented software system. The authors firstly employed a widely used link analysis algorithm to identify the layers of the system, and analyzed the static structure of a software system. Then lexical information was extracted from the source code to identify similarity among pairs of classes and partitions each layer into software modules. In [S22], the authors employed topic modeling to construct traceability links among the artifacts that were produced during the software development process. This approach can aid developers in analyzing the semantic nature of artifacts as well as the entire architecture. Word clustering based on the measurement of the semantic similarity between words is an important task in CE. Word clustering analyzes and groups the words of the textual dataset in order to summarize the topics of groups. In [S28], the authors proposed a word clustering technique (Latent Dirichlet Allocation, LDA) that combines lexical and structural information of a given system to identify meaningful topics in the entities of the system. These topics were used to group similar entities into clusters corresponding to responsibilities of the system. The approach is effective to recover layered architecture of the system.

Clustering (Ct) techniques and *Natural Language Processing (NLP)* techniques rank forth (both 12 out of 80 techniques, 15.2%). Ct is an unsupervised technique that uses data mining algorithms to group similar documents into clusters. There are two types of Ct techniques employed in the selected studies: partitional clustering ([S2][S8][S17][S19][S30]) and hierarchical clustering ([S17][S33][S39][S41]). The output of partitional clustering techniques is an initial partition with a certain number of clusters. K-means is the most frequently used algorithm of partitions clustering (Miner et al., 2012). For example, in [S19], the authors analyzed entities of software and calculate the dissimilarity between the entities using LSA, and then K-means algorithm was used to divide software entities into groups that implement similar functionality. In [S2], the authors applied NLP techniques and K-means algorithm to semantically categorize candidate responsibilities into groups. This approach firstly processes requirements documents by POS tagging technique to detect the actions and tasks that the system needs. Afterwards, K-means is used to group similar responsibilities into the architectural components. The experiments show that the results obtained by this approach correspond to the expected architectural components made by experts. Hierarchical clustering is an iterative process which iteratively merges smaller clusters into larger ones. The clusters that generated by hierarchical clustering are hierarchies that is more informative than an unstructured set of clusters. Most of the selected studies used hierarchical clustering to produce a hierarchical decomposition of entities (e.g., software modularization, entities are files or functions), and hierarchical clustering can support the bottom-up architecture recovery process. For example, in [S41], the researchers conducted the experiments with four Open Source Software (OSS) systems to compare six hierarchical clustering techniques about their performance in recovering the architecture and modules.

Natural Language Processing (NLP) provides a means to understand more about human language overlaps into computer science (Yi et al., 2003). There are several specific NLP techniques (e.g., Part of Speech, Tokenizer), which have been used in pre-processing architectural information in the selected studies. For example, in [S2], NLP techniques are used to separate the elements in a requirement sentence. The process of Part of Speech (POS) tagging is to assign a lexical class marker to every word in a sentence. In this work, POS tagging techniques and a set of domain rules are used in requirements documents to judge whether the verb phrases can be selected as a candidate responsibility. The verb phrases within requirement sentences most likely refer to tasks to be performed by the system and can be translated to responsibilities associated to some components in the software model. The results show that the elements detected by POS tagging technique become responsibilities of certain components in the final architecture. In [S6], the authors proposed an approach for implementing a stakeholder-centric and agile architecture documentation process. Their approach is mainly used to address the dynamic aspect of the stakeholder profiles, which are built on top of existing stakeholders' characterizations. The authors used a named entity recognizer to detect named entities (such as persons, institutions, and artifacts), and then employed POS tagger to automatically assign a grammatical label to every word in a sentence. By using these two NLP techniques (i.e., named entity extraction and POS tagging), each user's message can be associated to a set of entities, which is helpful for a stakeholder-centric architecture documentation process.

Classification (Cf) techniques rank sixth (8 out 80 techniques, 10.0%) of text analysis techniques. Classification is supervised or semi-supervised techniques of assigning text or documents into two or more categories. The most commonly used classification algorithms in the selected studies are Support Vector Machine (SVM) and Naïve Bayes. In [S16], the authors used POS tagging technique to fragment Non-Functional Requirements (NFRs) from plain text under the supervision of experts, then applied SVM to classify the identified NFRs into two categories "architecture concerns" and "quality attributes". In [S20], the authors proposed an approach to detect architecturally relevant classes and used these classes to represent architectural components. In their work, several tools implement the approach based on Naïve Bayes classification algorithm. The results of experiments show that the approach is effective for detecting key classes that conceptualize architectural components. In [S40], the authors evaluated and compared the efficacy of the six classification algorithms (i.e., SVM, C45, Bagging, SLIPPER, Bayesian logistic regression, and AdaBoost) for identifying architectural tactics from source code.

Web Mining (WM) is the least explored text analysis techniques. Three studies employed semantic search mechanism in the web context to search architectural information. For example, in [S23], a framework was presented that can be used to search architecture knowledge in the textual artifacts produced in Virtual Communities for Software Development (VCSD). Their work provides a lightweight mechanism that can be easily adopted by the virtual community and is especially beneficial for new developers who need to understand the decisions taken during system construction.

4.3. RQ2: Applying text analysis techniques in architecting activities

Our second research question is to investigate what software architecting activities the text analysis techniques have been applied to and how the various text analysis techniques are used to enhance

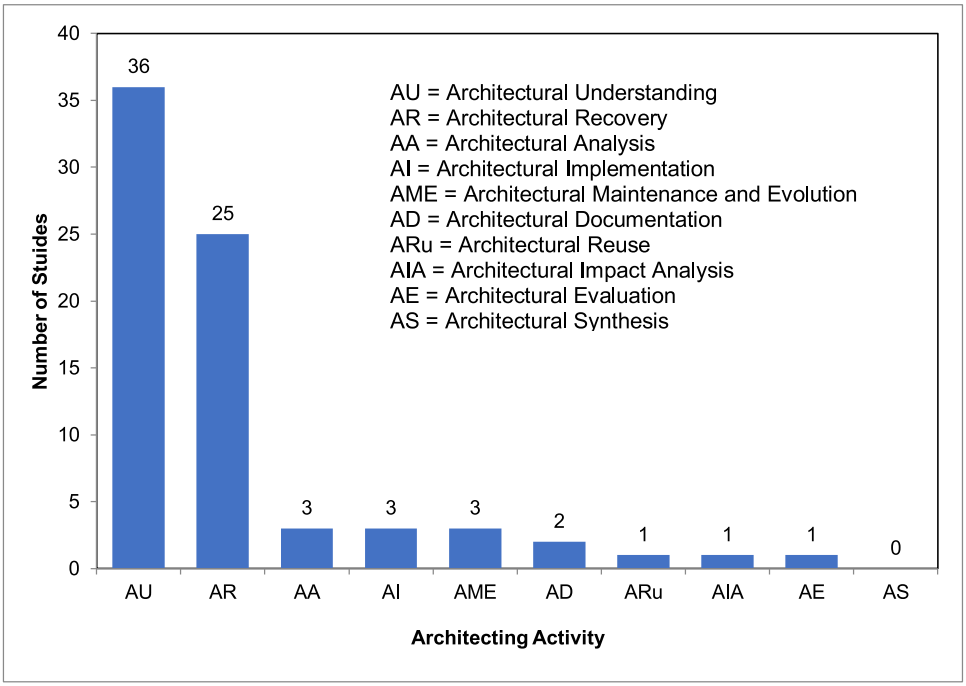

Fig. 6. Number of selected studies over architecting activities.

software architecting activities. As mentioned in Section 2.1, we followed a pre-defined set of architecting activities in (Li et al., 2013) when answering this RQ. We illustrated the numbers of selected studies over ten architecting activities in Fig. 6. The difference is notable: text analysis techniques are used in Architectural Understanding (AU) with thirty-six studies, while there is no studies applied text analysis techniques in Architectural Synthesis (AS). The architecting activities are presented below according to popularity:

Architectural Understanding (AU). Researchers and practitioners focused their efforts on AU when applying text analysis techniques in SA. Software architecture should be well understood before any corrections or changes can be applied to it (Knodel et al., 2008). AU is a prerequisite for and supportive to other architecting activities (e.g., AA, AI, AME, and AE). Architectural components are vital in achieving the desired software quality, and comprehension of architectural components and their relationships is important in supporting AU (Stevanetic and Zdun, 2014). Understanding architecture is also a key factor for maintainability of software system (Shahin et al., 2014). For example, AU enables maintainers to gain a comprehensive overview of the complex dependence relationships between components and/or connectors in the software (Zhao, 2001). One common practice of text analysis techniques is to process textual artifacts and mine key information to improve understanding of the content of the textual artifacts, especially the large collections of texts (Feldman and Sanger, 2007). In this SMS, more than half of the selected studies (i.e., 36 out of 55 studies, 65.5%) applied text analysis techniques to facilitate AU. Four purposes of AU were identified: understanding of architectural knowledge (including rationale), understanding of architecture elements (e.g., architecture patterns, tactics, views, and models), understanding of relationships between architecture elements (e.g., the relationship between architecturally significant requirements and components), and understanding of the whole system; The purposes of AU and the employed specific text analysis techniques are shown in Table 7. For example, in [S2], the authors employed and compared four partitional and hierarchical clustering techniques to analyze requirements documents for detecting architectural responsibilities and conceptual components that can aid architects understanding the system. In [S9], the authors presented an approach based on NLP techniques (i.e., text parsing) to discover stakeholders' interests in architecture documents. Their work supports stakeholders in capturing relevant architectural knowledge that is helpful to comprehend their specific parts and the tasks of the system. Note that, [S38] compared more than forty-seven classification algorithms for rationale extraction, and we only listed the top four effective algorithms in Table 7.

Table 7
Purposes of architecting understanding and the employed specific techniques.

Purposes of AU	Text analysis techniques	Studies
Understanding of architectural knowledge	Keyword search	[S5][S12] [S23][S51] [S52]
	Entity extraction	[S3][S10] [S49]
	POS tagging	[S6]
	Tokenizer, stemming	[S9]
	LSA	[S7]
	SVM, NB, AdaBoost, Bagging	[S38]
	Word clustering	[S46]
Understanding of architecture elements (e.g., architecture patterns, tactics, views, and models)	POS tagging	[S8][S16]
	Keyword search	[S11][S23]
	Tokenizer, Stemming	[S8]
	Web mining	[S55]
Understanding of relationships between architecture elements	Link analysis	[S31][S37]
	Entity extraction	[S14]
	LDA	[S22]
	POS tagging	[S2]
	SVM, NB, DT, BLR, AdaBoost, Ensemble rule learning, Bagging	[S40]
Understanding of the whole system	Link analysis	[S8][S37] [S47]
	POS tagging	[S2][S15]
	LSA	[S42]

Architectural Recovery (AR) is conducted to cope with architectural erosion (Medvidovic and Jakobac, 2006), through reverse engineering of existing implementations (Lung, 1998), (Chardigny and Seriai, 2010). We list the purposes of AR in Table 8. From the selected studies, four AR purposes are identified, i.e., recovery of architectural

Table 8
Purposes of architecting recovery and the employed specific techniques.

Purposes of AR	Text analysis techniques	Studies
Recovery of architectural components, layered architecture (package level)	Link analysis	[S8][S21] [S47]
	Word clustering	[S17][S28]
	Keyword search	[S4][S14]
	WCA, LIMBO	[S48][S54]
	NB	[S20]
	LSA	[S42]
	SVM	[S40]
	Entity extraction	[S36]
	HAC	[S39]
Recovery of architectural view	LSA	[S19]
	WCA, LIMBO	[S41]
	Bunch-based clustering	[S44]
	K-means	[S45]
Recovery of architectural documentation	LSA	[S30]
Recovery of architectural rationale, tactics, and concept	Entity extraction	[S1]
	LSA	[S29]
	LDA	[S50]

components, recovery of architectural view, recovery of architectural documentation, and recovery of architectural rationale and concept. As mentioned in Section 2.1, recovered architecture and architectural information (e.g., rationale) are usually obtained from source code, documentation etc. In this SMS, we found that most of the selected studies recovered architecture from textual information of the source code (e.g., [S15][S17][S28][S19][S33][S39][S50][S54]). In terms of AR approaches, clustering and pattern detection are two complementary approaches. Clustering is a technique of automatically constructing categories or taxonomies for a set of objects. Clustering aims at grouping all entities (e.g., source files or classes) into clusters (Quilici, 1995) to support AR. For example, in [S28], the authors proposed a word clustering technique, Latent Dirichlet Allocation (LDA), which groups structural and lexical information of the systems to recover its layered architecture. The results show that this approach can extract the responsibilities of a system, and group the responsibilities into manageable and conceptually cohesive clusters. The effectiveness of this approach has been corroborated by the manual analysis of the source code and the documentation of the system. In [S33], the authors conducted a case study that uses several clustering techniques to group classes of the source code of an object-oriented system. The results show that the quality of decompositions of this approach is relatively effective with the manual architecture recovery approaches. While, pattern detection concerns common abstractions hidden and embedded in the system. However, in practice not all the entities of a system can be covered by pattern detection approach (Quilici, 1995). In [S1], the authors presented a pattern-based information extraction approach to recover, represent, and explore information about architecture rationale from architecture documents. The result shows that this approach is helpful to architecture rationale reuse and architects training.

Architectural Analysis (AA), Architectural Maintenance and Evolution (AME), *and* ***Architectural Implementation (AI)*** each has three studies. In [S14], the authors mentioned that the first step of AA is to identify tasks that the architecture must solve. The authors used NLP techniques (e.g., named entity extraction and tokenization) in system documentation to analyze the candidate architectural components, architecturally significant requirements, and potential architectural styles of the system. A large number of instances of architectural layers can be discovered. For example, this approach can discover that the application layer contains an array of classes, and this information provides clues to indicate that the system used a typical layered architecture and provides important references for further analysis of software documents. This approach can assist maintainers in analyzing the architecture of a software system (i.e., architectural analysis). AME ensures consistency and integrity of architecture during the development (Postma et al., 2004). In [S27], an ontological approach was proposed that provides consistent and flexible expressions for a set of artifacts, including documentation and source code. In this work, the authors used ontology and NLP technique (i.e., POS tagging) to analyze architecture documents for identifying potential components of the system and their containing source code entities. In addition, this approach can identify the relationships between packages or components. The identified information (e.g., components, layers) can provide guidelines for architectural maintenance activity through understanding the relationships between packages or components, such as data and control communications between components. This approach is also applicable to other architecting activities, such as architectural recovery.

Architectural Documentation (AD) is also not well-explored research area. As an example of AD, in [S7], the authors used LSA for documents identification with diminishing levels of abstraction: from the highest level architectural descriptions through service definitions, and to use case specification at the lowest level.

Architectural Impact Analysis (AIA), Architectural Reuse (ARu), and **Architecture Evaluation (AE)** each has one study. In [S32], the changed Architecturally Significant Functional Requirements (ASFRs) can be extracted by an automatic classification method (e.g., Naïve Bayes). Then the extracted ASFRs are classified into the categories according to the different types of architectural impact they have. In [S1], the authors proposed an IE approach (i.e., entity extraction) and a tool to identify and extract architectural rationale information. This approach enables rationale and architecture reuse, rationale auditing, and architects training. AE is used to guarantee that the selected architectural solution is a suitable one. In [S43], the authors developed an architecture knowledge management tool (PAKME). PAKME has a search component which helps the users to search through the knowledge repository based on keywords. The reporting component of PAKME supports AE by helping stakeholders to generate utility trees and present the findings of AE as a result tree, which categorizes the findings into different risk themes.

There is no study that employs text analysis techniques in enhancing Architectural Synthesis (AS). From the results, we can see that there is a considerable difference on the number of architecting activities supported by text analysis techniques. Researchers and practitioners had spent most effort on AU and AR, and the rest of architecting activities have not been explored well. We will discuss the potential reasons in Section 5.

4.4. RQ3: Applying text analysis techniques to architecture-related software artifacts

To answer RQ3, we collected sixty-one software artifacts related to architecture from the selected studies and analyzed them by their category. These architecture-related software artifacts can be classified into four categories: Architecture-related Document (ArD), Source Code and Comment (SCC), Requirements Document (RD), and Community Artifact (CA). The distribution of the four categories of architecture-related artifacts is shown in Fig. 7. In addition, to get a detailed view of the relationships between analyzed architecture-related artifacts and employed text analysis techniques, we present in Table 9 the numbers of each category of architecture-related artifacts analyzed by the seven text analysis techniques.

Architecture-related Document (ArD) is the most frequently (i.e., 24 out of 61 studies, 39.3%) analyzed architecture-related artifact by text analysis techniques. Architecture documents are a set of textual artifacts in software development that capture the architectural elements and key architectural decisions (Clements et al., 2010). In this work, architecture documents and the documents related to architecture (e.g., descriptions of different architecture patterns [S34]) are

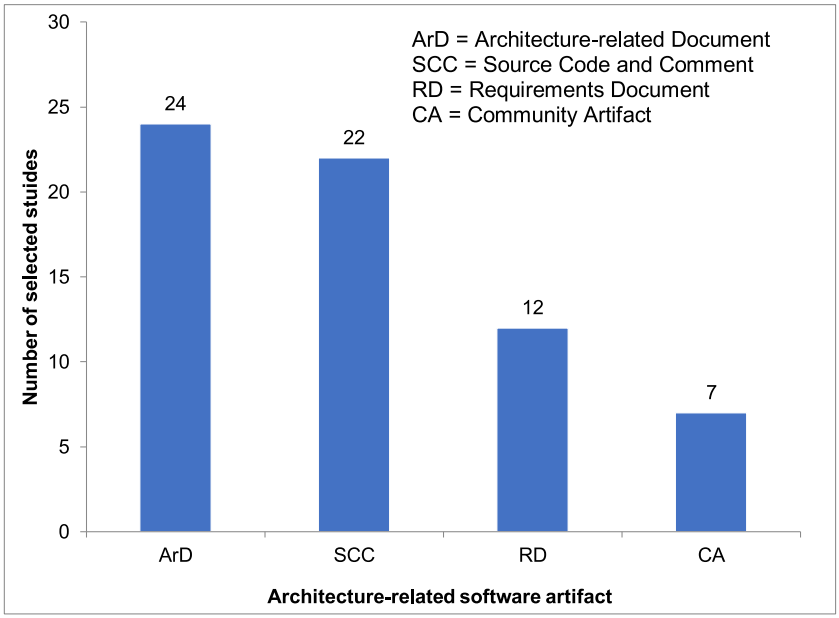

Fig. 7. Number of studies over architecture-related software artifacts.

Table 9
Number of architecture-related software artifacts analyzed by each text analysis technique.

		Text analysis technique						
		SIR	NLP	CE	IE	Ct	Cf	WM
Architecture-related software artifact	ArD	13	9	1	6	0	0	0
	SCC	6	0	6	3	11	2	0
	RD	3	2	1	6	1	1	0
	CA	4	2	1	1	0	0	3

regarded as ArDs. ArDs can bridge the gaps between requirements and technical solutions, and allow stakeholders to better understand the software system (Clements et al., 2010). ArDs are useful for textual architecture information analysis. For example, in [S3], the authors applied NLP techniques (i.e., named entity extraction) to automatically extract architectural knowledge from ArDs. Table 9 shows that SIR and NLP are the most frequently used text analysis techniques (13 and 9 studies, respectively) in processing ArDs.

Source Code and Comment (SCC). Twenty-two studies used SCCs as the architecture-related artifacts to be analyzed by text analysis techniques. Comments in source code are written in natural language and briefly explain a function of code and what the code does. Mining source code comments can be useful to uncover software features, patterns, and analyze architecture-related information (Andritsos and Tzerpos, 2005). In [S21], source code as the input is analyzed by clustering technique. To be specific, the approach first analyzes the entities (e.g., classes) of the source code and its comments, and then LSI was used to cumulate the dissimilarity between the entities. These entities were further partitioned into software modules based on K-means algorithm. This approach is helpful for AR by decomposing the system into meaningful subsystems. Table 9 shows that the most frequently used text analysis technique to SCCs is clustering and the specific algorithm is K-means.

The rest of the architecture-related artifacts used in the studies are classified into ***Requirements Document (RD)*** and ***Community Artifact (CA)*** (Xu et al., 2005), and they represent 19.7% (12 studies), 11.5% (7 studies) of the selected studies, respectively. RDs are written in natural language that records specific requirements of a system. System requirements specify functions that the system must be capable of performing, and these functions are defined as Functional Requirements (FRs) (Chung et al., 2012). In the design phase, a set of quality attributes of the system are determined. These qualities are expressed as Non-Functional Requirements (NFRs), such as maintainability and security, which are crucial for the development of the system. In this SMS, we regarded functional requirements and non-functional requirements documents as the same category (i.e., RD). Applying text analysis techniques to RDs can help to capture architecture-related information (Egyed et al., 2001). For example, in [S32], the authors developed a tool (i.e., ArcheR) to identify Architecturally Significant Functional Requirements (ASFRs) in requirements documents. These ASFRs were further classified into different categories of architectural impact they have. IE is the most frequently employed text analysis technique in analyzing RDs as shown in Table 9.

Open source development communities allow stakeholders to use free-form textual description and discussion for recording information about development. Community Artifacts (CAs) such as emails, discussion forums, and bug reports may contain architecture-related descriptions and discussions (Henttonen and Matinlassi, 2009), (Ding et al., 2015). For example, in [S5], the authors proposed a framework that empowers virtual communities (i.e., e-mails, meeting notes, Wikis) to retrieve architectural knowledge. In [S38], the authors applied classification techniques in bug reports to extract design rationale of open source software. SIR and NLP are the most popular text analysis techniques employed to support mining or recovering architecture information in CAs. In [S12], the authors employed an approach to identify major concepts in Wikis about software architecture. This approach enables architects to understand important concepts of architecture and to identify the gaps in Wiki's current architecture content.

4.5. RQ4: Tool support for textual architecture information analysis

Tools are essential for the application of text analysis and knowledge retrieval. In this SMS, we collected tools from the selected studies that can support text analysis techniques employed in enhancing architecting activities. There are 58.1% (i.e., 32 out of 55) studies that used various tools to support textual architecture information analysis. We classified the tools into two types: general tool (GT) and dedicated tool (DT). GTs are usually can be applied in various situations, not limited in architecting activities. For example, GT2 (i.e., Stanford NLP

Table 10
General tools for textual architecture information analysis.

ID	Tool name	Description	Studies
GT1	GATE API	A development framework to build information extraction tools, which can be used for textual architecture information extraction.	[S1][S13][S14][S27][S34][S38]
GT2	Stanford Parser	An implementation of a probabilistic natural language parser that can be used to analyze the grammar of textual architecture information.	[S2][S27][S51]
	Stanford POS tagger	It can be used to automatically assign a grammatical label to every word in a sentence of architecture-related software artifacts.	[S2][S6][S16][S27]
GT3	Weka	It is a machine learning tool that supports data mining tasks, and can be used to classify and cluster textual architectural information.	[S2][S32][S38][S40][S49]
GT4	Bunch	It implements several clustering algorithms for automated architecture recovery.	[S33][S35][S44]
GT5	Lucene	It provides keyword-based search results over the set of architectural artifacts.	[S6][S22]
GT6	OpenNLP	It can be used to split input architecture-related text into a set of sentences and divide sentences into terms.	[S8]
GT7	LingPipe	It can be used to process textual architecture information based on computational linguistics.	[S16]

toolset) is frequently used in textual information pre-processing tasks. However, DTs are developed to support specific architecting activities.

Seven tools are classified as general tools. “GT1: GATE API” (6 studies, 10.9%), “GT2: Stanford NLP toolset” (7 studies, 12.7%), and “GT3: Weka” (5 studies, 9.1%) are widely used in text mining and machine learning fields. For examples, Weka is a machine learning tool implementing a set of algorithms that can be used in textual information clustering and classification [S38]. “GT4: Bunch” is used in 3 studies. “GT5: Lucene” is used in 2 studies (3.6%). “GT6: OpenNLP”, and “GT7: LingPipe” are used in 1 study. The detail information including the tool name, description, and the studies the tools were used is provided in Table 10.

Automatic tools can save time and energy to process, manage, and analyze textual architecture information. There are about 32.7% studies (18 out of 55 studies) developed specific tools to support textual architecture information analysis. We collected dedicated tools (DT1 -DT20) from the selected studies (in which [S22] presented two dedicated tools). The detail information of the tools is shown in Table 11, which lists the names, descriptions, and the employed text analysis techniques of the tools. Several tools do not have an exact name, and are noted as “Not mentioned”.

4.6. RQ5: Benefits and limitations of textual architecture information analysis

The benefits and limitations of using textual architecture information analysis were discussed in most of the selected studies. In this section, we used qualitative analysis approach to analyze benefits and limitations to answer our research question, i.e., data item D12 and D13 in Table 4. The execution of the qualitative analysis approach is detailed in Section 3.2.4. The benefits and limitations of applying textual architecture information analysis are summarized in Section 4.6.1 and 4.6.2, respectively. These benefits and limitations are judged to be more reliable if they are based on real-life industrial projects. Such research projects could mine industry documents and/or ask practitioners to participate in the projects. Table 12 lists the selected studies with their evidence and evidence levels, and Fig. 8 shows the distribution of the selected studies over the evidence levels. The evidence levels used in this study are adopted from (Alves et al., 2010).

Level 1: No evidence.
Level 2: Evidence obtained from demonstration (e.g., toy examples).
Level 3: Evidence obtained from expert opinions (e.g., surveys or interviews from researchers and practitioners).
Level 4: Evidence obtained from academic studies (e.g., controlled experiments conducted in an academic setting, e.g., with students).
Level 5: Evidence obtained from industrial studies (e.g., case studies conducted in an industrial setting, e.g., with practitioners).
Level 6: Evidence obtained from industrial practice (e.g., the method has already been adopted by industrial originations).

4.6.1. Benefits

The benefits about which aspects text analysis techniques can facilitate architecting activities are discussed in this section. The benefits were classified into five main types (i.e., “*Architecture knowledge*”, “*Stakeholders*”, “*Reduced efforts*”, “*Traceability*”, and “*Maintenance*”), and each type is subdivided into subtypes (see Table 13).

“B1: *Architecture knowledge*” is the most frequently discussed benefit in the application of textual architecture information analysis. There are 53 studies (i.e., 96.4%) that focus on various aspects of benefits for architecture knowledge, such as architecture knowledge extraction, architecture recovery, architecture understanding, and architectural issues identification. For example, studies by the subtype “*Architecture knowledge extraction*” focus on applying text analysis techniques in architecture-related software artifacts to identify useful architecture information. In [S5], the authors proposed a framework that can be used for architecture information search in the textual artifacts generated in emails and wiki etc.

Subtype “*Architecture understanding*” aims at gaining the better understanding of architecture design and design rationale throughout the software system development. In [S7], the authors used LSA to detect the semantic structure in architecture documents for the purpose of improving the understanding of SA. The hierarchical clustering algorithm applied in [S41] support the bottom-up architecture recovery process that assists stakeholders in gaining the architectural understanding of a system. In this paper, the authors conducted case studies in four OSS projects to evaluate the algorithms on clustering results of architecture recovery, and they found that hierarchical clustering is an important factor for gaining architectural understanding.

“B2: *Benefits for stakeholders*” refers to how stakeholders benefit from textual architecture information analysis. The selected studies employ various text analysis techniques for different architectural purposes that support specific types of stakeholders. These types of stakeholders were explicitly mentioned in the benefits of the approach presented in the selected studies, and we then extracted and identified four types of stakeholders that can benefit from textual architecture information analysis. For example, the approach proposed in [S1] uses information extraction technique to recover architectural rationale. This approach can assist training new architects or less experienced team members to gain a better understanding of how and why architecture decisions were taken. In [S16], the authors applied NLP and machine learning techniques in requirements documents to classify related NFRs into architectural concerns. This approach can guide architects in linking architectural concerns from problem domain to the running components and connectors of the solution domain. In [S27], the authors utilized NLP techniques (e.g., POS tagging) and ontology-based analysis in architecture documents and source code to identify potential components. This approach can mine architecture knowledge (e.g., data and control communications between components and implemented design patterns), which can assist the maintainers to comprehend the system architecture by identifying components and

Table 11
Dedicated tools for textual architecture information analysis.

ID	Tool name	Description	Employed text analysis techniques	Study
DT1	ArcheR	It was used to automatically identify the architecturally significant functional requirements from requirement specification documents.	Classification	[S32]
DT2	TREx	It annotated architectural related documents to identify and retrieve architectural entities (including issues, drivers, rationale)	Information extraction	[S1]
DT3	CASE	It was used to semantic search for architecture documents stored in the repository based on the semantic annotations.	Information retrieval	[S11]
DT4	Revealer	It enables fast extraction of patterns to allow developers quickly comprehend the specific architectural aspects based on the source code patterns	Information extraction	[S15]
DT5	NFR2AC	It was used to extract corresponding Architectural Aspects (AA) and Quality Attributes (QA) from NFRs.	Natural language processing	[S16]
DT6	AAKET	It was used to capture architectural knowledge from documents and emails.	Information extraction	[S18]
DT7	TEAM	It was used to visualize architecture with traceability links.	Concept extraction	[S22]
DT8	Traceclipse	It was an information retrieval tool that generated traceability links.	Information retrieval	[S37]
DT9	KaitoroCap	It was used to capture users' architecture document exploration paths. Then save the paths for future analysis.	Information extraction	[S24]
DT10	TRASE	It was a search engine over the artifacts of software architecture with relevant artifacts and architectural components.	Concept extraction	[S22]
DT11	LISA toolkit	It provides various views and editors for visualizing, editing, and analyzing architectural information.	Information extraction	[S36]
DT12	Hapax	It was used to analyze the distribution of the concepts over the system's structure.	Information retrieval	[S42]
DT13	PAKME	It was used for searching, extracting, and using architecture knowledge to support architecting process.	Information retrieval	[S43]
DT14	Design Abstractor	It clusters a class-level design graph into a subsystem-level graph that represents an architecture module view.	Clustering	[S45]
DT15	BUDGET	It collects source code files from public code repositories, and then it can sample the source code files that are related to the given architectural tactics.	Web mining, Text classification	[S55]
DT16	Not mentioned	It was used to extract design information from architectural diagrams.	Information extraction	[S10]
DT17	Not mentioned	It was a user's profiling tool based on text mining, which serve to link the stakeholders to the architectural documents	Information extraction	[S6]
DT18	Not mentioned	A tool can be used to recommend specific SAD documents for each stakeholder.	Natural language processing, Information extraction	[S8]
DT19	Not mentioned	A tool that allows users to refine the recovered architecture layers	Information extraction	[S28]
DT20	Not mentioned	A tool was used to identify architecture layers for understanding and evolution of the system.	Concept extraction	[S47]

studying their properties.

"B3: *Reduced efforts*" denotes the benefits of applying automatic or semi-automatic textural architecture information analysis that can reduce the enormous manual efforts. As examples of the subtypes "*Reduced time*" and "*Reduced manual effort*", the approach in [S3] conducted NLP techniques to extract architecture knowledge from architectural related documents. This approach can reduce a large quantity of manual decisions on identification work (i.e., lowest annotation time). In this work, the authors conducted a comparison experiment. The results show that the manual approach spent nine minutes for the architectural issues annotation, the semi-automated approach spent three minutes, and automated approach spent three seconds. In addition, the automated and semi-automated approach perform relatively well (i.e., with the highest recall and the highest precision, respectively). By using this approach, the knowledge engineers do not need to go through the whole documentation and they can reject the sentences that (s)he does not consider as architecture entities. In [S36], the authors used IE technique to extract architectural information for support the several activities (e.g., design and quality control) of service-oriented architecture (SOA) systems. This approach can reduce time required for analysis and eliminate potential human errors in the extraction process. While, the subtype "*Accuracy and precision rates*" denotes the application of using textual architecture information analysis can reach high accuracy and precision rate, which indicate the approaches or methods are effective and reliable.

"B4: *Traceability*" relates to the benefits in constructing traceability links between various architecture-related software artifacts, components, and implementations through applying textual architecture information analysis. For example, in [S2], the authors employed NLP (e.g., POS tagger) and clustering techniques to extract and gather functionalities in requirements documents (see details of the approach in Section 4.2). This approach can identify the potential responsibilities and components in the final architecture, which bridges the gap between and relates requirements and architecture design. In [S31], the authors used information retrieval and classification techniques to identify architecture tactic-related classes in source code. This approach can be used to automatically construct traceability links between the source code and architectural tactics. This approach minimizes the human effort required to establish traceability that can used to support maintenance activities, in order to help mitigate the pervasive problem of architectural erosion. There is no study that mentions the subtypes of "*Traceability*", as such we did not list the subtypes of "*Traceability*" in Table 13.

"B5: *Maintenance*". Studies classified in this category outline the benefits for maintenance. Knowledge can aid maintenance, indirectly, by providing software traceability (B4) and additional knowledge. For example, in [S29], the authors applied LSI technique to recover architectural concepts embedded in source code by clustering variable names based on their similarity. Identifying these concepts by clustering the variable names instead of clustering documents results in more precise clusters and such refined knowledge can provide more refined knowledge. The results of the experiments show that this approach is helpful to software maintenance through understanding of architecture. In [S33], the authors proposed a clustering technique based on structural relations of classes (e.g., associations, generalizations, and dependencies relations) that can be used to reconstruct static architectural views (i.e., the relationships between components) of object-oriented software. The results show that the approach is beneficial for software maintenance through understanding the reconstructed architectural views. In [S44], the authors stated that the tool (i.e., Bunch) is valuable for performing software maintenance activity because it provides useful architecture-level views of a system's structure. Note that, there is no discussion of any specific maintenance tasks by any of the studies, therefore no subtype benefits of "*Maintenance*" are listed in Table 13.

Table 12
Evidence and evidence levels of the selected studies.

No.	Evidence of the selected studies	Evidence level
[S1]	A case study with a financial securities settlement system	Level 5
[S2]	Case studies with two industrial projects and one academic project	
[S5]	Data from the Brazilian Public Software community	
[S6]	Data from InfoQ.com a practitioner-driven community site	
[S7]	Data from 80 documents of a software product audit company	
[S8]	A case study with an OSS project	
[S9]	Data from the architecture documents of an Electronic Voting system	
[S14]	A case study with an OSS project	
[S15]	A case study with an industrial project (a distributed intrusion detection system)	
[S16]	A case study with an industrial project (the automotive domain)	
[S17]	Case studies with nineteen OSS projects (from various application domains)	
[S18]	Data from e-mails between architects of an architecture development team	
[S19]	Case studies with five OSS projects	
[S20]	Case studies with two OSS projects	
[S21]	Case studies with three OSS projects	
[S22]	Data from a set of artifacts of the ArchStudio 4 system	
[S23]	Data from the Brazilian Public Software community	
[S24]	Data from the architecture document of an industrial software product	
[S25]	Data from three documents of an industrial software product	
[S26]	Case studies with three enterprise architecture models	
[S27]	A case study with an OSS project	
[S28]	Case studies with four OSS projects	
[S29]	A case study with an industrial system	
[S30]	Case studies with three OSS projects	
[S31]	Case studies with ten OSS projects	
[S32]	Data from thirty requirements specification documents (the insurance domain)	
[S33]	A case study with a printer controller system	
[S35]	Case studies with four software systems (System A and B are legacy systems used at university, and Systems C and D deal with administrative university routines)	
[S36]	Case studies with several industrial projects (the banking domain)	
[S38]	Data from Chrome bug reports	
[S39]	Case studies with thirteen OSS projects	
[S40]	Case studies with fifty OSS projects	
[S41]	Case studies with four OSS projects	
[S42]	Case studies with two OSS projects	
[S44]	A case study with an industrial system	
[S45]	A case study with an industrial system	
[S46]	Data from an industrial cruise-control system	
[S47]	Case studies with seven software systems (three OSS projects and four projects developed by students at university)	
[S50][S55]	Data from source code of 116,000 OSS projects	
[S48]	Case studies with several industrial systems	
[S51]	Data from requirements and architecture documents of industrial projects	
[S52]	Data from three architecture documents of an industrial project	
[S54]	Data from source code of industrial projects (web applications)	
[S3]	A controlled experiment conducted with 19 students	Level 4
[S4]	Data from source code of a software system developed at university	
[S10]	Data from (1) the architecture documents created by students in a course, and (2) paper-scanned diagrams from an architecture documentation book	
[S11]	Data from several architecture books	
[S34]	Data from several architecture books	
[S37]	A controlled experiment conducted with 74 students	
[S53]	A controlled experiment conducted with 190 students	
[S12]	Data from Wikipedia related to software architecture	Level 3
[S13]	Data from glossaries, thesaurus, and literatures on software architecture with the knowledge from domain experts	
[S49]	Data from Stack Overflow related to software architecture	
[S43]	Examples from an architecture knowledge repository that contains generic and project-specific architecture knowledge	Level 2

4.6.2. Limitations

The limitations refer to the obstacles of applying textual architecture information analysis. We extracted the limitations from the selected studies and classified them into three categories. The main types, subtypes, and detail description of the limitations were listed in Table 14. Twenty-four studies did not mention limitation explicitly.

"LI: *Data source availability*": Studies classified in this category refer to the quality and size of the input data source (e.g., training data), which influences the performance (e.g., accuracy rate) of applying textual architecture information analysis. For example, in [S1], the authors mentioned that the extracted metadata (e.g., architectural rationale) is heavily dependent on the quality of the available textual artifacts. The machine learning approach in [S3] did not work well due to lack of sufficient training data; therefore, the authors only used the rule-based approach in an NLP framework to extract architectural entities from the architecture documents.

"L2: *Efforts and costs of text analysis*": Studies in this category refer to the required efforts (e.g., the time, labor costs, and prior knowledge) for applying textual architecture information analysis. For example, in [S2], the authors applied NLP and clustering techniques in analyzing requirements documents to obtain the functional decomposition of a system. This approach can guide architecture design during early stages of development process. While, this approach is very hard to accomplish and require a lot of effort, time, and experience from requirement analysts and software architects.

"L3: *Other influential factors*": Studies in this category denote the factors that impact the performance of textual architecture information analysis. For example, in [S6], authors mentioned that the tools they

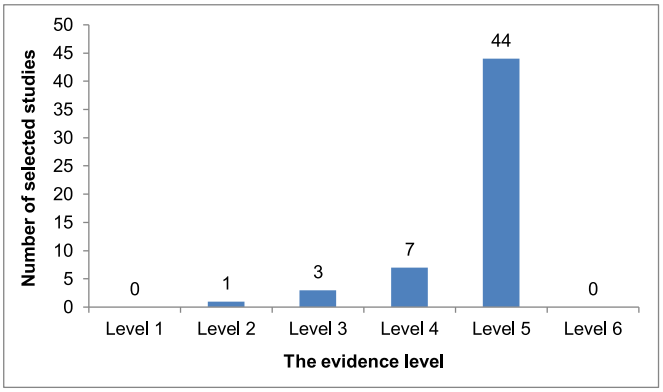

Fig. 8. Number of studies over the six evidence levels.

used or developed still need improvements, mainly on aspects of architecture knowledge extraction.

5. Discussion

Textual information in software development documents potentially contains useful information to support various architecting activities. This information is typically unstructured and buried within different places. Developers can use this information to obtain a better understanding of a system if there ways to extract the textual information. We conducted this SMS to understand the current state of textual information extraction and analysis techniques and tools for software architecting activities. In this section, we discuss our findings and their implications.

5.1. Extraction and use of knowledge

Each phase of a software development lifecycle produces a set of artifacts that can be mined to obtain architectural knowledge. To apply text analysis techniques in enhancing architecting activities, it is useful to understand the characteristics, performance, and behaviors of various text analysis techniques, the architecture-related software artifacts, and the architecting activities. The relationships between text analysis techniques, architecture-related software artifacts used by these techniques, and the architecting activities that the resulting knowledge supports are shown in Fig. 9. The dotted lines show the text analysis techniques that are used to create software artifacts. The solid lines represent that the architecture-related software artifacts are then analyzed for helping certain architecting activities. We illustrate the relationships between the analyzed architecture-related software artifacts, architecting activities, and internal quality through analyzing the architectural purposes (in rectangle boxes). The thickness of a line denotes the number of the selected studies. Note that, we used the abbreviations for representing specific text analysis techniques in Fig. 9, in order to make the dotted lines easy to trace. According to the analysis results in Fig. 9 and Section 4, we have a number of interpretations that are discussed below:

Architectural understanding: Thirty-one studies use text analysis techniques to assist Architectural Understanding (AU). Examining Fig. 9, we see 27 studies that retrieve AK (e.g., rationale and concepts) and 9 studies that establish traceability links for assisting AU. The key extracted AKs are design decision and rationale. Architectural understanding is also achieved by establishing traceability links between artifacts (e.g., architecturally significant requirements, architecture tactics, and source code). However, there are several limitations, which hinder textual architecture information analysis for AU. For example, [S1] mined architecture rationale and it suffered from L1 (i.e., quality of data) and L3 (i.e., text analysis technique limited). One of the limitations is due to flexible languages (i.e., synonyms and homonyms) to name concepts. For instance, [S34] cited the difficulty that a lack of standardized terminology could have an impact on how the concepts are understood. As such, a naming standardization and convention used amongst developers could greatly facilitate the accuracy of textual architecture information analysis.

Architectural information recovery: Architectural description is often incomplete in a system. Knowledge and concepts such as rationale, views, and structures are often absent. It is critical to recover such architectural information that can help to provide a complete and unambiguous description of the architecture of a system. Fig. 9 shows that recovering system structure is a fundamental aspect of architecture information recovery. Eleven studies attempted this and they are mainly based on component recovery (e.g., [S4][S8][S21][S33][S47]) by identifying components and connectors and the related structure between components (Koschke, 2002).

Extraction of architectural information from low-level information, such as source code, of a system is one way to uncover hidden structure of a system. Ct techniques (including partitional and hierarchical clustering techniques) are frequently used text analysis techniques to analyze SCCs. The recovered structure of a given system based on Ct techniques can identify components that implement similar or different functionalities. Recovery of architecture view (4 studies), recovery of architecture rationale, tactics, and concept (7 studies), and recovery of architecture documentation (1 study) are not widely explored by applying textual architecture information analysis. This is mainly because these tasks are heavily dependent on the quality of the source data (e.g., architecture documents, wiki, meeting notes), and they are usually not well documented. For example, design rationale captures the knowledge that justifies the resulting design. However, architects do not document even key rationale information (e.g., discarded solutions and tradeoffs) (Tang et al., 2006). In order for AR to be more effective, one may use multiple information sources to enrich and complement each other. That means text analysis techniques need to analyze and collate multiple sources of information.

Difficulties in end-to-end architectural knowledge extraction: AK extraction is the most popular purpose for applying textual architecture information analysis (i.e., 37 out of 55 studies from Fig. 9). AK contains design rationale, design argumentation, design concerns, alternative solutions, significant entities, and general knowledge that can provide knowledge for most architecting activities, especially AU. AK is often analyzed or extracted from ArDs and RDs, mainly because it is typically generated and consumed in early requirement and architectural analysis phases. ArDs and RDs are some of the main textual deliverables containing specific textual architecture information. AK also can be extracted from other forms of architecture-related software artifacts (such like bug report) in subsequent development phases, but SCCs are used in AK extraction. Even though top-down approaches can relate requirement, architectural decision, and other forms of high-level knowledge to the source code. It is difficult to mine SCCs to retrieve the underlying AK (e.g., rationale) when the information is undocumented. Each architectural concern may be realized by multiple code segments and are intertwined in numerous program files. The relationships between code and their architecture knowledge are hard to reveal (Zapalowski et al., 2014). The issue is a lack of overview of the structure of the system, linking to the source code and program files (Clements et al., 2010).

Architecture-centric traceability: From the traceability box in Fig. 9, we see 9 AU studies and 2 AME studies that depend on traceability. When researchers analyze textual architecture information, two architecting activities (i.e., AU and AME) can benefit from traceability. The sheer volume of artifacts produced in a project, the differing levels of formality and specificity between documents and the complex interrelationships between artifacts make the understanding of the relationships between artifacts very complex. As discussed in Section 4.6.1, text analysis techniques (e.g., topic modeling) can to some extent support automatic or semi-automatic architecture-centric

Table 13
Classification for benefits of applying textual architecture information analysis.

ID	Benefit type	Subtype	Description	Studies
B1	Architecture knowledge	Architecture knowledge extraction	Text analysis techniques can enable the identification or extraction of architecture knowledge in architecture-related software artifacts.	[S1][S5][S6][S7][S10][S13][S14][S18][S19] [S24][S25][S26][S28] [S29][S32][S33][S35][S37][S38][S40][S43] [S46][S49][S51]
		Architecture understanding	Applying textual architecture information analysis can achieve a better understanding of architectural design decisions and rationale.	[S1][S5][S6][S7][S8][S10][S12][S15][S16] [S19][S22][S25][S26] [S32][S36][S38][S40][S41][S42][S51][S52] [S53][S55]
		Architecture knowledge recovery	Text analysis techniques can help recover architectural concept, documentation, and rationale that not explicit.	[S1][S4][S10][S19][S28][S35][S39][S41][S42] [S44][S45][S50][S54]
		Components identification	Text analysis techniques can help to identify components from architecture-related software artifacts.	[S4][S8][S10][S21][S27][S28][S33][S47][S48]
		Architectural issues identification	Text analysis techniques were applied in technical and projects documentation, which can help to identify architectural issues.	[S1][S3][S4][S31][S40]
B2	Benefits for stakeholders	Benefits for architects	Enable architects to identify architectural information and rationale, and guide architects for making decision effectively.	[S1][S2][S5][S6][S7][S11][S24][S31][S34] [S43][S49][S52][S53][S55]
		Benefits for maintainers	Maintainers can employ textual architecture information analysis in existing architecture-related software artifacts to get a comprehension of the system.	[S5][S6][S14][S24][S27][S38]
		Benefits for developers	Textual architecture information analysis can help developers to understand knowledge about views of modules and components-and-connectors.	[S5][S6][S8][S11][S12][S19][S24]
		Benefits for documenters	Text analysis techniques assist the documenters in his/her documentation tasks and update architecture documents.	[S5][S6][S7][S14]
B3	Reduced efforts	Accuracy and precision rates	The approaches can achieve a high accuracy and precision in identifying and recovering architectural information.	[S1][S2][S6][S19][S20][S30][S33][S34][S35] [S40][S55]
		Reduced time	The application of textual architecture information analysis can reduce the time for recovering, mining, and analyzing architecture information.	[S18][S33][S36][S38]
		Reduced manual effort	Text analyze techniques can reduce a quality of manual effort to search, mine, and analyze architectural information.	[S3][S18][S38][S55]
B4	Traceability	Not mentioned	The application of textual architecture information analysis provides useful semantic information that can be used to construct the semantic relationships between architecture-related software artifacts (e.g., components).	[S2][S8][S22][S31][S37][S40][S42][S55]
B5	Maintenance	Not mentioned	Textual architecture information analysis can assist maintenance activity.	[S27][S29][S32][S33][S38][S39][S40][S44]

Table 14
Classification for limitations of applying textual architecture information analysis.

ID	Limitation type	Subtype	Description	Studies
L1	Data source availability	Quality of data source	The performance of text analysis techniques in recovering, mining, and analyzing architecture information is largely based on the quality of the input data source.	[S1][S8][S13][S17][S22][S23][S24][S42][S49][S54]
		Size of data source	A lack of sufficient data source (e.g., training data) will influence the performance of the textual architecture information analysis.	[S3][S13][S54]
L2	Efforts and costs of text analysis	Prior knowledge	Some prior knowledge about text analysis techniques and software architecture are required.	[S2][S6][S12][S13][S18][S34][S41]
		Time and labor	The textual architecture information analysis requires a lot of time and experience to training data (e.g., manual annotation processing for textual data is expensive).	[S2][S4][S5][S34]
L3	Other influential factors	Limitation of techniques	The existing text analysis techniques cannot extract all architecture knowledge from the textual artifacts.	[S1][S4][S11][S38][S41]
		Tools	More sophisticated and extensible tools are needed that support text analysis techniques in mining and recovering architecture knowledge.	[S1][S5][S6][S16]
		Size of software system	It is difficult using text analysis techniques to identify components in large systems.	[S15][S20]

traceability link construction (i.e., using the architecture as the primary artifacts to trace other artifacts). Traceability can be classified into vertical and horizontal traceability according to whether the relations associate with different levels of software artifacts (Spanoudakis and Zisman, 2005). Horizontal traceability aims at tracing artifacts at the same level, for example tracing requirements documents of different versions with the concerned quality attribute (e.g., performance) of the system, while vertical traceability is constructed at different levels of artifacts (Cleland-Huang et al., 2012), such as the traceability from requirements to design artifacts. We found that all selected studies focus on constructing vertical traceability links between the artifacts produced in different phases of the software development life cycle. No work has been done on constructing horizontal traceability.

Software maintenance: Software maintenance has always been a challenging issue especially with the increase of the size and complexity of systems (Grubb and Takang, 2003). From the maintenance box in Fig. 9, we see 14 studies discuss extraction and analysis of architectural information based on text analysis techniques can help maintainers, who often did not develop the original software, understand a system better. For example, text analysis techniques can be used for component

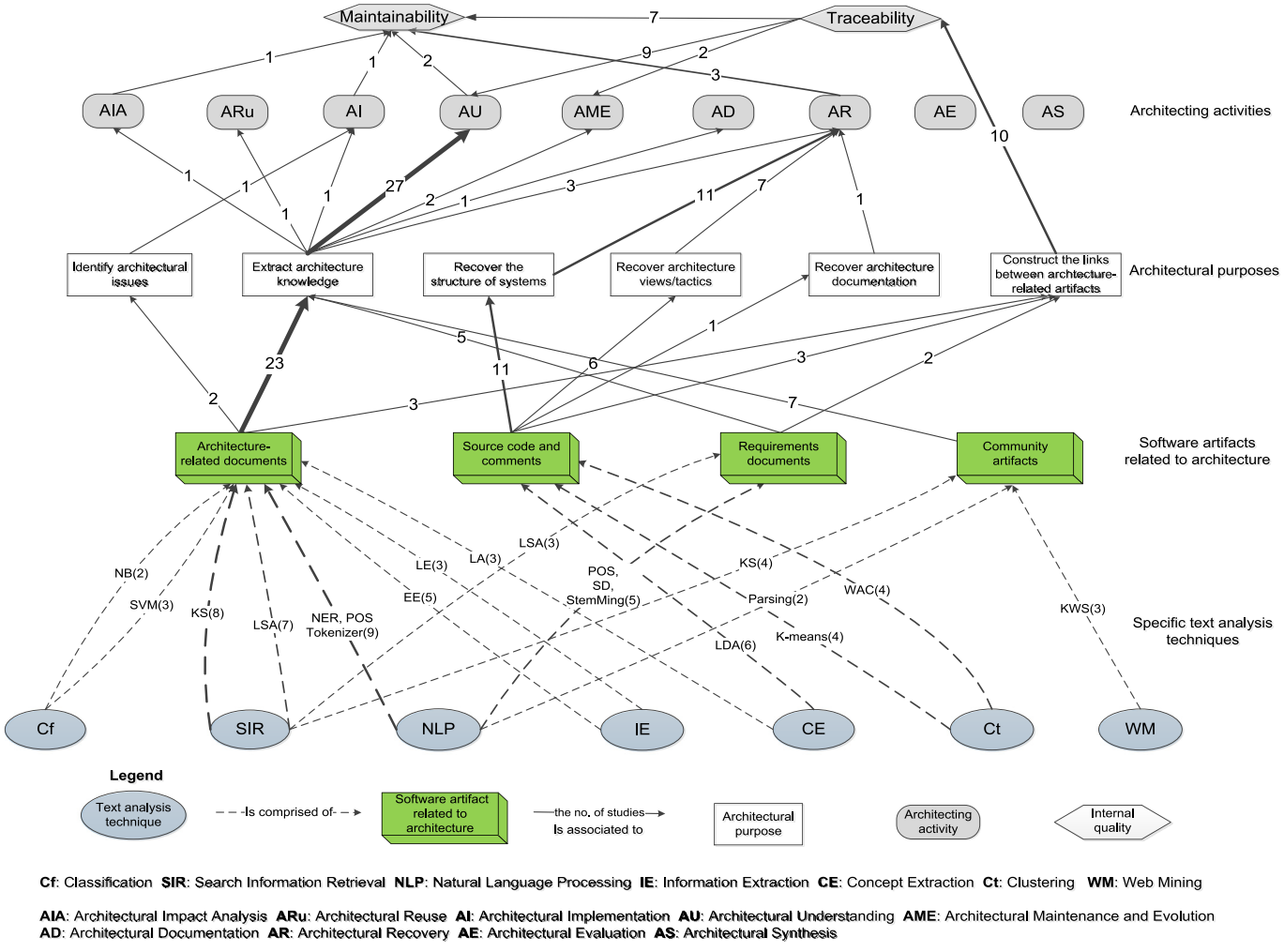

Fig. 9. Relationships between text analysis techniques, architecture-related software artifacts, and architecting activities.

recovery, which assists maintainers to gain an understanding of dependence relationships between components and/or connectors [S33]. In addition, applying text analysis techniques (e.g., topic modeling) to construct traceability between architecture-related software artifacts may help maintainers analyze consistency between artifacts (e.g., [S22] [S31]).

Other architecting activities: Several architecting activities such as Architectural Synthesis (AS) and Architectural Evaluation (AE) received less attention. No studies use text analysis techniques to support AS. One potential reason is that these architecting activities (e.g., AS and AE) heavily depend on human effort and involvement (Tekinerdoğan and Akşit, 2002), (Raiha et al., 2009). For example, AS involves assessments and decision making by experts based on their experience. Similar to supporting software maintenance, automatic or semi-automatic text analysis techniques may provide useful knowledge to support these architecting activities. This area is unexplored.

5.2. Implications for researchers and practitioners

This mapping study shows that automatic and semi-automatic text analysis techniques can extract additional information from existing artifacts. This information may be used in supporting architecting activities. Researcher and practitioners can use the 55 selected studies in this SMS to search for evidence. Here we present the implications for researchers and practitioners that are summarized based on the results of this SMS.

- ***Industrial relevance***: The high evidence level can provide practitioners sufficient confidence to employ text analysis techniques in SA. We extracted the evidence and evidence levels of the selected studies (see Table 12 and Fig. 8). Forty-four selected studies are based on industrial data or projects (i.e., evidence level 5), seven selected studies are based on academic data or student projects (i.e., evidence level 4), three selected studies are based on expert opinions (i.e., evidence level 3), and one selected study demonstrates the usefulness of a tool through examples (i.e., evidence level 2). Note that fifteen selected studies conducted case studies with OSS projects, which are non-trivial systems and treated as industrial projects. The results show that the overall evidence level of the selected studies is relatively high (the mean score is 4.69), and most of the studies have evidence obtained from industrial studies (i.e., 81.2% of the studies are at the evidence level 5). However, there are no studies that provide evidence from industrial practice (i.e., evidence level 6), which shows that textual architecture information analysis has not been tested in industry practice. We encourage that the practitioners can explore the convincing benefits and limitations by applying textual architecture information analysis in industrial practice.
- ***The quality of architecture-related software artifacts***: Quality of architecture-related software artifacts is important for textual architecture information analysis. 10 studies mentioned that data quality (e.g., lack adequate of training data [S9]) will influence the performance of an approach, and there are no fundamental principles for assessing the quality of textual artifacts. A number of works aim to improve data quality, including acquiring features, formulating data, cleaning data, obtaining expert labeling of data (Sheng et al., 2008). For example, in the tasks of semi-automatic textual architecture information analysis (classification for architectural information), the ground truth of the training data is crucial to the final results. The unstructured nature of text and documents used makes the tasks of labeling data and selecting features a challenge, and recruiting professional architects to participate is sometimes difficult, thus the experiments of selected studies use student as participants to label the data. With low-cost labeling, the subsequent parts of work (e.g., preparing the test data) can become considerably more expensive than labeling. Regarding these limitations, we hope that researchers can provide a solid foundation on which future research can be built.
- ***Improving the performance of textual architecture information analysis***: The overall performance of most applications of textual architecture information analysis shows that the approaches presented in the literature are effective (i.e., F-measure is above 70%). However, in some cases, precision or and recall are affected by various factors. For example, when certain NLP techniques (e.g., stemming) were applied, precision was improved while recall fell (Buckland and Gey, 1994), and the selected textual features impact the recall of classification for architectural rationale information [S38]. One factor is that text analysis techniques (e.g., pre-processing techniques) remove certain textual feature terms that makes training data incapable to provide sufficient coverage of the concept. Further exploration in techniques combination is needed, and it is critical to conduct comparison experiments to identify potential impact factors that influence the performance of textual architecture information analysis.
- ***Text analysis techniques and architecture-related software artifacts selection***: Different text analysis techniques and architecture-related software artifacts are chosen for different architecting activities. We found that several text analysis techniques are specifically used in enhancing certain architecting activities. For example, topic modeling can be used to construct the traceability links between architecture-related software artifacts [S22], such as the links between artifacts in the problem space (architecturally significant requirements) and solution space (source code). The hierarchical clustering technique is suitable for grouping items or entities (e.g., functions or files of source code) to recover the architecture of software systems. Practitioners need to select effective and suitable text analysis techniques and architecture-related software artifacts, when conducting different architecting activities.
- ***The balance between the cost and benefit***: The cost and benefit considerations for applying text analysis techniques in architecting activities is important. For example, meaningful and available input data is vital for the final analysis results. In certain cases, the effort and time required to collect and analyze the data is expensive (Lethbridge et al., 2005). Especially the quality can vary between textual artifacts and low quality textual artifacts might not convey any necessary architectural information (see L1 in Table 14). In addition, some tasks are time-consuming, such as the careful peer-review process on the input data source (see L2 in Table 14). The balance of cost and benefits of employing text analysis techniques requires further exploration and investigation.
- ***Textual architecture information analysis in OSS***: Having an SA is one of the characteristics of successful open source software (OSS) (Fielding, 2005). OSS projects often produce a large amount of potentially useful textual artifacts. Textual architecture information analysis can be helpful for stakeholders to understand software systems and facilitate their daily tasks. In social media of OSS development, the developers discuss high-level concepts, such as features and domain concepts, which are architecturally relevant (Pagano and Maalej, 2013). However, architectural information is often not recorded in OSS projects, and a lack of architecture documentation is problematic for OSS development. Text analysis techniques can be beneficial for researchers and practitioners in analyzing architectural information, identifying the rationale of architecture design, and constructing traceability, which are crucial for the evolution and maintenance of OSS (Ding et al., 2015).
- ***Tool support for textual architecture information analysis***: Using tools is important for textual architecture information analysis, especially in architecture knowledge extraction. As presented in Section 4.5, there are approximately 60% (i.e., 32 out of 55) studies that employ various tools for supporting textual architecture information analysis. The use of different tools (see Table 10 and Table 11) depends on the characteristics of architecture-related

Table 15
An assessment of specific text analysis techniques usage in architecture-related software artifacts.

Architecture-related software artifact	Specific text analysis technique					
	NLP	Hierarchical clustering	Topic Modelling	Latent Semantic Analysis	SVM, Naïve Bayes	Keyword Search
ArD	Extract AK		Traceability construction	Extract AK	Identify architectural issues	
RD	Extract AK		Traceability construction			
SCC		Recover the structure of systems			Recover architecture views and tactics	
CA						Extract AK

software artifacts and architectural purposes. As such, researchers and practitioners can select the appropriate tools according to the artifacts available and the purposes to be achieved when they conduct specific textual architecture information analysis.

- ***An assessment of the results***: We summarized the important relationships between the frequently employed text analysis techniques and architecture-related software artifacts in Table 15 according to Fig. 9. This table can provide suggestions to researchers and practitioners about which specific text analysis techniques are more suitable and promising in analyzing which category of architecture-related artifacts regarding the specific architectural purposes.

6. Threats to validity

We analyze the validity threats to our research according to four types of validity threats (Zhou et al., 2016), (Wohlin et al., 2012).

Construct validity focuses on whether we collected data correctly. The main constructs of this SMS are the two concepts “software architecture” and “text analysis technique”. Inappropriate search terms and strategy may introduce a large quantity of irrelevant results, and several relevant studies could not be retrieved. To mitigate this threat, a pilot search was conducted before formal search to improve the suitableness of the search terms. In addition, to ensure the completeness of the search terms that can cover the potentially relevant studies, we checked the definition of text analysis techniques of the related literatures and discussed among all the authors to decide the final search terms. We then chose seven popular electronic databases in computer science to search primary studies. The risk of omitting relevant studies was reduced through above measures. In addition, when conducting study selection, a researcher may exhibit certain bias because of his/her own understanding and subjectivity. We conducted two strategies to mitigate the threat caused by personal bias. Firstly, a pilot data selection was conducted before the formal study selection in order to guarantee that all the authors had a consistent understanding of the selection criteria. Secondly, the first and fourth authors conducted the study selection in the third round independently, and all the authors discussed the controversial studies to achieve a consensus on the final selection results.

Internal validity refers to the factors that may have an influence on the analysis of the extracted data. To ensure that the quality of the extracted data can be used to answer the research questions, several measures were conducted to mitigate the bias of researchers who performed data extraction. We conducted a pilot data extraction and analysis using ten studies before the formal data extraction and analysis, and then all the authors discussed the results of the pilot to reach an agreement. In the formal data extraction and analysis, the first author carried out the data extraction and analysis, the second author reviewed the results, and all the authors discussed the controversial data results to reach a consensus. In addition, we employed a descriptive statistics method to present the results of this study, and this threat is minimized.

External validity refers to the degree to which the findings of a study can be generalized. The results of this SMS provide an overview of the state of research on text analysis techniques in software architecture. In order to achieve external validity, we developed a search protocol to obtain a wide and representative collection of studies. We conducted the study search in seven most popular databases, and we did not limit the start time of study search in order to mitigate the risks of omitting potential studies. The wide ranging of literature reviewed allows us to provide a holistic map of text analysis techniques used in architecting activities.

Conclusion validity refers to whether the study yields the same results if other researchers replicate it, which is related to the factors such as missing studies and incorrect conclusions. Bias on collecting and analyzing data may affect the interpretation of the results. By making explicit the processes of data collection and analysis of this SMS (as detailed in Section 3.2), we argue that this threat is partially mitigated. In addition, the relationships between the extracted data and the conclusions were strengthened through the statistical information and qualitative analysis.

7. Conclusions and future work

Given the large size and complexity of software systems, architecture became a vital concern (Bass et al., 2012). With the development of the systems, rich textual artifacts (e.g., requirements documents) are produced that can provide insights for SA, consequently, the use of various text analysis techniques in textual artifacts to extract or mine desired information is feasible and critical in software development. In this SMS, we found 55 relevant studies in seven electronic databases.

We extracted the data from the selected studies to answer five research questions. We identify and analyze the popularities of applied text analysis techniques, the supported architecting activities, analyzed architecture-related software artifacts, and tools. We find five general benefit categories (i.e., architecture knowledge, stakeholder benefit, reduced efforts, traceability, and maintenance) and three limitation categories (i.e., quality of data source, efforts and costs, and other influential factors) that concern textual architecture information analysis. This mapping study shows that architecture knowledge extraction is the most important purpose, and two internal qualities (i.e., maintainability and traceability) can be benefited from the application of textual architecture information analysis. The results of the mapping study also show that text analysis techniques can be used in analyzing architectural information, which can further support architecting activities. To conclude, this SMS has analyzed and summarized knowledge for researchers and practitioners on applying text analysis techniques in software architecture, such as how to select suitable text analysis techniques, architecture-related software artifacts to be analyzed, and supporting tools to reach specific goals in assisting architecting activities.

The results of this SMS can motivate researchers venturing into several potential research directions. Architecture-related software

artifacts (e.g., architecture-related documents) may contain mixed textual and graphical architecture information, and graphical architecture information, such as UML models stored in images or other formats (e.g., XMI), is also vital for various architecting activities (e.g., architectural understanding) (Hebig et al., 2016). We could compare graphical architecture information analysis with textual architecture information analysis systematically. In addition, an attempt of combining these two techniques might be helpful for researchers and practitioners when they conduct certain architecting activities (e.g., architectural recovery). Furthermore, architectural information can also be stored as video or audio formats, and textual architecture information analysis can be employed to analyze the transcriptions.

Acknowledgements

This work is partially sponsored by the NSFC under Grant no. 61472286. The authors gratefully acknowledge the financial support from the China Scholarship Council.

Appendix A. Selected studies

[S1] C. López, V. Codocedo, H. Astudillo, and L. M. Cysneiros. Bridging the gap between software architecture rationale formalisms and actual architecture documents: an ontology-driven approach, Science of Computer Programming, 77(1): 66–80, 2012.

[S2] A. Casamayor, D. Godoy, and M. Campo. Functional grouping of natural language requirements for assistance in architectural software design. Knowledge-Based Systems, 30(6): 78–86, 2012.

[S3] M. Anvaari and O. Zimmermann. Semi-automated design guidance enhancer (SADGE): a framework for architectural guidance development, in: Proceedings of the 8th European Conference on Software Architecture (ECSA), Vienna, Austria, pp. 41–49, 2014.

[S4] A. Corazza, S. Di Martino, V. Maggio, A. Moschitti, A. Passerini, G. Scanniello, and F. Silvestri. Using machine learning and information retrieval techniques to improve software maintainability, in: Proceedings of the 2nd International Workshop on Eternal Systems (EternalS), Montpellier, France, pp. 117–134, 2012.

[S5] A. M. Figueiredo, J. C. dos Reis, and M. A. Rodrigues. Improving access to software architecture knowledge: an ontology-based search approach, International Journal Multimedia and Image Processing, 3(1/2): 143–149, 2013.

[S6] M. Nicoletti, J. A. Díaz-Pace, and S. Schiaffino. Towards software architecture documents matching stakeholders' interests, in: Proceedings of the 2nd International Conference on Advances in New Technologies, Interactive Interfaces, and Communicability (ADNTIIC), Huerta Grande, Argentina, pp. 176–185, 2011.

[S7] R. C. de Boer and H. van Vliet. Architectural knowledge discovery with latent semantic analysis: constructing a reading guide for software product audits, Journal of Systems and Software, 81(9): 1456–1469, 2008.

[S8] Q. Zhang, D. Qiu, Q. Tian, and L. Sun. Object-oriented software architecture recovery using a new hybrid clustering algorithm, in: Proceedings of the 7th International Conference on Fuzzy System and Knowledge Discovery (FSKD), Yantai, China, pp. 2546–2550, 2010.

[S9] M. Nicoletti, J. A. Díaz-Pace, S. Schiaffino, A. Tommasel, and D. Godoy. Personalized architectural documentation based on stakeholders' information needs, Journal of Software Engineering Research and Development, 2(1): 1–26, 2014.

[S10] E. Maggiori, L. Gervasoni, M. Antúnez, A. Rago, and J. A. Díaz-Pace. Towards recovering architectural information from images of architectural diagrams, in: Proceedings of the 15th Argentine Symposium on Software Engineering (ASSE), Buenos Aires, Argentina, pp. 36–50, 2014.

[S11] R. Duddukuri and T. V. Prabhakar. Helping architects in retrieving architecture documents: a semantic based approach, in: Proceedings of the 1st International Workshop on Semantic Matchmaking and Resource Retrieval (SMRR), Seoul, Korea, pp. 113–121, 2006.

[S12] L. T. Babu, M. S. Ramaiah, T. V. Prabhakar, and D. Rambabu. ArchVoc - towards an ontology for software architecture, in: Proceedings of the 2nd Workshop on Sharing and Reusing Architectural Knowledge - Architecture, Rationale, and Design Intent (SHARK/ADI), Minneapolis, MN, USA, pp. 5–11, 2007.

[S13] A. Fraga, S. Sánchez, and J. Lloréns. Knowledge representation for software architecture domain by manual and automatic methodologies, Electronic Journal of CLEI, 9(1): 2–16, 2006.

[S14] R. Witte, Y. Zhang, and J. Rilling. Empowering software maintainers with semantic web technologies, in: Proceedings of the 4th European Semantic Web Conference (ESWC), Innsbruck, Austria, pp. 37–52, 2007.

[S15] M. Pinzger, M. Fischer, H. Gall, and M. Jazayeri. Revealer: a lexical pattern matcher for architecture recovery, in: Proceedings of the 9th Working Conference on Reverse Engineering (WCRE), Richmond, VA, USA, pp. 170–178, 2002.

[S16] G. Gokyer, S. Cetin, C. Sener, and M. T. Yondem. Non-functional requirements to architectural concerns: ML and NLP at crossroads, in: Proceedings of the 3rd International Conference on Software Engineering Advance (ICSEA), Sliema, Malta, pp. 400–406, 2008.

[S17] A. Corazza, S. Di Martino, V. Maggio, and G. Scanniello. Weighing lexical information for software clustering in the context of architecture recovery, Empirical Software Engineering, 21(1): 72–103, 2016.

[S18] Aman-ul-haq and M. A. Babar. Tool support for automating architectural knowledge extraction, in: Proceedings of the 4th Workshop on Sharing and Reusing Architectural Knowledge (SHARK), Vancouver, BC, Canada, pp. 49–56, 2009.

[S19] M. Risi, G. Scanniello, and G. Tortora. Using fold-in and fold-out in the architecture recovery of software systems, Formal Aspects of Computing, 24(3): 307–330, 2012.

[S20] L. do Nascimento Vale and M. de Almeida Maia. Keecle: mining key architecturally relevant classes using dynamic analysis, in: Proceedings of the 2nd International Conference on Software Maintenance and Evolution (ICSME), Bremen, Germany, pp. 566–570, 2015.

[S21] G. Scanniello, A. D'Amico, C. D'Amico, and T. D'Amico. Using the Kleinberg algorithm and vector space model for software system clustering, in: Proceedings of the 18th International Conference on Program Comprehension (ICPC), Braga, Minho, Portugal, pp. 180–189, 2010.

[S22] H. U. Asuncion, A. U. Asuncion, and R. N. Taylor. Software traceability with topic modeling, in: Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE), Cape Town, South Africa, pp. 95–104, 2010.

[S23] A. M. C. M. Figueiredo, J. C. dos Reis, and M. A. Rodrigues. Semantic search for software architecture knowledge: a proposal for virtual community's environment, in: Proceedings of the 2nd International Conference on Information Society (i-Society), London, UK, pp. 270–274, 2011.

[S24] M. T. Su, J. Hosking, and J. Grundy. Capturing architecture documentation navigation trails for content chunking and sharing, in: Proceedings of the 9th Working IEEE/IFIP Conference on Software Architecture (WICSA), Boulder, Colorado, USA, pp. 256–259, 2011.

[S25] R. C. de Boer. Architectural knowledge discovery: why and how? in: Proceedings of the 1st Workshop on Sharing and Reusing Architectural Knowledge (SHARK), Torino, Italy, pp. 1–4, 2006.

[S26] M. Bakhshandeh, C. Pesquita, and J. Borbinha. An ontological matching approach for enterprise architecture model analysis, in: Proceedings of the 19th International Conference on Business Information Systems (BIS), Leipzig, Germany, pp. 315–326, 2016.

[S27] Y. Zhang, R. Witte, J. Rilling, and V. Haarslev. Ontology-based program comprehension tool supporting website architectural evolution, in: Proceedings of the 8th International Symposium on Web Site Evolution (WSE), Philadelphia, Pennsylvania, USA, pp. 41–49, 2006.

[S28] A. B. Belle, G. EI Boussaidi, and S. Kpodjedo. Combining lexical and structural information to reconstruct software layers, Information and Software Technology, 74: 1–16, 2016.

[S29] P. van der Spek, S. Klusener, and P. van de Laar. Towards recovering architectural concepts using latent semantic indexing, in: Proceedings of the 12th European Conference on Software Maintenance and Reengineering (CSMR), Athens, Greece, pp. 253–257, 2008.

[S30] G. Scanniello, M. Risi, and G. Tortora. Architecture recovery using latent semantic indexing and k-means: an empirical evaluation, in: Proceedings of the 8th IEEE International Conference on Software Engineering and Formal Methods (SEFM), Pisa, Italy, pp. 103–112, 2010.

[S31] M. Mirakhorli, Y. Shin, J. Cleland-Huang, and M. Cinar. A tactic-centric approach for automating traceability of quality concerns, in: Proceedings of the 34th International Conference on Software Engineering (ICSE), Zurich, Switzerland, pp. 639–649, 2012.

[S32] P. R. Anish, B. Balasubramaniam, J. Cleland-Huang, R. Wieringa, M. Daneva, and S. Ghaisas. Identifying architecturally significant functional requirements, in: Proceedings of the 5th International Workshop on Twin Peaks of Requirements and Architecture (TwinPeaks), Florence, Italy, pp. 3–8, 2015.

[S33] A. Wierda, E. Dortmans, and L. Somers. Using version information in architectural clustering - a case study, in: Proceedings of the 10th European Conference on Software Maintenance and Reengineering (CSMR), Bari, Italy, pp. 214–228, 2006.

[S34] P. Velasco-Elizondo, R. Marín-Piña, S. Vazquez-Reyes, A. Mora-Soto, and J. Mejia. Knowledge representation and information extraction for analyzing architectural patterns, Science of Computer Programming, 121: 176–189, 2016.

[S35] R. Paiva, G. N. Rodrigues, R. Bonifácio, and M. Laderia. Exploring the combination of software visualization and data clustering in the software architecture recovery process, in: Proceedings of the 31st Annual ACM Symposium on Applied Computing (SAC), Pisa, Italy, pp. 1309–1314, 2016.

[S36] R. Weinreich, C. Miesbauer, G. Buchgeher, and T. Kriechbaum. Extracting and facilitating architecture in service-oriented software systems, in: Proceedings of the Joint 10th Working IEEE/IFIP Conference on Software Architecture and 6th European Conference on Software Architecture (WICSA/ECSA), Helsinki, Finland, pp. 81–90, 2012.

[S37] M. A. Javed, S. Stevanetic, and U. Zdun. Cost-effective traceability links for architecture-level software understanding: a controlled experiment, in: Proceedings of the 24th Conference the Australasian Software Engineering (ASWEC), Adelaide, SA, Australia, pp. 69–73, 2015.

[S38] B. Rogers, J. Gung, Y. Qiao, and J. E. Burge. Exploring techniques for rationale extraction from existing documents, in: Proceedings of the 34th International Conference on Software Engineering (ICSE), Zurich, Switzerland, pp. 1313–1316, 2012.

[S39] A. Corazza, S. Di Martino, V. Maggio, and G. Scanniello. Investigate the use of lexical information for software system clustering, in: Proceedings of the 15th European Conference on Software Maintenance and Reengineering (CSMR), Oldenburg, Germany, pp. 35–44, 2011.

[S40] M. Mirakhorli and J. Cleland-Huang. Detecting, tracing, and monitoring architectural tactics in code, IEEE Transactions on Software Engineering, 42(3): 205–220, 2016.

[S41] O. Maqbool and H. A. Babri. Hierarchical clustering for software architecture recovery, IEEE Transactions on Software Engineering, 33(11): 759–780, 2007.

[S42] A. Kuhn, S. Ducasse, and T. Girba. Semantic clustering identifying topics in source code, Information and Software Technology, 49(3): 230–243, 2007.

[S43] M. A. Babar, X. Wang, and I. Gorton. PAKME: a tool for capturing and using architecture design knowledge, in: Proceedings of the 9th International Multitopic Conference (INMIC), Karachi, Pakistan, pp. 1–6, 2005.

[S44] B. S. Mitchell and S. Mancoridis. On the automatic modularization of software system using the bunch tool, IEEE Transactions on Software Engineering, 32(3): 193–208, 2006.

[S45] R. A. Bittencourt and D. D. S. Guerrero. Comparison of graph clustering algorithms for recovering software architecture module views, in: Proceedings of the 13th European Conference on Software Maintenance and Reengineering (CSMR), Kaiserslautern, Germany, pp. 251–524, 2009.

[S46] M. Galster, A. Eberlein, and M. Moussavi. A process module to pre-process requirements for architecting, in: Proceedings of the 14th Annual IEEE International Conference and Workshop on Engineering of Computer Based System (ECBS), Tucson, Arizona, USA, pp. 611–612, 2007.

[S47] G. Scanniello, A. D'Amico, C. D'Amico, and T. D'Amico. Architectural layer recovery for software system understanding and evolution, Software: Practice and Experience, 40(10): 897–916, 2010.

[S48] Y. Wang, P. Liu, H. Guo, X. Chen. Improved hierarchical clustering algorithm for software architecture recovery. In: Proceedings of the 1st International Conference on Intelligent Computing and Cognitive Informatics (ICICCI), Kuala Lumpur, Malaysia, pp. 247–250, 2010.

[S49] M. Soliman, M. Galster, A. R. Salama, and M. Riebisch. Architectural knowledge for technology decisions in developer communities: an exploratory study with stackoverflow, in: Proceedings of the 13th Working IEEE/IFIP Conference on Software Architecture (WICSA), Venice, Italy, pp. 128–133, 2016.

[S50] R. Gopalakrishnan, P. Sharma, M. Mirakhorli, and M. Galster. Can latent topics in source code predict missing architectural tactics? In: Proceedings of the 39th International Conference on Software Engineering (ICSE), Buenos Aires, Argentina, pp. 15–26, 2017.

[S51] T. Wakabayashi, S. Morisaki, N. Atsumi, and S. Yamamoto. An empirical evaluation of detecting omissions by comparing words between requirement and architectural documents. In: Proceedings of the 8th International Workshop on Empirical Software Engineering in Practice (IWESEP), Tokyo, Japan, pp. 12–17, 2017.

[S52] K. Shumaiev and M. Bhat. Automatic uncertainty detection in software architecture documentation, in: Proceedings of the 1st IEEE International Conference on Software Architecture Workshops (ICSAW), pp. 216–219, 2017.

[S53] C. Czepa, H. Tran, U. Zdun, T. T. T. Kim, E. Weiss, and C. Ruhsam. On the understandability of semantic constraints for behavioral software architecture compliance: a controlled experiment. In: Proceedings of the 1st IEEE International Conference on Software Architecture (ICSA), Gothenburg, Sweden, pp. 155–164, 2017.

[S54] X. Li, L. Zhang, and N. Ge. Framework information based java software architecture recovery, in: Proceedings of the 24th Asia-Pacific Software Engineering Conference Workshops (APSECW), Nanjing, China, pp. 114–120, 2017.

[S55] W. Zogaan, I. Mujhid, J. C. S. Santos, D. Gonzalez, and M. Mirakhorli. Automated training-set creation for software architecture traceability problem, Empirical Software Engineering, 22(3): 1028–1062, 2017.

Appendix B. Abbreviations used in the study

AA	Architectural Analysis
AD	Architectural Documentation
AE	Architectural Evaluation
AI	Architectural Implementation
AIA	Architectural Impact Analysis
AK	Architecture Knowledge
AME	Architectural Maintenance and Evolution
AR	Architectural Recovery
ArD	Architecture-related Document
ARu	Architectural Reuse
AS	Architectural Synthesis
ASR	Architecturally Significant Requirement
AU	Architectural Understanding
BLR	Bayesian Logistic Regression
CA	Community Artifact
CE	Concept Extraction
Cf	Classification
Ct	Clustering
DT	Dedicated Tool
EE	Entity Extraction
FR	Functional Requirement
GT	General Tool
HAC	Hierarchical Agglomerative Clustering
IE	Information Extraction
KS	Keyword Search
LA	Link Analysis
LDA	Latent Dirichlet Allocation
LE	Link Extraction
LSA	Latent Semantic Analysis
MST	Minimum Spanning Tree
NER	Named Entity Recognition
NB	Naïve Bayes
NFR	Non-Functional Requirement
NLP	Natural Language Processing
OSS	Open Source Software
POS	Part of Speech
QA	Quality Attribute
RD	Requirements Document
RQ	Research Question
SA	Software Architecture
SCC	Source Code and Comment
SIR	Search Information Retrieval
SLR	Systematic Literature Review
SWS	Semantic Web Search
SMS	Systematic Mapping Study
SVM	Support Vector Machine
VCSD	Virtual Communities for Software Development
WAC	Weighted Combined Algorithm
WM	Web Mining

References

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

Andritsos, P., Tzerpos, V., 2005. Information-theoretic software clustering. IEEE Trans. Softw. Eng. 31 (2), 150–165.

Arksey, H., O'Malley, L., 2005. Scoping studies: towards a methodological framework. Int. J. Soc. Res. Methodol. 8 (1), 19–32.

Babar, M.A., Gorton, I., 2007. A tool for managing software architecture knowledge. In: Proceedings of the 2nd Workshop on Sharing and Reusing Architectural Knowledge - Architecture, Rationale, and Design Intent (SHARK/ADI). Minneapolis, MN, USA. pp. 11–17.

Babar, M.A., Dingsyr, T., Lago, P., van Vliet, H., 2009. Software Architecture Knowledge Management: Theory and Practice. Springer.

Bass, L., Clements, P., Kazman, R., 2012. Software Architecture in Practice, 3rd Edition. Addison-Wesley Professional.

Bengtsson, P.O., Lassing, N., Bosch, J., van Vliet, H., 2004. Architecture-level modifiability analysis (ALMA). J. Syst. Softw. 69 (1-2), 129–147.

Brereton, P., Kitchenham, B.A., Budgen, D., Turner, M., Khalil, M., 2007. Lessons from applying the systematic literature review process within the software engineering domain. J. Syst. Softw. 80 (4), 571–583.

Buckland, M., Gey, F., 1994. The relationship between recall and precision. J. Am. Soc.r Inf. Sci. 45 (1), 12–19.

Cambria, E., Hussain, A., 2015. Sentic Computing: A Common-sense-based Framework for Concept-level Sentiment Analysis. Springer.

Capilla, R., Jansen, A., Tang, A., Avgeriou, P., Babar, M.A., 2016. 10 years of software architecture knowledge management: practice and future. J. Syst. Softw. 116, 191–205.

Chardigny, S., Seriai, A., 2010. Software architecture recovery process based on object-oriented source code and documentation. In: Proceedings of the 4th European Conference on Software architecture (ECSA). Copenhagen, Denmark. pp. 409–416.
Chen, L., Babar, M.A., Zhang, H., 2010. Towards an evidence-based understanding of electronic data source. In: Proceedings of the 14th International Conference on Evaluation and Assessment in Software Engineering (EASE). Keele, UK. pp. 135–138.
Chowdhury, G.G., 2010. Introduction to Modern Information Retrieval.
Chung, L., Nixon, B.A., Yu, E., Mylopoulos, J., 2012. Non-functional Requirements in Software Engineering. Springer Science and Business Media.
Cleland-Huang, J., Gotel, O., Zisman, A., 2012. Software and System Traceability. Springer.
Clements, P., Bachmann, F., Bass, L., Garlan, D., Ivers, J., Little, R., Merson, P., Nord, R., Stafford, J., 2010. Documenting Software Architectures: Views and Beyond, 2nd Edition. Addison-Wesley.
Cooley, R., Mobasher, B., Srivastava, J., 1997. Web mining: information and pattern discovery on the World Wide Web. In: Proceedings of the 9th International Conference on Tools with Artificial Intelligence (ICTAI). Newport Beach, CA, USA. pp. 558–567.
Cowie, J., Lehnert, W., 1996. Information extraction. Commun. ACM 39 (1), 80–91.
Ding, W., Liang, P., Tang, A., van Vliet, H., 2014. How do open source communities document software architecture: an exploratory survey. In: Proceedings of the 19th International Conference on Engineering of Complex Systems (ICECCS). Tianjin, China. pp. 136–145.
Ding, W., Liang, P., Tang, A., van Vliet, H., 2015. Understanding the causes of architecture changes using OSS mailing lists. Int. J. Softw. Eng. Knowl. Eng. 25 (9-10), 1633–1651.
Ducasse, S., Pollet, D., 2009. Software architecture reconstruction: a process-oriented taxonomy. IEEE Trans. Softw. Eng. 35 (4), 573–591.
Egyed, A., Grünbacher, P., Medvidovic, N., 2001. Refinement and evolution issues in bridging requirements and architecture - the CBSP approach. In: Proceedings of the 1st International Workshops From Requirements to Architecture (STRAW). Toronto, Canada. pp. 42–47.
Feldman, R., Sanger, J., 2007. The Text Mining Handbook: Advanced Approaches in Analyzing Unstructured Data. Cambridge University Press.
Fielding, R.T., 2005. Software architecture in an open source world. In: Proceedings of the 27th International Conference on Software Engineering (ICSE). Saint Louis, MO, USA. 43-43.
Grubb, P., Takang, A.A., 2003. Software Maintenance: Concepts and Practice. World Scientific.
Haiduc, S., Bavota, G., Oliveto, R., Marcus, A., De Lucia, A., 2012. Evaluating the specificity of text retrieval queries to support software engineering tasks. In: Proceedings of the 34th International Conference on Software Engineering (ICSE). Zurich, Switzerland. pp. 1273–1276.
Haiduc, S., Bavota, G., Marcus, A., Oliveto, R., De Lucia, A., Menzies, T., 2013. Automatic query reformulations for text retrieval in software engineering. In: Proceedings of the 35th International Conference on Software Engineering (ICSE). San Francisco, CA, USA. pp. 842–851.
Haiduc, S., Arnaoudova, V., Marcus, A., Antoniol, G., 2016. The use of text retrieval and natural language processing in software engineering. In: Proceedings of the 38th International Conference on Software Engineering (ICSE). Austin, TX, USA. pp. 898–899.
Hebig, R., Quang, T.H., Chaudron, M.R.V., Robles, G., Fernandez, M.A., 2016. The quest for open source projects that use UML: mining GitHub. In: Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems (MoDELS). Saint-Malo, France. pp. 173–183.
Henttonen, K., Matinlassi, M., 2009. Open source based tools for sharing and reuse of software architectural knowledge. In: Proceedings of the Joint 13th Working IEEE/IFIP Conference on Software Architecture and the 10th European Conference on Software Architecture (WICSA/ECSA). Cambridge, UK. pp. 41–50.
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.
IEEE. IEEE Std 1517-2010, IEEE Standard for Information Technology - System and Software Life Cycle Processes - Reuse Processes, 2010.
ISO. ISO/IEC/IEEE Std 42010:2011, Systems and Software Engineering - Architecture description, 2011.
Jansen, A., Avgeriou, P., van der Ven, J.S., 2009. Enriching software architecture documentation. J. Syst. Softw. 82 (8), 1232–1248.
Javed, M.A., Zdun, U., 2014. The supportive effect of traceability links in architecture-level software understanding: two controlled experiments. In: Proceedings of the 11th Working IEEE/IFIP Conference on Software Architecture (WICSA). Sydney, Australia. pp. 215–224.
Kao, A., Poteet, S., 2005. Text mining and natural language processing: introduction for the special issue. ACM SIGKDD Explor. Newsl. 7 (1), 29–44.
B. Kitchenham and S. Charters. Guidelines for Performing Systematic Literature Reviews in Software Engineering, Version 2.3. EBSE Technical Report EBSE-2007-01, Keele University and Durham University, 2007.
Kitchenham, B., Brereton, O.P., Budgen, D., Turner, M., Bailey, J., Linkman, S., 2009. Systematic literature reviews in software engineering - a systematic literature review. Inf. Softw. Technol. 51 (1), 7–15.
Knodel, J., Muthig, D., Naab, M., 2008. An experiment on the role of graphical elements in architecture visualization. Empir. Softw. Eng. 13 (6), 693–726.
Koschke, R., 2002. Atomic architectural component recovery for program understanding and evolution. In: Proceedings of the 18th International Conference on Software Maintenance (ICSM). Montreal, Quebec, Canada. pp. 478–481.
Kruchten, P., Obbink, H., Stafford, J., 2006. The past, present, and future for software architecture. IEEE Softw. 23 (2), 22–30.
Lethbridge, T.C., Sim, S.E., Singer, J., 2005. Studying software engineers: data collecting techniques for software field studies. Empir. Softw. Eng. 10 (3), 311–341.
Li, Z., Liang, P., Avgeriou, P., 2013. Application of knowledge-based approached in software architecture: a systematic mapping study. Inf. Softw. Technol. 55 (5), 777–794.
Lung, C.H., 1998. Software architecture recovery and restructuring through clustering techniques. In: Proceedings of the 3rd International Workshop on Software Architecture (ISAW). Orlando, Florida, USA. pp. 101–104.
Maalej, W., Kurtanović, Z., Nabil, H., Stanik, C., 2016. On the automatic classification of app reviews. Requir. Eng. 21 (3), 311–331.
Malavolta, I., Lago, P., Muccini, H., Pelliccione, P., Tang, A., 2013. What industry needs from architectural languages: a survey. IEEE Trans. Softw. Eng. 39 (6), 869–891.
Manevitz, L.M., Yousef, M., 2001. One-class SVMs for document classification. J. Mach. Learn. Res. 2 (12), 139–154.
Medvidovic, N., Jakobac, V., 2006. Using software evolution to focus architectural recovery. Autom. Softw. Eng. 12 (2), 225–256.
Miner, G., Delen, D., Elder, J., Fast, A., Hill, T., Nisbet, R., 2012. Practical Text Mining and Statistical Analysis for Non-structured Text Data Applications. Academic Press.
Mooney, R.J., Bunescu, R., 2005. Mining knowledge from text using information extraction. ACM SIGKDD Explor. Newsl. 7 (1), 3–10.
Moreno, L., Bavota, G., Haiduc, S., Di Penta, M., Oliveto, R., Russo, B., Marcus, A., 2015. Query-based configuration of text retrieval solutions for software engineering tasks. In: Proceedings of the Joint 10th European Software Engineering Conference and the 24th ACM SIGSOFT Symposium on Foundations of Software Engineering (ESEC/FSE). Bergamo, Italy. pp. 567–578.
Pagano, D., Maalej, W., 2013. How do open source communities blog? Empir. Softw. Eng. 18 (6), 1090–1124.
Postma, A., America, P., Wijnstra, J.G., 2004. Component replacement in a long-living architecture: the 3RDBA approach. In: Proceedings of the 4th Working IEEE/IFIP Conference on Software Architecture (WICSA). Oslo, Norway. pp. 89–98.
Quilici, A., 1995. Reverse engineering of legacy systems: a path towards success. In: Proceedings of the 17th International Conference on Software Engineering (ICSE). Seattle, Washington, USA. pp. 333–336.
Raiha, O., Koskimies, K., Makinen, E., 2009. Scenario-based genetic synthesis of software architecture. In: Proceedings of the 4th International Conference on Software Engineering Advances (ICSEA). Porto, Portugal. pp. 437–445.
Shahin, M., Liang, P., Li., Z., 2014. Do architectural design decisions improve the understanding of software architecture? Two controlled experiments. In: Proceedings of the 22nd International Conference on Program Comprehension (ICPC). Hyderabad, India. pp. 3–13.
Sheng, V.S., Provost, F., Ipeirotis, P.G., 2008. Get another label? Improving data quality and data mining using multiple, noisy labelers. In: Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD). Las Vegas, Nevada, USA. pp. 614–622.
Spanoudakis, G., Zisman, A., 2005. Software traceability: a roadmap. Handbook of Software Engineering and Knowledge Engineering 3. pp. 395–428 Chapter 14.
Stevanetic, S., Zdun, U., 2014. Exploring the relationships between the understandability of components in architectural component models and component level metrics. In: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering (EASE). London, UK. pp. 1–10.
Tang, A., Lau, M.F., 2014. Software architecture review by association. J. Syst. Softw. 88 (1), 87–101.
Tang, A., Babar, M.A., Gorton, I., Han, J., 2006. A survey of architecture design rationale. J. Syst. Softw. 79 (12), 1792–1804.
Tang, A., Avgeriou, P., Jansen, A., Capilla, R., Babar, M.A., 2010. A comparative study of architecture knowledge management tools. J. Syst. Softw. 83 (3), 352–370.
Taylor, R.N., Medvidovic, N., Dashofy, E.M., 2010. Software Architecture: Foundations, Theory, and Practice. Wiley.
Tekinerdoğan, B., Akşit, M., 2002. Synthesis-based Software Architecture Design, Software Architectures and Component Technology. Springer, pp. 143–173 Chapter 5.
Vico, H., Calegari, D., 2015. Software architecture for document anonymization. Electron. Notes Theor. Comput. Sci. 314 (1), 83–100.
Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A., 2012. Experimentation in Software Engineering. Springer.
Wohlin, C., 2014. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering (EASE). London, England, United Kingdom. pp. 23–31.
Xiao, L., Cai, Y., Kazman, R., 2014. Design rule spaces: a new form of architecture insight. In: Proceedings of the 36th International Conference on Software Engineering (ICSE). Hyderabad, India. pp. 967–977.
Xu, W., Liu, X., Gong, Y., 2003. Document clustering based on non-negative matrix factorization. In: Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR). Toronto, Canada. pp. 267–273.
Xu, J., Gao, Y., Christley, S., Madey, G., 2005. A topological analysis of the open source software development community. In: Proceedings of the 38th International Conference on System Sciences (HICSS). Big Island, HI, USA. pp. 198–208.
Yi, J., Nasukawa, T., Bunescu, R., Niblack, W., 2003. Sentiment analyzer: extracting sentiments about a given topic using natural language processing techniques. In: Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM). Melbourne, Florida, USA. pp. 427–434.
Zapalowski, V., Nunes, I., Nunes, D.J., 2014. Revealing the relationship between architectural elements and source code characteristics. In: Proceedings of the 22nd

International Conference on Program Comprehension (ICPC). Hyderabad, India. pp. 14–25.
Zhang, W.E., Sheng, Q.Z., Abebe, E., Babar, M.A., Zhou, A., 2016. Mining source code topics through topic model and words embedding. In: Proceedings of the 12th International Conference on Advanced Data Mining and Applications (ADMA). Gold Coast, QLD, Australia. pp. 664–676.
Zhao, J., 2001. Using dependence analysis to support software architecture understanding. Comput. Res. Repos. 1 (9), 135–142.
Zhou, X., Jin, Y., Zhang, H., Li, S., Huang, X., 2016. A map of threats to validity of systematic literature reviews in software engineering. In: Proceedings of the 23rd Asia-Pacific Software Engineering Conference (APSEC). Hamilton, New Zealand. pp. 153–160.

Tingting Bi is a Ph.D. candidate in the State Key Lab of Software Engineering (SKLSE), School of Computer Science, Wuhan University, China and a visiting Ph.D. candidate at the Faculty of Science, Engineering and Technology, Swinburne University of Technology, Australia. She received her bachelor and master degrees in computer science and technology from Harbin University of Science and Technology, China. Her current research interests include software architecture, natural language processing, and machine learning. She has published several articles in peer-reviewed international journals.

Peng Liang is Professor of Software Engineering in the State Key Lab of Software Engineering (SKLSE), School of Computer Science, Wuhan University, China. His research interests concern the areas of software architecture and requirements engineering. He is a Young Associate Editor of Frontiers of Computer Science, Springer. He has published more than 90 articles in peer-reviewed international journals, conference and workshop proceedings, and books.

Antony Tang is Associate Professor in Swinburne University of Technology, Australia. He received a PhD degree in Information Technology from Swinburne in 2007. Prior to being a researcher, he had spent many years designing and developing software systems. His main research interests are software architecture design reasoning, software development processes, software architecture knowledge management.

Chen Yang is currently a general manager at the Technology Center of IBO Technology Company Limited (located in Shenzhen, China). He focuses on all the scientific and technological issues of the company. He got his Ph.D. (double degree) in Computer Science from University of Groningen and Software Engineering from Wuhan University. Before becoming a Ph.D. student, he had worked as a senior software engineer and architect in industry for four years.