Information and Software Technology 56 (2014) 545–567

Contents lists available at ScienceDirect

Information and Software Technology

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

Knowledge-based approaches in software documentation: A systematic literature review

Wei Ding [a,d], Peng Liang [a,c,*], Antony Tang [b], Hans van Vliet [c]

[a] State Key Lab of Software Engineering, School of Computer, Wuhan University, China
[b] Faculty of Information and Communication Technologies, Swinburne University of Technology, Australia
[c] Department of Computer Science, VU University Amsterdam, Netherlands
[d] Key Laboratory of Earthquake Geodesy, Institute of Seismology, China Earthquake Administration, China

ARTICLE INFO

Article history:
Received 12 August 2013
Received in revised form 9 January 2014
Accepted 18 January 2014
Available online 25 January 2014

Keywords:
Knowledge-based approach
Software documentation
Systematic literature review
Knowledge activity
Software architecture design

ABSTRACT

Context: Software documents are core artifacts produced and consumed in documentation activity in the software lifecycle. Meanwhile, knowledge-based approaches have been extensively used in software development for decades, however, the software engineering community lacks a comprehensive understanding on how knowledge-based approaches are used in software documentation, especially documentation of software architecture design.
Objective: The objective of this work is to explore how knowledge-based approaches are employed in software documentation, their influences to the quality of software documentation, and the costs and benefits of using these approaches.
Method: We use a systematic literature review method to identify the primary studies on knowledge-based approaches in software documentation, following a pre-defined review protocol.
Results: Sixty studies are finally selected, in which twelve quality attributes of software documents, four cost categories, and nine benefit categories of using knowledge-based approaches in software documentation are identified. Architecture understanding is the top benefit of using knowledge-based approaches in software documentation. The cost of retrieving information from documents is the major concern when using knowledge-based approaches in software documentation.
Conclusions: The findings of this review suggest several future research directions that are critical and promising but underexplored in current research and practice: (1) there is a need to use knowledge-based approaches to improve the quality attributes of software documents that receive less attention, especially *credibility*, *conciseness*, and *unambiguity*; (2) using knowledge-based approaches with the knowledge content in software documents which gets less attention in current applications of knowledge-based approaches in software documentation, to further improve the practice of software documentation activity; (3) putting more focus on the application of software documents using the knowledge-based approaches (knowledge reuse, retrieval, reasoning, and sharing) in order to make the most use of software documents; and (4) evaluating the costs and benefits of using knowledge-based approaches in software documentation qualitatively and quantitatively.

Contents

1. Introduction . . . 546
2. Research method . . . 547
 2.1. Context and research questions . . . 547
  2.1.1. Knowledge-based approach . . . 547
  2.1.2. Software documentation . . . 548
  2.1.3. Research questions . . . 549
 2.2. Inclusion and exclusion criteria . . . 549

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

http://dx.doi.org/10.1016/j.infsof.2014.01.008
0950-5849/

2.3. Search process . . . 549
2.3.1. Search scope . . . 551
2.3.1.1. Time period . . . 551
2.3.1.2. Electronic databases . . . 551
2.3.1.3. Journals, conferences, and workshops . . . 551
2.3.2. Search terms . . . 551
2.3.3. Search strategy . . . 552
2.4. Data extraction and synthesis . . . 552
3. Results . . . 553
3.1. Overview of results . . . 553
3.2. RQ1: Quality attributes of software documents and knowledge-based approaches . . . 554
3.2.1. Quality attributes of software documents . . . 554
3.2.2. How knowledge-based approaches improve quality attributes of software documents . . . 555
3.2.3. Quality attributes of software documents and their concerned elements in software documents . . . 555
3.3. RQ2: Knowledge-based approaches in software documentation . . . 556
3.3.1. Distribution of knowledge-based approaches in software documentation . . . 556
3.3.2. General and specific knowledge-based approaches . . . 557
3.3.3. Knowledge-based approaches and documented content . . . 558
3.4. RQ3: Costs and benefits of using knowledge-based approaches . . . 558
3.4.1. Costs of using knowledge-based approaches . . . 558
3.4.2. Benefits of using knowledge-based approaches . . . 559
3.5. Evidential support . . . 560
4. Discussion . . . 561
4.1. Scope of the systematic review . . . 561
4.2. Study quality assessment . . . 561
4.3. Validity threats . . . 561
4.4. Further research . . . 562
5. Conclusions . . . 563
Acknowledgements . . . 564
Appendix A. Primary studies in the review . . . 564
Appendix B. Abbreviations used in the review . . . 566
References . . . 566

1. Introduction

Software is defined as the "*intellectual creation comprising the programs, procedures, rules, and any associated documentation pertaining to the operation of a data processing system*" [6]. Software documentation (SDt[1])[2] is a formal writing in both print or electronic form that supports the efficient and effective use of software in its intended environment [12]. SDt is regarded as an integral part of the software development process [54] and has a number of uses in software lifecycle (e.g., as a communication medium for stakeholders, information repository for maintainer, and guide for software users) [62]. The essence of software development process is the coordination and communication of "individuals" towards achieving common and explicitly recognized goals in order to produce working software [38]. Software documents (SD[3]) provide an asynchronous way for the communication among stakeholders, which overcomes the time and geographical restrictions during software development process. Improving the quality of SDt will improve the quality of software accordingly [55].

Software development is a knowledge-intensive activity [60,72]. In knowledge management (KM) theory, knowledge can be classified as "tacit knowledge" or "explicit knowledge" [53]. Explicit knowledge is the knowledge codified in certain form (e.g., a document or a model). Tacit knowledge resides in people's head and is not easily visible and expressible. Explicit knowledge is easy to use and reuse, while tacit knowledge tends to vaporize over time, especially when the people who possess the knowledge leave. From a KM perspective, SDt provides a way to transform tacit knowledge into explicit knowledge, and exchanges the knowledge among individuals and organizations [60].

There are many types of SDt, such as requirements documentation, design documentation and test documentation which are used extensively across the lifecycle of software development. To achieve a better focus and meaningful results in this review, we stress the SD for architecture design, covering requirements and architecture documents. The software requirements document (SRD) describes externally-observable behaviors and characteristics expected of a software system [22]. The software architecture document (SAD) records architecture design and related architecture information in e.g., architecture views [20]. SRD and SAD are the software artifacts produced in the requirements engineering and architecting phase respectively, two closely-related phases in the software lifecycle. These two types of documents are typically written in natural language with supporting diagrams (e.g., UML use case diagrams and component diagrams) for the communication between various stakeholders of the project, such as customers, managers, requirements engineers, architects, and developers. Various formats of SDt are used in practice, most of them are in files (e.g., MS Word document), as well as in emails, text massages, blogs, and wikis [71]. All these formats of SDt that contain requirements or architectural information are regarded as SRD or SAD. The quality of SRD contributes to the successful and cost-effective creation of software [22], and improving the quality of SAD can facilitate various architecting activities, such as architecture review [36]. To this end, the quality aspect of SDt is a research focus in this review.

[1] SDt (software documentation) and SD (software document), these two terms and abbreviations are both used in this paper with different meanings. SD denotes the document artifact produced by software documentation activity, and SDt represents both the document artifact and documentation activity.

[2] For readability and clarity, we list all the abbreviations used in this paper in Appendix B for reference.

[3] SD is singular as well as plural based on the context in which it is used. This rule is also applied to abbreviations SRD (software requirements documents), SAD (software architecture documents), and QA (quality attributes).

A knowledge-based approach in this context is one that explicitly facilitates the development, evolution, and use of knowledge that is critical to successful software development and evolution [34]. Knowledge-based approaches can facilitate the understanding and management of documentation [26,39,46]. A rigorous assessment and review on knowledge-based approaches in SDt is meaningful and necessary to increase their acceptance and guide their application in SDt practices.

We decided to use systematic literature review (SLR) method in this study since we aimed at identifying, evaluating, interpreting, and synthesizing all available studies to answer particular research questions, and establishing the state of evidence, with in-depth analysis [37,57]. The objective of this SLR is to understand what knowledge-based approaches can be employed to improve the quality of SDt. We also investigate what quality attributes matter to SDt, and the costs and benefits of using knowledge-based approaches in SDt. This work has a special focus on software requirements and architecture documentation [56].

To the best of our knowledge, there is currently no survey or systematic literature review specifically conducted on knowledge-based approaches to SDt. However, there are a number of surveys and secondary studies (e.g., systematic mapping studies) on the sub-areas of SDt using specific knowledge (e.g., rationale knowledge in architecture description) and different aspects of KM for software development. The studies on using knowledge or KM in software engineering, information systems, and management disciplines are too broad, and are out of the scope of the related work.

(1) *Knowledge in architecture design*: Nakagawa et al. used a systematic mapping to explore, understand, organize, and summarize the research and practices of software architectural knowledge [51]. Their study focuses on the architectural knowledge for constructing reference architectures, while our review pays attention to using knowledge for architecture documentation. Tang et al. employed a survey research method to get an understanding of architects' perceptions on the documentation and use of architectural design rationale [66], which is an important part of architectural knowledge. In another work, Tang et al. surveyed five architectural KM tools and made a comparison on their support for architectural knowledge management and their satisfaction of the criteria from the architecture description standard [65]. Their work used a comparison method to analyze how and to what extent an architectural KM tool can support architecture descriptions. Shahin et al. analyzed existing architectural design decision models to identify their consensus and differences at the model level [61]. Their work focuses on architectural design decisions themselves and relevant supporting tools without considering the impact to the architecting process.

(2) *Knowledge in requirements specification*: Barmi et al. conducted a systematic mapping study on the alignment of requirements specification and testing [13]. Nicolás and Toval systematically reviewed the literature related to the use of software engineering (SE) models to generate requirements specification [52]. These two secondary studies focus on specific knowledge (e.g., models and alignment knowledge) in requirements specification, but have little discussion about the relationship between general knowledge and their application in requirements specification.

(3) *Knowledge engineering and management techniques for SE*: Briand surveyed the knowledge engineering techniques that SE problems can benefit from [16]. His study has little discussion of SD and, as a position paper, it only introduced the candidate knowledge engineering techniques without further investigation. Dingsøyr and Conradi conducted a survey on the literature that reports case studies of using KM approaches in SE whose results have a high level of evidence [27]. The definition of KM approaches in their survey is similar to the knowledge-based approaches used in this review, but the scope of their survey is much broader than the scope of SDt and has a special focus on case studies research.

(4) *Software documentation*: Biehl made a survey on how researchers and practitioners document the "why" in architecture descriptions in the form of design rationale, design decisions, and architectural knowledge [14]. His survey stresses software architecture descriptions, but our review focuses on both software requirements and architecture documents. Forward and Lethbridge presented a practical survey that evaluated tools and technologies for SDt [31]. Their survey did not cover the relevance of knowledge to SDt, and related tools and technologies.

This paper presents the results of a SLR on knowledge-based approaches in SDt published from January 2001 to September 2011. The main objectives of this review are the following: (1) systematically select and review literature, and present a holistic overview of existing studies on knowledge-based approaches in SDt, with a special focus on documentation for architecture design (i.e., requirements and architecture documents); (2) comprehensively understand how knowledge-based approaches are employed in SDt, their influences, and the costs and benefits of using knowledge-based approaches in SDt; (3) identify research challenges and gaps that require further exploration and investigation in this area, and provide evidence-based recommendation to the future research directions on this topic.

The remainder of this paper is organized as follows. In Section 2, we describe the SLR research method and the review process. The results of this SLR are presented in Section 3. Section 4 discusses the scope of this review, the quality assessment of selected studies, and the threats to the validity of the review results. The conclusions and future directions are outlined in Section 5.

2. Research method

2.1. Context and research questions

2.1.1. Knowledge-based approach

Knowledge is information possessed in the mind of individuals, and it is personalized information related to facts, procedures, concepts, interpretations, ideas, observations, and judgments [8]. In KM theory, KM is largely regarded as a process involving various knowledge activities, including creating, storing/retrieving, transferring, and applying knowledge [8]. In this review, we define a knowledge-based approach as any approach which can be applied in KM and facilitates the knowledge activities in KM. The classification of knowledge-based approaches employed in this SLR can be readily mapped to these basic knowledge activities discussed below.

According to the definition and analysis of knowledge-based approaches, we propose a knowledge management process in SDt, which is shown in Fig. 1. The general KM process proposed in [8] is shown at the top of Fig. 1. The arrows in this general KM process denote the sequence of knowledge activities that constitutes a KM process. The KM process in SDt shown in the lower part of Fig. 1 can be regarded as a specific KM process, one in which knowledge-based approaches of producing and consuming software document knowledge are mapped to knowledge activities in the general KM process.

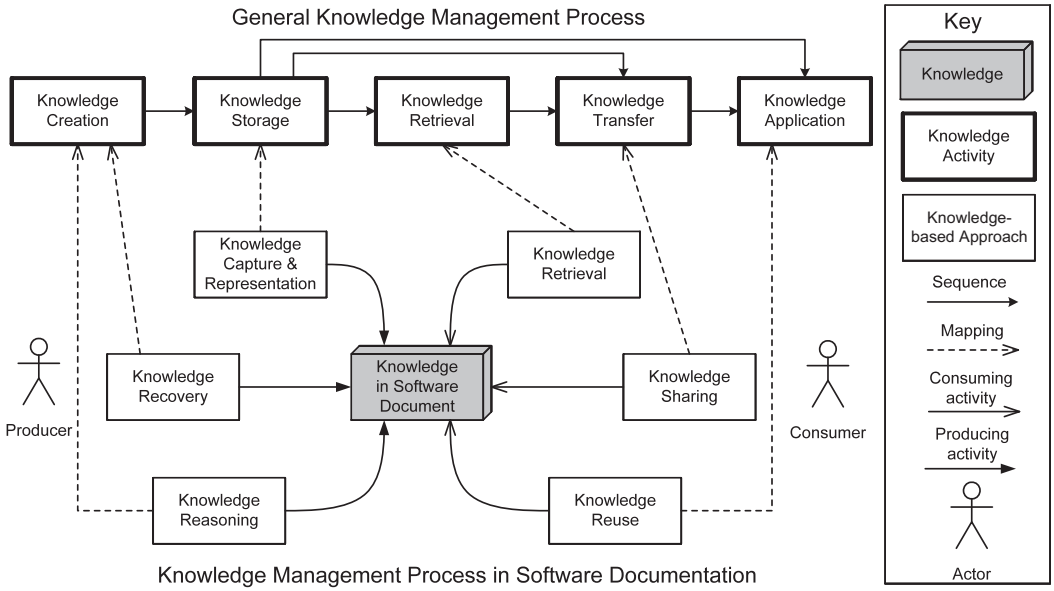

Fig. 1. Using knowledge management process in software documentation.

The mappings with dashed arrows from knowledge-based approaches to general knowledge activities are shown in Fig. 1 from the perspective of the use (production and consumption) of knowledge in SD: (1) Knowledge can be captured from SD. Meanwhile, captured knowledge can be represented in a certain form (e.g., natural language or formal models). These activities belong to knowledge capture and knowledge representation (KCR) which can be mapped to the knowledge storing activity. (2) The activity of retrieving knowledge from SD belongs to knowledge retrieval (KRt) (e.g., captured knowledge can be returned in a structured form), which can be directly mapped to the knowledge retrieval activity. (3) Knowledge in SD can be shared with other individuals and organizations. The approach is called knowledge sharing (KS), and it can be mapped to the knowledge transfer activity. (4) Knowledge in a SD can be (re)used in another SD, e.g., software requirements reuse in a product family or architecture patterns to address similar design issues, which are classified as knowledge reuse (KR), and it can be mapped to the knowledge application activity. (5) Implicit knowledge in SD, such as implicit dependencies between parts of a document, can be recovered to become explicit knowledge, which belongs to knowledge recovery (KRv). Since this approach creates new (explicit) knowledge, it can be mapped to the knowledge creation activity. (6) New knowledge can also be created from existing knowledge in SD through knowledge reasoning (KRs), and this approach can be mapped to the knowledge creation activity. For example, reasoning on architecture design (i.e., existing architectural knowledge) can be used to detect design conflicts in architecture (i.e., new knowledge) [69].

Based on Alavi and Leidner's review on KM and KM systems [8], the classification of knowledge-based approaches in [42], and the mapping from the KM process in SDt to general knowledge activities discussed above, the knowledge-based approaches to SDt can be classified into knowledge capture and representation, retrieval, reuse, sharing, recovery, and reasoning, which are further elaborated below:

- *Knowledge capture and representation (KCR)* aims to extract knowledge from different types of SD, and represents knowledge in certain forms so that the captured knowledge can be used by other knowledge-based approaches. The reason we combine these two approaches (i.e., knowledge capture and knowledge representation) as an integrated approach is twofold. Firstly, knowledge capture is the prerequisite of knowledge representation. i.e., only captured knowledge from SD is represented. Secondly, knowledge representation is integrated with knowledge capture, in the form of natural language or formal models to represent (describe/specify) explicit knowledge. KCR is widely used to improve SD quality (e.g., KCR is used to exploit and integrate existing information and collect new knowledge to support architectural assessments [49]).
- *Knowledge retrieval (KRt)* seeks to return knowledge in a structured form, such that the knowledge can be used in a meaningful way to support software development [68]. For example, a SAD can be annotated and stored to a knowledge base, so that concerned stakeholders can retrieve architectural knowledge more efficiently for various purposes [25].
- *Knowledge reuse (KR)* reuses captured document knowledge in software processes (e.g., security requirements knowledge in requirements specifications can be reused in large projects [45]).
- *Knowledge sharing (KS)* exchanges knowledge (e.g., requirements, frameworks, or design decisions) in SD among stakeholders of software projects.
- *Knowledge recovery (KRv)* recovers knowledge that is not explicit in existing SD (e.g., architectural knowledge acquired from an existing SAD is used to understand a new SAD).
- *Knowledge reasoning (KRs)* draws conclusions and gets new knowledge from existing knowledge in SD (e.g., knowledge of design decision in a SAD is visualized to reason about new knowledge [40]).

2.1.2. Software documentation

Poor software documentation is the cause of many errors and reduces efficiency in software development and use. How to produce and use SDt has been a critical issue in software development [55]. As mentioned earlier, the scope of this SLR is SDt for architecture design, concerning SRD and SAD. The QA of SD define the evaluation of the quality of SDt [22]. In order to find which QA of SD are affected by using knowledge-based approaches, we survey and collect the QA of SD from existing literature and standards on SRD and SAD [1,2,22,36], and derive an initial set of QA used in this review, including *completeness*, *understandability*, *traceability*,

clarity, *consistency*, *evolvability*, *conciseness*, and *reusability*. This initial set of QA of SD is refined continuously when new QA are identified and added to the set during the process of this SLR, which is detailed in the answer to RQ2 in Section 3.2.

2.1.3. Research questions

We conduct this SLR by following the guidelines for performing SLR in SE proposed by Kitchenham and Charters [37]. To understand existing research and practices on knowledge-based approaches in SDt, the following research questions (RQs) are formulated with their rationale, following the recommendations in [37] about defining RQs of a SLR:

RQ1: What software document (SD) quality attributes (QA) are influenced by knowledge-based approaches?

Rationale: The quality of SDt has indirect impact on the quality of software [55]. Improvements on the quality of SDt can be reflected and evaluated through the improvements of specific QA of SD. We first classify QA of SD collected during the search process (see Section 2.3), and further identify the QA that are affected by using knowledge-based approaches, and how these approaches improve the QA of SD. Answering this RQ can help identify various aspects of improving the overall quality of SDt with different knowledge-based approaches.

RQ2: What knowledge-based approaches are employed in software documentation (SDt)?

Rationale: Answers to this RQ tell us what the current knowledge-based approaches are. We analyze these knowledge-based approaches to identify the gaps for further investigation, for example the knowledge-based approaches in SDt that received less attention in selected studies.

RQ3: What are the costs and benefits of using knowledge-based approaches in software documentation (SDt)?

Rationale: Applying knowledge-based approaches in SDt has certain benefits but not without costs. Answering this RQ can help to understand their trade-off.

The answers to these three RQs can be directly linked to the objective of this SLR: an understanding of how knowledge-based approaches are employed in SDt (RQ2), their influences (RQ1), and their costs and benefits (RQ3).

2.2. Inclusion and exclusion criteria

We defined the following inclusion and exclusion criteria to select studies from the search results based on the SLR guidelines [37]:

Inclusion criteria:

I1: The theme of the study is documentation for software architecture design, including requirements and architecture documents.
I2: The study presents one or more knowledge-based approaches to address problems in SDt or use SDt to support other software development activities.

Exclusion criteria:

E1: If two papers publish the same study, the less mature one is excluded.
E2: Any paper whose full text is not accessible is excluded.
E3: If a paper introduces an approach to address problems in SDt activity or use SD to support other software development activities, but this approach is not knowledge-based as classified in Section 2.1.1, the paper is excluded.
E4: A paper without evidential support (i.e., "no evidence" as classified in [10]) is excluded.
E5: A paper, that introduces a method for specifying and verifying requirements or architecture using formal representations, is excluded. We introduce this exclusion criterion because this SLR focuses on how human-understandable knowledge can be used in SDt. Techniques for specifying and verifying requirements or architecture using formal representation focus on machine-processible knowledge (including formal requirements specification and architecture description languages), which is intentionally left out of the scope of this SLR to achieve meaningful and focused review results.

2.3. Search process

We design a SLR protocol to guide the search process based on the SLR guidelines [37]. Relevant papers are retrieved automatically from the databases (i.e., through a database search), as well as manually from target journals, conferences, and workshops as a supplementary source to the database search. The study selection process in both databases and target venues consist of the following three phases:

Phase 1: The first author applies the search strategy, which is elaborated in Section 2.3.3, to identify potential primary studies. Two of the authors check the titles of all potential primary studies against inclusion and exclusion criteria. If it is difficult to decide whether one paper should be included or not by title, this paper will be included for the next phase of paper selection.
Phase 2: Two of the authors check the abstracts of the selected papers of Phase 1 against inclusion and exclusion criteria. Disagreements about paper selection results are discussed and resolved by all the participants of this review. If a disagreement about a study cannot be resolved (e.g., we cannot decide whether an approach proposed in the study is knowledge-based or not), the study will be included for the next phase of selection.
Phase 3: Two of the authors read the full text of the papers selected after Phase 2 and use the inclusion and exclusion criteria to decide whether the papers will be finally included or not. Reference search is performed in this phase to check whether the references of selected papers in Phase 3 should be included or not. Reference search is a supplementary search in addition to database search and manual search. The additional studies obtained from the references search in Phase 3 will undergo the selection process in Phases 2 and 3.

The finally-selected studies are a combination of the selected studies from database, manual, and reference searches.

Fig. 2 illustrates the search process and the number of papers included at each stage. Duplicate papers are removed in Phase 1 of the database search when results are retrieved from different databases and in Phase 3 when the search results of database search and manual search are merged. In Phase 1, the search retrieved a total of 31,840 papers in databases and 12,725 papers in target journals, conferences, and workshops. They are captured in the reference management tool EndNote. This tool is also used in the subsequent steps for storing and sorting retrieved papers. After reading and checking the titles of these papers, 410 papers are included for further selection in Phase 2. As we can see in Fig. 2, a huge number of papers are excluded in Phase 1, and the number of included papers is decreased from 44,565 to 410. We

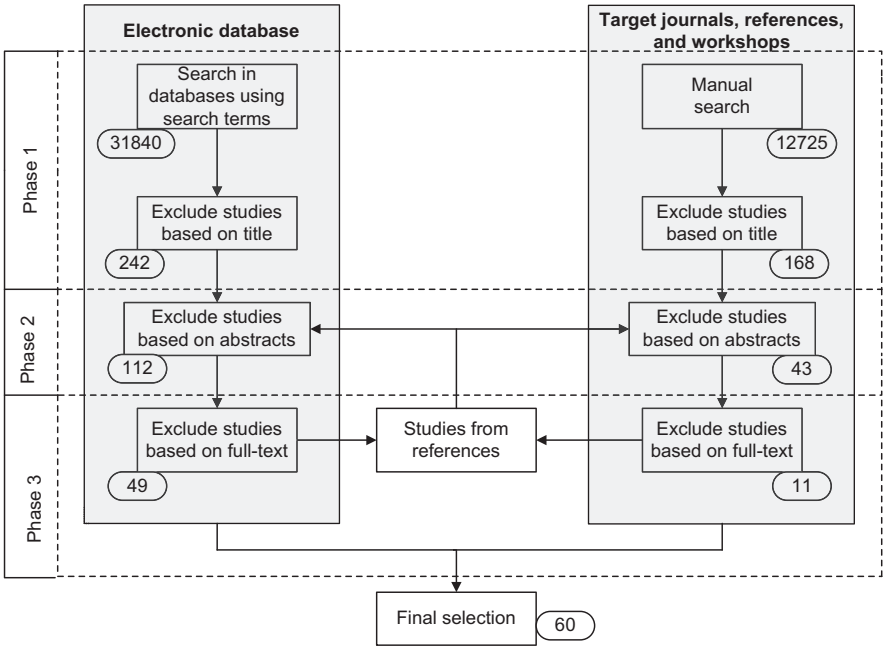

Fig. 2. Study search and selection results in three phases of paper selection.

Table 1
Electronic databases included in this SLR.

#	Electronic databases
DB1	IEEE Xplore
DB2	ACM Digital library
DB3	ScienceDirect
DB4	EI Compendex
DB5	ISI Web of Science
DB6	SpringerLink
DB7	Wiley InterScience
DB8	EBSCO
DB9	Google Scholar

found that most of the excluded studies in Phase 1 are due to the interference of general search terms, such as *description*, *decision*, and *reusable*. In Phase 2, we include and exclude studies by reading further their abstracts, and 155 studies are retained for full text screening in Phase 3 to ensure that their major contribution is indeed related to the topic of this SLR. To make the search process more comprehensive, we also iteratively scan (i.e., snowball) the references of selected studies got in Phase 3. The SLR guidelines proposed by Kitchenham and Charters suggest that the creditability of a study is based on the type of experiment [37]. In order to make this SLR credible, the studies without any validation were intentionally excluded (i.e., exclusion criteria E4) according to the

Table 2
Journals, conferences, and workshops included in this SLR.

#	Journal
J1	IEEE Transactions on Software Engineering (TSE)
J2	Empirical Software Engineering (ESE)
J3	IEEE Software (IEEE SW)
J4	International Journal of Software Engineering and Knowledge Engineering (IJSEKE)
J5	Journal of Systems and Software (JSS)
J6	Information and Software Technology (IST)
J7	Software Process Improvement and Practice (SPIT)
J8	Software and System Modeling (SoSyM)
J9	Software Quality Journal (SQJ)
J10	Automated Software Engineering (ASE)
J11	Software: Practice and Experience (SPE)
	Conference
C1	International Conference on Software Engineering and Knowledge Engineering (SEKE)
C2	International Conference on Software Engineering (ICSE)
C3	Working IEEE/IFIP Conference on Software Architecture (WICSA)
C4	European Conference on Software Architecture (ECSA)
C5	International Conference on the Quality of Software Architectures (QoSA)
C6	IEEE/ACM International Conference on Automated Software Engineering (ASE)
C7	International Requirements Engineering Conference (RE)
C8	International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ)
C9	ACM Symposium on Document Engineering (DocEng)
	Workshop
W1	Workshop on SHAring and Reusing architectural Knowledge (SHARK)
W2	International Workshop on Managing Requirements Knowledge (MaRK)
W3	International Workshop on Empirical Requirements Engineering (EmpiRE)

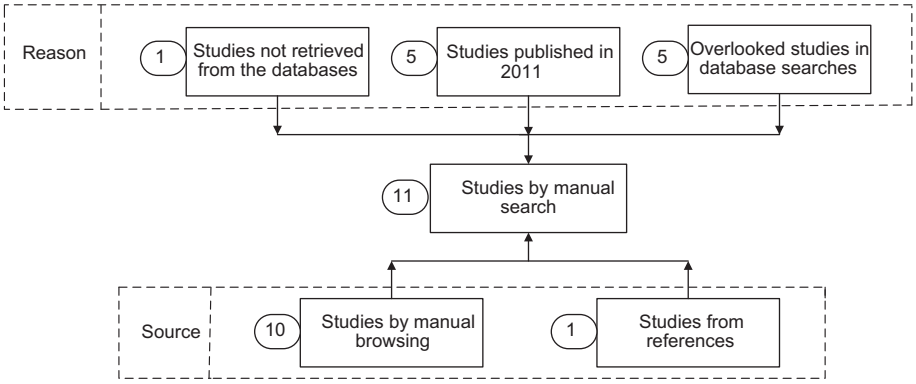

Fig. 3. Sources and reasons of the selected studies retrieved by manual search.

evidence levels described in Section 2.4. In the end, 60 papers are finally selected to be further analyzed in this SLR.

Note that, of the 11 papers finally obtained through manual search, 10 are retrieved by manually browsing the target journals, conferences, and workshops listed in Table 2, and the remaining paper is retrieved by manually browsing the references of the selected papers.

One of the papers obtained through manual search [S34] cannot be retrieved from the databases (see Table 1). Five of the papers [S30, S40, S44, S50, S59] are published in 2011, which may have not been indexed in the databases when we started this SLR. The remaining five papers [S5, S8, S25, S32, S42] of the manual search results can be retrieved by database search, but were overlooked in the selection process. We suppose that this is partly because these 5 papers can only be retrieved in IEEE or ACM databases, which returned a large number of search results in Phase 1, and consequently negatively affected participants' focus and judgement in paper selection. The sources and reasons of the selected studies got by manual search are illustrated in Fig. 3.

2.3.1. Search scope

2.3.1.1. Time period. We specify the time period of published studies for this SLR from January 2001 to September 2011, when we started this SLR. As mentioned in the Introduction section, there is currently no survey or SLR on knowledge-based approaches in SDt. Forward and Lethbridge made a general survey on SDt, and related tools and techniques, covering the studies published before 2001 [31]. In order to reduce repetitive effort and make use of existing work, we set the starting time of the published studies included in this SLR to January 2001.

2.3.1.2. Electronic databases. According to the suggestion in [18], the following databases are selected as the primary study sources (Table 1). INSPEC database has been merged into EI Compendex database, consequently INSPEC is excluded in the search process.

2.3.1.3. Journals, conferences, and workshops. We apply two criteria for selecting journals, conferences, and workshops (Table 2) as the target venues for manual search: (1) they should be highly relevant to cover research areas of SDt or both SDt and KM; (2) they are the leading journals, conferences, and workshops in the review areas, including SE, intersection of SE and KM, requirements engineering, and software architecture. Note that, the selection of journals, conferences, and workshops for the manual search may not be comprehensive since we regarded the manual search as a supplementary, but not exhaustive, source to the database (automatic) search. Considering the two criteria, C3, C4, C5, and W1 are top conferences or workshops on software architecture, which are relevant to SAD. C7, C8, W2, and W3 are included because they are conferences and workshops on requirements engineering and requirements knowledge, which have a close relationship to SRD. Furthermore, publications on SE venues or both SE and KM venues are potentially relevant to SDt. These venues include journals J1, J2, J3, J4, J5, J6, J7, J8, J10, J11, and conferences C1 and C2. J9 is included because the publications in this journal focus on software quality, which may contain the studies on improving the quality of SDt.

2.3.2. Search terms

We use population, intervention, comparison, and outcome (PICO) criteria to define the search terms for database search in this SLR based on the SLR guidelines [37].

Population: The population in this SLR is "SDt". We use the words that are relevant to SDt as the population (e.g., documentation, specification, and description). In order to cover as many studies as possible, the word "software" is not included in the population.

Intervention: The intervention is "knowledge-based approaches". We use the word "knowledge" and its synonyms for the intervention (e.g., knowledge, semantic, rationale, and decision).

Comparison: Since there is no compared approach for this review according to the SLR guidelines [37], the part of comparison specified in PICO is not considered in the construction of search terms.

Outcome: The outcome we focus on in this SLR by applying knowledge-based approaches is the "QA of SD".[4] As discussed in Section 2.1.2, we survey and collect the QA of SD from existing literature and standards on SRD and SAD [1,2,22,36], and derive the initial set of the outcomes, which is composed of the following QA: *completeness*, *understandability*, *traceability*, *clarity*, *consistency*, *evolvability*, *conciseness*, and *reusability*, Newly identified QA of SD and their synonyms are added to the outcome during the search process (e.g., *comprehensibility*, *unambiguity*, *retrievability*, *modifiability*, *correctness*, *credibility*, and *maturity*). We decide that a study addresses a certain QA of SD if the QA term is explicitly mentioned in the paper and the term is related to SD. Meanwhile, some synonyms of QA are excluded in the search process because these terms cause a high number of search results (e.g., *consist*, *correct*, and *clear*). Note that, the QA *maturity* can be divided into three sub-QA: *completeness*, *correctness*, and *consistency* as suggested in [36]. With this consideration, *maturity* is also included as a search term in the outcome in order to retrieve relevant papers more comprehensively.

[4] More precisely, improved QA of SD mean the improved "value" of QA of SD. For conciseness, we use the phrase "improved QA of SD" in the rest of the paper.

Table 3
List of search terms in population, intervention, and outcome.

	Search terms
Population (P)	Documentation, document, documenting, specification, specify, specifying, description
Intervention (I)	Knowledge, semantic, rationale, decision
Outcome (O)	Completeness, understandability, understandable, comprehensibility, comprehensible, traceability, traceable, clarity, consistency, modifiability, evolvability, evolutionary, conciseness, concise, reusability, unambiguity, retrievability, correctness, credibility, maturity, communicability

The final list of the search terms in population (P), intervention (I), and outcome (O) are presented in Table 3.

2.3.3. Search strategy

The search strategy, describing how to combine the search terms, is used in this SLR to obtain a fair and comprehensive literature review. The search strategy that supports the search process in three phases is described below:

(1) An initial set of search terms is proposed according to the description of search term identification in PICO specified in Section 2.3.2.
(2) Various combinations of search terms are used in trial searches. The search terms are revised according to the results of the trial searches. Boolean operators "OR" and "AND" are used to join search terms. The search terms within the population, intervention, and outcome are joined with "OR" (e.g., for intervention, joined search term is "*knowledge* OR *semantic* OR *rationale* OR *decision*"). We followed the guidelines in [37] to reach the search string by formulating a combination of population, intervention, and outcome, which is "P AND I AND O". Google Scholar restricts the length of search strings, due to which the search string "P AND I AND O" is broken into three sub search strings: "P AND I", "P AND O", and "O AND I". Meanwhile, this database only supports paper searches by either full-text or title of a paper, and the former case leads to too many search results (e.g., using "P AND I" to search in Google Scholar by full-text returned 1,890,000 papers), consequently we decided to use three sub search strings to search in Google Scholar by title. Although the Google Scholar database contains many duplicated search results from other databases (i.e., DB1 to DB8), it still contributes several selected studies which are not indexed by any other database (e.g., [S16] is not indexed in DB1 to DB8).
(3) Formal searches are performed in two sub-steps sequentially, the automatic search in databases and manual search in target venues:
   (a) We search potentially relevant primary studies in databases. We limit the search on papers in computer science (we can set up the domains in the search of databases, including ScienceDirect, EI Compendex, ISI Web of Science, and SpringerLink) and on papers written in English (we can constrain the languages of published papers in the search of databases, including EI Compendex, ISI Web of Science, SpringerLink, and Google Scholar). Papers that are not in the domain of computer science or written in English, are manually excluded in the search results of the databases which cannot constrain subject domains and publication languages.
   (b) We perform manual browsing to identify the potentially relevant primary studies in target journals, conferences, and workshops.

The detailed search process using the search strategy and search terms has been elaborated in the beginning of Section 2.3.

Selected studies from both database and manual search were recorded in an Office Excel spreadsheet for duplication check and further analysis. Each entry of a selected study records the following information: authors' name, year of publication, title of publication, source (journal, conference, or workshop name), and publication type (journal paper, conference paper, workshop paper, book, book chapter, technical report, or others).

2.4. Data extraction and synthesis

To answer the RQs defined in Section 2.1.3, we extract specific data from the selected studies. Table 4 describes the data items (D1 to D8) extracted for the analysis in this review. D1 and D2 provide clues for the distribution of knowledge-based approaches over years and venues of publication. D3 (i.e., improved QA of SD) directly contributes to the answers of RQ1. D4 (i.e., knowledge-based approaches) can be used to answer RQ2. D5, D6, and D7 contribute to the answer of RQ3 and further discussion of knowledge-based approaches in SDt in Sections 3 and 4. To ensure that the data extraction results are unbiased, two authors performed the data extraction independently, and then one checked the data extraction results of the other, and finally they discussed and reached a consensus on the data extraction results. Since the evidence level of the selected studies is critical information for understanding the existing practice in the review topic, we employed a six-level classification for evidence evaluation proposed by Alves et al.

Table 4
Data items extracted from each study.

#	Item name	Description	Relevant RQ
D1	Publication year	In which year was the study published?	Study overview
D2	Venue	What is the name of the journal, conference, or workshop that the paper was published?	Study overview
D3	Quality attribute(s)	Which QA of SD does the proposed approach in the study try to improve?	RQ1
D4	Knowledge-based approach(es)	Which knowledge-based approach(es) is employed in the study to address the problem(s) in SDt?	RQ2
D5	What problem(s) addressed	What specific problem(s) concerning the QA of SD does the study try to address by knowledge-based approach(es)?	RQ3
D6	Benefit(s)	What are the benefit(s) of the knowledge-based approach(es) in SDt?	RQ3
D7	Cost(s)	What are the cost(s) in relation to the knowledge-based approach(es) in SDt?	RQ3
D8	Evidence level	What is the evidence level of the evaluation of the proposed approach?	Study quality

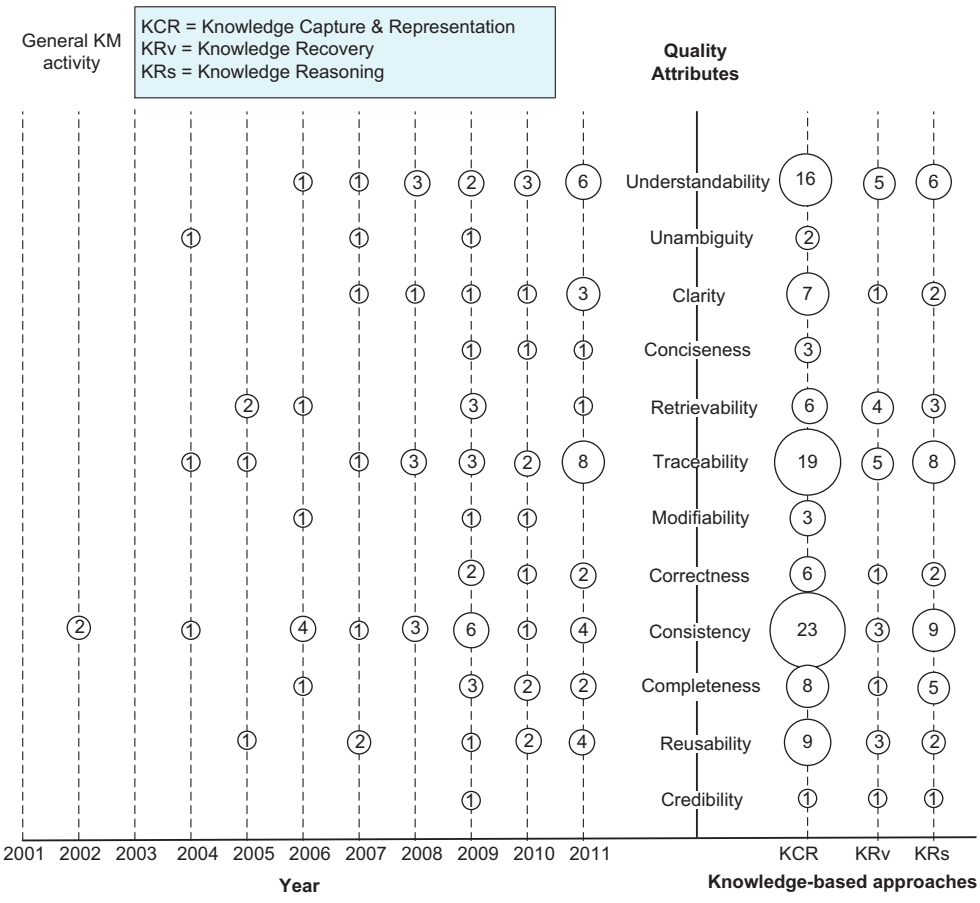

Fig. 4. Applications of knowledge-based approaches for software documentation (due to the space limitation of the bubbles, the study IDs in each bubble of this systematic map can be found in http://www.cs.vu.nl/~liangp/project/KbSDt/systematicmap.htm).

[10]. The evidence levels (from weakest to strongest) are defined as follows:

Level 0: No evidence (0.0).
Level 1: Evidence obtained from demonstration or toy examples (0.2).
Level 2: Evidence obtained from expert opinions or observations (0.4).
Level 3: Evidence obtained from academic studies, e.g., controlled lab experiments (0.6).
Level 4: Evidence obtained from industrial studies, e.g., causal case studies (0.8).
Level 5: Evidence obtained from industrial practice (1.0).

In order to identify the evidence level in a quantitative way, we assign 0.2 point to represent an increase of evidence level. The evidence levels from weakest (Level 0) to strongest (Level 5) are accordingly 0.0, 0.2, 0.4, 0.6, 0.8, and 1.0, as shown in parentheses of relevant evidence level listed above.

3. Results

3.1. Overview of results

After three phases of paper selection as shown in Fig. 2 and 60 studies (listed in Appendix A) are finally included in the review results. In this section, we answer the research questions from Section 2.1.3 by analyzing and synthesizing the extracted data from the selected studies.

Fig. 4 presents a systematic map of the applications of knowledge-based approaches for SDt, distributed over three dimensions: year of publication, knowledge-based approaches employed, and improved QA of the SD. The left part in Fig. 4 denotes the relationship between studies and year of publication. The number in a bubble represents the number of studies on a specific QA improved by knowledge-based approaches published in a certain year. The right part of Fig. 4 shows the relationship between studies and knowledge-based approaches. Similarly, the number in a bubble represents the number of studies applying a certain knowledge-based approach to improve a specific QA of the SD. In this systematic map, only three knowledge-based approaches are included: KCR, KRv, and KRs, because only the approaches that produce knowledge (as shown in Fig. 1) can enrich the content of SD and further improve the QA of SD. The remaining knowledge-based approaches: KRt, KS, and KR are approaches that only consume knowledge from the SD without making any change or improvements to the SD, are not included in this figure, and these knowledge-based approaches use SD to support other software development activities except for documentation. Note that, the sum of the numbers of studies on the column labeled KCR (103) exceeds the total number of selected studies (60), because one study may improve several QA. This situation is further elaborated in the third paragraph of Section 3.2. The analysis of this systematic map (Fig. 4) is also presented in Section 3.2.

Table 5 presents the distribution of selected studies over publication sources, including the publication name, type, count (i.e., the number of selected studies from each source), and the percentage of selected studies. The 60 selected studies are distributed over 36 publication sources, suggesting knowledge-based approaches

Table 5
Distribution of selected studies over publication sources.

Publication source	Type	Count	%
Journal of System and Software (JSS)	Journal	8	13.3
The Working IEEE/IFIP Conference on Software Architecture (WICSA)	Conference	7	11.7
International Requirements Engineering Conference (RE)	Conference	5	8.3
European Conference on Software Architecture (ECSA)	Conference	3	5.0
SHAring and Reusing Architectural Knowledge (SHARK)	Workshop	3	5.0
International Journal of Software Engineering and Knowledge Engineering (IJSEKE)	Journal	2	3.3
Asia–Pacific Software Engineering Conference (APSEC)	Conference	2	3.3
International Conference on Software Engineering (ICSE)	Conference	2	3.3
Managing Requirements Knowledge (MaRK)	Workshop	2	3.3
Relating Software Requirements and Architectures	Book	1	1.7
Decision Support Systems (DSS)	Journal	1	1.7
IEEE Software (IEEE SW)	Journal	1	1.7
IET Software	Journal	1	1.7
Information and Software Technology (IST)	Journal	1	1.7
International Journal of Cooperative Information Systems (IJCIS)	Journal	1	1.7
Journal of Software Maintenance and Evolution: Research and Practice (JSME)	Journal	1	1.7
Requirements Engineering (RE)	Journal	1	1.7
Software Process: Improvement and Practice (SPIP)	Journal	1	1.7
European Conference on Software Maintenance and Reengineering (CSMR)	Conference	1	1.7
International Conference of the Web Services (ICWS)	Conference	1	1.7
International Conference on Advances in Semantic Processing (SEMAPRO)	Conference	1	1.7
International Conference on Applied Computer Science (ACS)	Conference	1	1.7
International Conference on Design of Communication (SIGDOC)	Conference	1	1.7
International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)	Conference	1	1.7
International Conference on Program Comprehension (ICPC)	Conference	1	1.7
International Conference on the Quality of Software Architectures (QoSA)	Conference	1	1.7
International Conference on Software Engineering Research and Practice (SERP)	Conference	1	1.7
International Multi-Conference Software Engineering (SE)	Conference	1	1.7
International Symposium on Empirical Software Engineering (ISESE)	Conference	1	1.7
Proceedings of Innovation for Enterprise Software (PRIMIUM)	Conference	1	1.7
Software Engineering and Knowledge Engineering (SEKE)	Conference	1	1.7
Cooperative and Human Aspects on Software Engineering (CHASE)	Workshop	1	1.7
Engineering of Computer Based Systems (ECBS)	Workshop	1	1.7
Recommendation Systems for Software Engineering (RSSE)	Workshop	1	1.7
Software Engineering for Secure Systems (SESS)	Workshop	1	1.7
Total		60	100

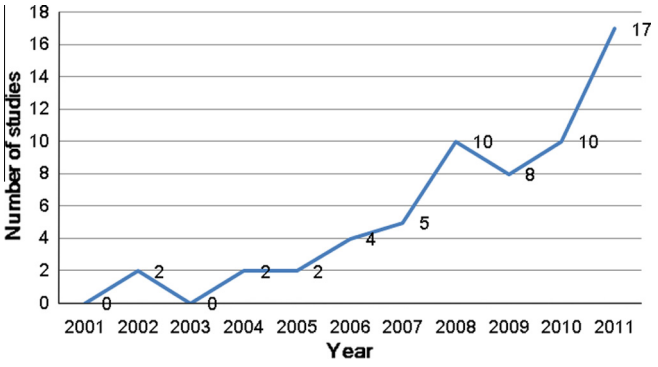

Fig. 5. Distribution of selected studies over time period.

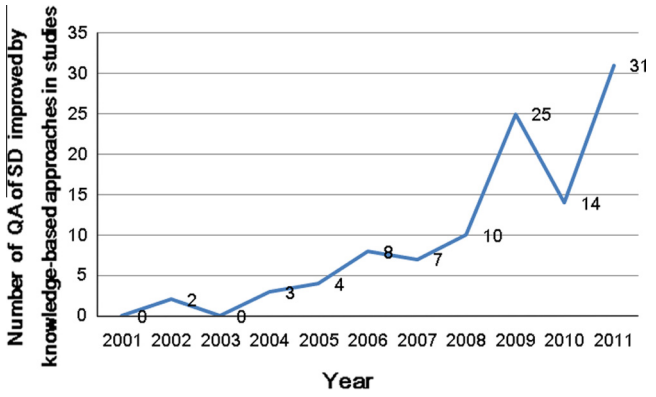

Fig. 6. Number of QA of SD improved by knowledge-based approaches over time period.

for SDt have been a widespread concern in the research community. As shown in Table 5, the leading venues in this study topic are JSS, WICSA, RE, and SHARK. WICSA is a leading conference in the software architecture community. RE is the flagship conference in the requirements engineering community, and JSS is a major journal on software systems. This result is to be expected since the topic of this SLR focuses on requirements and architecture documentation of software systems.

Fig. 5 shows the distribution of selected studies over year of publication from 2001 to 2011. The number of studies applying knowledge-based approaches in SDt has been increasing in the last decade (only 2 studies relevant to using knowledge-based approaches in SDt published in 2004, and this number grows to 17 in 2011. The number of studies increases since 2006 with a small variation in 2009). As shown in Fig. 6, more QA of SD are improved through knowledge-based approaches from 2001 to 2011 (the number of QA improved by knowledge-based approaches is 2 in 2002, and this number increases to 31 in 2011).

3.2. RQ1: Quality attributes of software documents and knowledge-based approaches

3.2.1. Quality attributes of software documents

In this RQ, we focus on the influence of using knowledge-based approaches to the quality of SDt. The three approaches

(KCR, KRv, and KRs) included in Fig. 4 are the approaches that may have an impact on the quality of SDt (i.e., the intervention of the SLR topic), and the QA of SD are the result of the intervention (i.e., the outcome). We collected the QA of SD from the literature and standards [1,2,22,36] as an initial set of QA. This list of QA is refined iteratively during the search process. For example, the QA *maturity* is further decomposed into the QA *correctness*, *consistency*, and *completeness* in [36]. According to the systematic map shown in Fig. 4, the following QA of SD are affected and improved by using knowledge-based approaches: *understandability* [22,36], *unambiguity* [1,22,36], *clarity* [2], *conciseness* [1], *retrievability* [36], *traceability* [22,36], *modifiability* [1], *correctness* [22,36], *consistency* [22,36], *completeness* [1], *reusability* [22], and *credibility* [36]. The meaning of these QA in the context of SD, and more specifically in SRD and SAD is elaborated below:

- *Understandability:* When stakeholders have different backgrounds, the language and concepts used to describe the requirements and architecture might not be understandable to everyone. It requires that the stakeholders can comprehend the meaning of the requirements with a minimum of explanation in SRD [22]. It is also a QA of the SAD. It means that an SAD conveys the intentions of the author when others read it [36].
- *Clarity:* It refers to the property that the document structure is simple and clear, which is a QA of SRD [2], but it can also be applied to the SAD and general SD.
- *Unambiguity:* denotes that each requirement stated has only one possible interpretation [1]. The difference between *unambiguity* and *clarity* is that *clarity* focuses on the structure of documents and *unambiguity* stresses the meaning of its contents.
- *Conciseness:* means that there is no redundancy and anomaly in the documentation [7]. It is a QA of general SD.
- *Retrievability:* How easy requirements or architecture information relevant to an information need can be obtained from the SRD or SAD. For example, annotated architectural knowledge provides an easy way to retrieve information from architecture documents [36].
- *Traceability:* represents the ability to describe and follow the life of a requirement in both a forward and backward direction in the SRD [19]. It is also a QA of the SAD, denoting the relationship of one architecture entity to other entities [3]. The *traceability* of documents can improve the *retrievability* and *modifiability* in SD.
- *Modifiability* means that any necessary changes in document structure and content can be made easily, completely, and consistently [1].
- *Consistency* means no subset of individual statements is in conflict [1]. It is a QA of SD, SRD, and SAD. The QA *correctness* and *completeness* are relative attributes to *consistency* in assessing the maturity of SD.
- *Correctness* may have specific meanings in different contexts. For example, every requirement represents something required of system to be built in a SRD [23]; an architectural decision in a SAD actually leads to a solution that meets the requirement [70]. It is a QA of SRD, SAD, and also a QA of general SD.
- *Completeness* means what the software is supposed to do and all the elements for understanding the requirements are included in the SRD [23]. For SAD, it means that all elements for understanding an architecture design are provided [1,4].
- *Reusability* means that elements, sentences, paragraphs, and sections can be easily adopted and adapted for use in a subsequent SD (e.g., requirements in requirements specifications [22] and viewpoints in architecture descriptions [4]). It is a QA of SAD, SRD, and SD.
- *Credibility* means that high accuracy is required for the content in SD [41]. It is a general QA of SD, including SAD and SRD. For instance, how reliable is the knowledge in a SAD when it evolves with changes in the implementation and requirements [36]. *Credibility* can improve the *reusability* of SD.

3.2.2. How knowledge-based approaches improve quality attributes of software documents

In the right-hand side of Fig. 4, we present the distribution of selected studies from the perspective of the QA of SD that are improved by various knowledge-based approaches. As shown in this figure, *consistency*, *traceability*, and *understandability* are the major QA of SD that are affected by using knowledge-based approaches, and the number of studies using various knowledge-based approaches differs dramatically. Detailed analysis of the QA of SD improved by different knowledge-based approaches is presented below.

KCR is widely employed for improving all QA of SD, but a significant difference exists in the numbers of studies using KCR for improving specific QA. KCR is mostly used to improve *consistency*, *traceability*, *understandability*, and *reusability*. This is because captured knowledge in SD, e.g., using models, can improve the *consistency* and *traceability* of SD, and knowledge representation improves the *understandability* and *reusability* of SD. KCR is less often used to address *unambiguity*, *conciseness*, and *credibility*. This is because *conciseness* and *credibility* are seldom considered as critical QA in SDt practice [41], and there is no consensus on whether accepting *unambiguity* as an indispensable QA of SD. For example, requirements specifications in a certain context deliberately introduce *ambiguity* in order to provide room for better stakeholder discussion and communication [47]. On the other hand, *unambiguity* of SAD is an important QA to improve the *understandability* of the architecture design [36].

KRv is mostly used to improve the following QA: *understandability*, *traceability*, and *retrievability*. As described in Section 2.1.1, KRv focuses on recovering knowledge which is not explicit. The *understandability* of SD can be improved when the knowledge in SD is made explicit. Recovered knowledge in SD can help trace back to the source where the recovered knowledge originally comes from (e.g., from recovered design decision to design artifact), therefore KRv can improve *traceability* and *retrievability* in SDt.

KRs is mainly used to improve the following QA: *understandability*, *retrievability*, *traceability*, *completeness*, and *consistency*. As described in Section 2.1.1, KRs stresses drawing a conclusion (i.e., new knowledge) from existing knowledge through inference. The traceability relationships between existing knowledge and new knowledge in SD can be recorded during the knowledge reasoning process, consequently KRs can improve the *traceability* and *retrievability* of SD. Meanwhile, identification of the knowledge reasoning process in SD can improve the *understandability* of SD, e.g., the reasoning process from a design decision to a design solution can improve the understanding of the solution [67]. When knowledge reasoning is performed on architecture documentation, a conceptual model can be used to check whether a specific architectural knowledge element is missing, while the *completeness* of architecture documents can be assessed and improved [36]. KRs can also improve the *consistency* of SD since reasoning can be used to check *consistency* in SD [36].

3.2.3. Quality attributes of software documents and their concerned elements in software documents

The QA of SD improved by knowledge-based approaches are concerned with various content elements of SD, i.e., the building blocks in a SD, for example, requirements in SRD and architecture design in SAD. We present the relationship between the QA of SD

Table 6
Relationship between QA of SD and their concerned documented elements.

QA of SD	Concerned documented elements		
	Most studied	Second most studied	Third most studied
Consistency	Between requirements S2, S7, S22, S26, S37, S44, S50, S55 **[8 studies]**	Between architectural design decisions S11, S21, S27, S57 **[4 studies]**	Between requirements and architectural design decision S6, S14, S25 **[3 studies]**
Traceability	From requirements to architecture design S4, S20, S24, S29, S34, S38, S48, S53, S54, S58 **[10 studies]**	Between requirements S30, S32, S36, S39, S44 **[5 studies]**	From architectural design decision to architecture design S4, S38, S39, S54, S58 **[5 studies]**
Understandability	Architecture design S1, S9, S10, S11, S19, S20, S45, S46, S47, S48, S52, S54 **[12 studies]**	Architectural design decision S9, S11, S19, S20, S46, S52, S54, S57 **[8 studies]**	Requirements S10, S15, S45, S46, S48, S50 **[6 studies]**
Reusability	Architecture design S5, S18, S21, S43, S51, S57 **[6 studies]**	Functional requirements S30, S33, S43, S44 **[4 studies]**	Architectural design decision S5, S18, S21, S57 **[4 studies]**
Completeness	Requirements S15, S23, S28, S40 **[4 studies]**	Architecture design S16, S19, S40 **[3 studies]**	Non-functional requirements S22, S49, S56 **[3 studies]**
Retrievability	Architecture design S5, S19, S27, S31, S34 **[5 studies]**	Architectural design decision S5, S19, S27, S31 **[4 studies]**	Architectural design rationale S5, S27, S31 **[3 studies]**

and the elements in SD in order to understand which QA of SD is mostly concerned with which SD elements. We select the six dominant QA of SD, *consistency*, *traceability*, *understandability*, *reusability*, *completeness*, and *retrievability*, and present the major content elements in SD concerning these QA in a descending order of the number of studies in each category shown in Table 6. The example studies listed in the table are elaborated below.

Consistency: Some knowledge-based approaches are used to improve the consistency between requirements, between architectural design decisions, and between requirements and architectural design decisions. For example, [S7] proposes a *model-based object-oriented approach for requirements engineering* (MORE) to improve the maintenance and consistency of requirements documents. Requirements documents in natural language are converted into objects and classes using ROMs (Requirement Object Models), which provide precise specification of the requirements semantics to prevent inconsistency of requirements.

Traceability: Some knowledge-based approaches are used to improve the traceability from requirements to architecture design, between requirements, and from architectural design decisions to architecture design. For instance, [S4] developed an *architecture design decision support system* (ADDSS) to codify design decisions that link requirements to architecture design.

Understandability: Knowledge-based approaches are frequently used in understanding architecture design, architectural design decisions, and requirements. For example, the domain architectural knowledge model constructed in [S19] is used to annotate architecture documentation, which improves the understandability of architecture design with design decisions.

Reusability: Knowledge-based approaches are used to improve the reusability of architecture design, functional requirements, and architectural design decisions. For instance, [S57] proposes a documentation framework consisting of four viewpoints to document architectural design decisions, which can facilitate the reuse of architecture design and design decisions (i.e., architectural solutions) in similar projects.

Completeness: Knowledge-based approaches are used to improve the completeness of requirements, architecture design, and non-functional requirements. For example, [S28] introduces the concept of "domain knowledge seed" in requirements evolution of agile development, which provides the core features in a given domain. The completeness of requirements specifications can be improved during system evolution when the seed (i.e., the domain knowledge) is evolved.

Retrievability: Knowledge-based approaches are used to improve the retrievability of architecture design, architectural design decision, and design rationale. For instance, [S19] employs a domain architectural knowledge model to annotate text-based architecture documents into semantically-enriched knowledge instances and store them in a knowledge repository, which can facilitate the retrieval of architectural knowledge, including architecture design and design decisions.

3.3. RQ2: Knowledge-based approaches in software documentation

3.3.1. Distribution of knowledge-based approaches in software documentation

The knowledge-based approaches classified in Section 2.1.1 are all used in SDt, either producing knowledge to SDt or consuming knowledge from SDt, including KCR, KRt, KR, KS, KRv, and KRs. We get the distribution of selected studies over knowledge-based approaches used in SDt as shown in Fig. 7.

KCR is the most frequently used approach in SDt. The reason of this result is that the output of KCR provides input to other knowledge-based approaches. For example, only when specific knowledge in SD is captured and represented, the knowledge can be readily shared, retrieved, and reused. The KCR approach is used in 54 studies, which means 90% of the total studies employ this knowledge-based approach in SDt, either improving various QA of SD or using SDt to support other development activities. For example, [S56] uses KCR to represent requirements knowledge and improve the *correctness*, *completeness*, and *consistency* of requirements specifications through KRs. [S41] uses different domain knowledge representations to support comprehensive description and domain knowledge reuse in requirements elicitation. Besides KCR, the most popular knowledge-based approach in SDt, all other approaches are evenly employed in SDt. There are 20 studies (33.3%) that employ KRs in SDt. For instance, [S25] uses formal reasoning with a knowledge base to check consistency

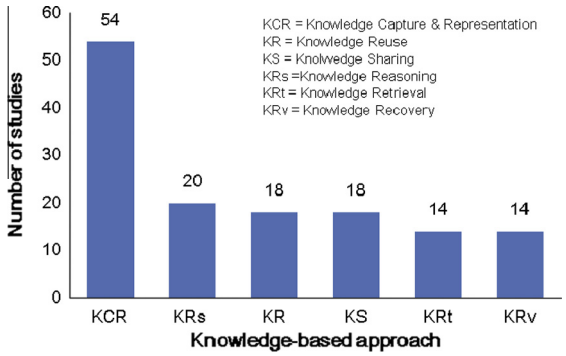

Fig. 7. Distribution of selected studies over knowledge-based approaches used in SDt.

between design and functional requirements specifications. KS and KR are both employed in 18 studies (30%). For example, [S16] developed *Ontobrowser*, a tool based on ontologies and semantic wikis, to share knowledge about software architecture and provides a collaborative way for architecture documentation. [S5] presents a model to represent and record rationale in architecture design, which can be reused and facilitates architecture understanding in new projects. KRt is used in 14 studies (23.3%). For example, [S54] developed a lightweight ontology and uses semantic annotation and query to improve the indexing and retrieval of software architectural knowledge. KRv is used in 14 studies (23.3%). For example, latent semantic indexing technique is used in [S34] to recover implicit semantic relationships between SDt and source code.

Note that the number of studies using certain knowledge-based approach in Fig. 7 is not equal to the sum of the numbers of the same knowledge-based approach in Fig. 4 (e.g., the number of studies using KCR is 54 in Fig. 7 and 103 in Fig. 4). The reason is that one knowledge-based approach (e.g., KCR, KRv, and KRs) can possibly impact several QA of SD. Meanwhile, the total number of studies (138) using various knowledge-based approaches in Fig. 7 is larger than the number of selected studies (60). This is due to the fact that one study may apply several knowledge-based approaches in SDt.

3.3.2. General and specific knowledge-based approaches

Specific knowledge-based approaches can be refined and classified by analyzing the six general knowledge-based approaches used in selected studies (i.e., KCR, KRs, KRv, KR, KS, and KRt). A specific knowledge-based approach employs a knowledge technique to support general knowledge-based approaches. Classification of the specific knowledge-based approaches is helpful in understanding the underlying characteristics and realization mechanism of the general knowledge-based approaches used in SDt. For example, conceptual modeling can be used to support all general knowledge-based approaches. Table 7 presents the studies classified in two dimensions: employed specific knowledge-based approach and general knowledge-based approach. For example, [S38] uses conceptual modeling, *tactic traceability information models* (TTIMs), to capture and represent the traceability links from architecture design to architectural design decisions in architecture documents. As shown in the second column of Table 7, conceptual modeling and natural language processing are two mostly used specific knowledge-based approaches for capturing and representing knowledge in SDt. The example studies of using specific knowledge-based approaches to realize general knowledge-based approach are elaborated below.

Conceptual modeling is the activity of formally describing some aspects of the physical and social world around us for the purpose of understanding and communication [50].

- *KCR:* [S39] develops a traceability model to capture and represent knowledge elements that are essential to comprehensively manage changes in software development and documentation, e.g., product, rationale, and version.
- *KRv:* [S17] constructs an argumentation model based on IBIS (issue-based information system) notations, including concepts *Issue*, *Position*, *Argument*, and *Decision*, to express tacit knowledge in argumentation of requirements engineers, which can help stakeholders to better understand the evolution of requirements.
- *KRs:* [S53] develops a conceptual model with reasoning rules and concept relationships implemented in an ontology to support reasoning from requirements to architecture design.
- *KRt:* [S4] employs specific attribute templates developed in *architecture design decision support system* (ADDSS) to characterize and store architectural knowledge, e.g., architecture styles, which can be further retrieved as design solutions to satisfy various user needs.
- *KR:* [S33] uses an *activity-based quality model* (ABQM) to get normalized requirements for reusing in requirements documents.
- *KS:* [S16] employs a *service-oriented architecture* (SOA) ontology to share SOA knowledge to domain experts and technique people.

Natural language processing aims to convert human language into a formal representation that is easy for computers to manipulate [21].

- *KCR:* [S34] employs *Latent Semantic Indexing*, a specific natural language processing technique, to capture and mathematically represent information from requirements or architecture documentation, which can be used to identify traceability links between documentation and code.
- *KRv:* [S36] uses *Latent Semantic Indexing* to recover implicit traceability links between requirements in requirements documentation by combining textual and structural information.
- *KRs:* [S25] proposes an approach to translate functional requirements specifications expressed in natural language into UML models with supporting axioms, which can facilitate consistency checking between requirements and design specifications through reasoning.

Annotation in documentation is an activity to tag the content in documents with various tags, for example in a folksonomy multiple users tag particular content with a variety of terms. Semantic annotation using ontology is that experts tag instance data (e.g., text in documents) with ontology classes [58].

- *KCR:* [S19] uses concepts (e.g., *Requirement*, *Design Decision*) in a domain architectural knowledge model to annotate architecture documents explicitly into architectural knowledge instances in order to improve understandability of architecture documents.
- *KRv:* [S54] develops a lightweight ontology to semantically annotate architecture documents, such as requirements, architecture design, and design decisions, which leads to improved retrievability and traceability of knowledge in architecture documentation.
- *KRs:* [S53] develops and uses an ontology to semantically annotate the traceability links between architecture requirements and design with reasoning rules in order to support the co-evolution between architectural requirements and design.
- *KRt:* [S37] develops the *4everedit* tool to support structured text-based documentation, e.g., architecture and requirements documents, by annotation with extended XML. The knowledge annotated from documents can then be retrieved by stakeholders for various purposes, e.g., checking structural and internal consistency of documents.
- *KR:* Architecture properties are used in [S43] to annotate architecture documents, and annotated architecture design in architecture documents can be reused by searching these properties.
- *KS:* [S31] uses an *advanced mapping quality prediction model* (AMQPM) to predict the sharing quality of architectural knowledge that is annotated from architecture documents by domain experts.

Table 7
Classification of studies by specific knowledge-based approach and general knowledge-based approach.

Specific knowledge-based approach	General knowledge-based approach					
	KCR	KRv	KRs	KRt	KR	KS
Conceptual modeling	S1, S2, S3, S4, S5, S6, S7, S10, S11, S12, S16, S17, S19, S20, S21, S22, S23, S24, S26, S27, S29, S30, S32, S35, S38, S39, S40, S41, S42, S44, S45, S46, S47, S48, S49, S51, S52, S53, S54, S55, S57, S58, S59, S60 **[44 studies]**	S4, S6, S15, S17, S19, S20, S21, S27, S33, S36, S54, S60 **[12 studies]**	S4, S5, S16, S19, S26, S27, S32, S35, S40, S48, S49, S52, S53, S54, S57, S58, S60 **[17 studies]**	S2, S4, S17, S28, S33, S44, S51, S53, S54, S57 **[10 studies]**	S4, S5, S7, S12, S17, S18, S19, S21, S29, S32, S33, S38, S41, S57, S58 **[15 studies]**	S4, S5, S10, S16, S18, S20, S21, S27, S30, S32, S45, S51, S54, S58, S60 **[15 studies]**
Natural language processing	S8, S9, S15, S23, S25, S34, S36, S50, S56 **[9 studies]**	S9, S15, S34, S36 **[4 studies]**	S25, S50, S56 **[3 studies]**			
Annotation	S14, S19, S37, S43, S53, S54 **[6 studies]**	S19, S54 **[2 studies]**	S19, S53,S54 **[3 studies]**	S14, S37, S53, S54 **[4 studies]**	S18, S43 **[2 studies]**	S18, S31, S54 **[3 studies]**

3.3.3. Knowledge-based approaches and documented content

Knowledge-based approaches to SDt are used with various requirements and architectural knowledge content in SD. To further understand the relationship between knowledge-based approaches and SD knowledge content, the selected studies are classified in two dimensions: knowledge-based approach and SD knowledge content, as presented in Table 8. The number of studies for each category is shown in each cell of Table 8, for example, KCR is mostly used to capture and represent *architecture design* in SD, which includes 28 studies. Besides documenting the basic knowledge desired in a typical SRD and SAD, i.e., *requirements* and *architecture design* respectively, knowledge-based approaches have also been used to capture *architectural design decisions* and *architectural design rationale*, i.e., the core of architectural knowledge [24]. The use of knowledge-based approaches in SDt can be extended to the content of requirements knowledge [43], especially *requirements decisions* and *requirements rationale* [44].

3.4. RQ3: Costs and benefits of using knowledge-based approaches

Using a bottom-up approach by analyzing the data items D5, D6, and D7 in Table 4, we identify the following categories of costs and benefits of using knowledge-based approaches in SDt, which are elaborated in this section.

3.4.1. Costs of using knowledge-based approaches

The cost of knowledge-based approaches in SDt refers to the expenditure of employing knowledge-based approaches in SDt, e.g., time, money, and labor. We identify four cost categories from SDt literature [15,17,41]: the cost of document creation [41], document maintenance and evolution [17], information retrieval from documents [15], and document distribution [17]. Since few studies evaluate the cost of using knowledge-based approaches in SDt quantitatively, we categorize the cost of SDt qualitatively in this SLR, i.e., identifying the types of cost relevant to using knowledge-based approaches in SDt. Note that, four studies did not explicitly discuss the cost of knowledge-based approaches in SDt [S8, S10, S47, S49].

Document creation: Studies in this category focus on the cost of development of documents using various documentation approaches according to the needs of document users. For instance, recording only domain-specific architectural knowledge in architecture documents using the *model-driven development* approach reduces the cost of document creation [S18]. Savolainen and Mannisto present an approach of considering stakeholder conflicts in architecture documentation [S46]. They develop a conflict-centric architectural view to document architecture trade-offs in quality requirements that are traditionally scattered among multiple architectural views. When the most important conflicts have been described, the documentation process will stop, which will consequently reduce document creation cost.

Document maintenance and evolution: Document content is prone to change quickly and needs to be updated and synchronized with other changed software artifacts. Meanwhile, SD should be organized to make them more accessible. Studies in this category focus on the cost caused by the abovementioned activities. For instance, *knowledge assisted agile requirements evolution* (K-gileRE) presents a "domain knowledge seed" for given domain and associated knowledge elements [S28]. The correctness, consistency, and completeness of requirements specifications are improved when analysts modify the seed according to the online recommendation, which can reduce the document maintenance cost. *Scenario-based documentation and evaluation method* (SceMethod) can indicate

"what"-"how"-"why" features for architectural design decisions [S6]. It keeps the architectural knowledge complete and consistent during architectural evolution, which can reduce document maintenance and evolution cost.

Information retrieval from documents: Studies in this category focus on the cost of retrieving useful information from SD. For instance, capturing architecturally significant traceability links through annotating and extracting information from existing architecture documents can reduce the time of information retrieval from documents [S38]. Using the latent semantic indexing technique requires less preprocessing and manipulation effort to support information retrieval from source code and associated documentation [S34]. The architecture properties are represented as an XML file which provides an easy way to search in the documents [S43].

Document distribution: Studies in this category focus on the cost of dissemination and sharing of the documents to various users who need the documents. For instance, by enhancing the traditional object-oriented programming paradigm, the approach proposed in [S45] enables adaptable SD as part of the source code. SD and code can be disseminated simultaneously which consequently reduces the cost of document distribution. The *4everedit* tool uses a pre-defined document structure to represent document knowledge, which can be used to maintain structural and internal consistency of SD in a team-based documentation environment [S37]. In such an environment, this tool can further reduce document distribution and sharing cost among documentation users.

Table 9 presents the summary of selected studies over cost categories, including the selected studies and percentage of selected studies in each category. As shown in Table 9, the cost of retrieving information from documents is the major concern of using knowledge-based approaches in SDt. The costs of creating, maintaining, and evolving documents are also important factors when applying knowledge-based approaches in SDt. Note that, one study (e.g., [S54]) may cover several cost categories of using knowledge-based approaches in SDt, and consequently the sum of the percentages of studies from each category exceeds 100% in Table 9.

3.4.2. Benefits of using knowledge-based approaches

The benefits discussed in this section indicate that some activities in software development are supported by using knowledge-based approaches in SDt, with a focus on requirements engineering or architecting activities. Nine benefit categories are identified from selected studies.

Requirements elicitation: Studies in this category focus on improving the efficiency of requirements elicitation. For instance, Li et al. propose a model-based approach, which provides a high level of requirements abstraction in a domain specific model to elicit requirements in scientific computing. This approach makes the elicited requirements easier to understand and reduces the learning effort for domain scientists [S30].

Requirements analysis: Studies in this category focus on detecting and resolving conflicts between requirements, discovering boundary of a software system and interaction with its environment [5]. For instance, the *model-based object-oriented approach* (MORE) proposed in [S7] can capture and model domain knowledge, which is used to evaluate the consistency, completeness, traceability, and reusability in requirements analysis. *TExtual aSSIstent* (TESSI) is a requirements specification and analysis tool, which is used to transform requirements specification and its constraints into a problem ontology, and further checks inconsistency in requirements specification through ontology reasoning [S26].

Requirements comprehension: Studies in this categories focus on facilitating understanding of requirements. For instance, [S2] proposes a way of structuring and representing the requirements specifications to improve requirements comprehension. In this approach, the requirements specifications consist of rigorous description of different views, e.g., use case view, context view with underlying domain model, which is helpful for understanding requirements.

Requirements evolution: Studies in this category focus on changes in requirements after initial requirements have been elicited. Changes in requirements can be adding to, removing, or modifying existing requirements [63]. For example, [S39] integrates software configuration management with a traceability model, which can represent knowledge elements that are essential to comprehensively manage changes, to support change management during the evolution of requirements artifacts.

Requirement traceability: Studies in this category focus on the ability to describe and follow the life of a requirement, in both a forward and backward direction [33]. For instance, [S36] develops a method to recover traceability links in requirements documentation using a combination of textual and structural information.

Co-evolution of requirements and architecture: Studies in this category focus on bridging the gap between requirements and architecture. For instance, the *Language for Integrated Software Architecture* (LISA) model links requirements decisions to architectural elements, which maintains consistency between requirements and architecture views [S58]. Tang et al. introduce a generic ontology model with a semantic wiki to support the co-evolution between architecture requirements and design [S53]. The semantic wiki developed in this work supports the traceability ontology model and semantic annotation, which help users to retrieve co-evolved requirements and architecture designs.

Architecture understanding: Studies in this category focus on better understanding of architecture design through architectural knowledge. For instance, de Boer and van Vliet employ latent semantic analysis technique to discover the semantic structure in a set of architecture documents, which provides a reading guide for architecture documents and further improves architecture understanding [S9].

Architecture evolution: Studies in this category focus on adapting an existing architecture to cope with the evolution requirements [11]. For instance, [S59] proposes an approach to support architecture evolution (generation of a list of tasks maintainers can perform to evolve the system) of software product lines using required architectural knowledge, which is codified in a meta-model.

Architecture recovery: Studies in this category focus on recovering architecture design and related architectural knowledge that is not explicit in existing architecture documents. For instance, Feilkas et al. proposed to recover and refine implicit architectural knowledge (e.g., violations between current architecture and the intended architecture) through nonconformance checking and discussion between the two architectures represented in XML [S14]. *Architectural design decision recovery approach* (ADDRA) uses a template based on a conceptual model to recover and document architectural design decisions after the fact [S20].

Table 10 presents the summary of selected studies over benefit categories, including the selected studies and percentage of selected studies in each category. As shown in this table, using knowledge-based approaches in SDt mainly supports the following activities: architecture understanding (41.7%, 25 studies),

Table 8
Classification of studies by knowledge-based approach and SD knowledge content.

Knowledge content in SD	Knowledge-based approach					
	KCR	KRv	KRs	KRt	KR	KS
Architecture design	S1, S3, S4, S5, S6, S9, S14, S16, S19, S20, S21, S27, S29, S34, S38, S40, S42, S43, S45, S46, S47, S48, S51, S52, S53, S54, S58, S59 [**28 studies**]	S4, S6, S9, S19, S20, S21, S27, S34, S54 **[9 studies]**	S4, S5, S16, S19, S27, S40, S48, S52, S53, S54, S58 **[11 studies]**	S4, S14, S51, S53 S54 **[5 studies]**	S4, S5, S18, S19, S21, S29, S43, S58 **[8 studies]**	S4, S5, S10, S13, S16, S18, S20, S21, S27, S31, S45, S51, S58 **[13 studies]**
Architectural design decision	S3, S4, S5, S6, S9, S11, S12, S19, S20, S21, S27, S29, S38 S39,, S46, S52, S53, S54, S57, S58, S59 **[21 studies]**	S4, S6, S9, S19, S20, S21 S27 S54 **[8 studies]**	S4, S5, S19, S27, S40, S52, S53, S54, S57, S58 **[10 studies]**	S4, S53, S54, S57 **[4 studies]**	S4, S5, S12, S18, S19, S21, S29, S38, S57, S58 **[10 studies]**	S4, S5, S13, S18, S20, S21, S27, S31, S54, S58 **[10 studies]**
Architectural design rationale	S4, S5, S6, S17, S20, S27, S38, S39, S48, S52, S57, S58 **[12 studies]**	S4, S20, S27 **[3 studies]**	S5, S16, S27, S48, S52, S57, S58 **[7 studies]**	S4, S57 **[2 studies]**	S4, S5, S38, S57, S58 **[5 studies]**	S5, S13, S20, S27, S31, S54, S58 **[7 studies]**
Architectural view	S4, S6, S20, S43 **[4 studies]**	S4, S6, S20 **[3 studies]**		S4 **[1 study]**	S4, S18, S43 **[3 studies]**	S4, S18, S20 **[3 studies]**
Architectural pattern	S16, S38, S43, S60 **[4 studies]**	S4, S60 [**2 studies**]		S4 [**1 study**]	S4, S43 [**2 studies**]	S4, S16, S60 **[3 studies]**
Architecturally significant requirement	S3, S38, S58, S10, S43, S46, S53 **[7 studies]**		S53, S58 [**2 studies**]	S53 [**1 study**]	S38, S43, S58 **[3 studies]**	S58 **[1 study]**
Requirement	S2, S6, S8, S12, S15, S19, S23, S25, S26, S34, S36, S37, S39, S40, S41, S45, S48, S50, S55 **[19 studies]**	S4, S15, S19, S34, S36, S54 **[6 studies]**	S19, S25, S26, S40, S50 **[5 studies]**	S2, S4, S28, S37 **[4 studies]**	S4, S12, S19, S32, S41 **[5 studies]**	S4, S10, S32, S45 **[4 studies]**
Functional requirement	S7, S11, S22, S30, S35, S43, S44, S52, S53, S54, S56 **[11 studies]**	S33, S54 [**2 studies**]	S35, S52, S53 S54, S56 **[5 studies]**	S33, S44, S53, S54 **[4 studies]**	S7, S33, S43 **[3 studies]**	S30, S31, S54 **[3 studies]**
Non-functional requirement	S7, S11, S22 S30, S38, S44, S46, S52, S53, S54, S56 **[11 studies]**	S33, S54 [**2 studies**]	S32, S49, S52, S53, S54, S56 **[6 studies]**	S33, S44, S53, S54 **[4 studies]**	S7, S32, S33, S38, S43 **[5 studies]**	S30, S31, S54 **[3 studies]**
Requirements rationale	S17, S53 [**2 studies**]	S33 [**1 study**]	S32, S53 [**2 studies**]	S17, S33, S53 **[3 studies]**	S17, S32, S33 **[3 studies]**	S32 [**1 study**]

requirements elicitation (21.7%, 13 studies), and co-evolution of requirements and architecture (15.0%, 9 studies). Similar to the cost categories, one study (e.g., [S52]) may cover several benefit categories of using knowledge-based approaches in SDt, and consequently the sum of the percentages of studies from each category also exceeds 100% in Table 7.

Fig. 8 presents the distribution of selected studies over the benefit categories for requirements engineering and architecting activities. We find that knowledge-based approaches in SDt are evenly used in and benefit both requirements engineering and architecting activities (34 vs. 35 studies). This result shows that knowledge-based approaches are promising and appropriate to support the documentation from requirements to architecture. Note that one study may benefit both requirements engineering and architecting activity (e.g., [S54], to support co-evolution from requirements to architecture), and consequently the sum of the studies in this figure (69) exceeds the number of selected studies (60).

3.5. Evidential support

According to the criteria to identify the evidence level of a study provided in Section 2.4, we evaluated the evidence level of all the selected studies and show the distribution of studies in each evidence level in Fig. 9. For instance, [S17] introduces an approach which adapts the IBIS (issue-based information system) argumentation model to characterize and capture tacit requirements knowledge (e.g., requirements rationale) in order to improve requirements documentation, but this work only demonstrates the application of the approach with a toy example (Evidence level 1). *4everedit* is a tool that represents document knowledge following a pre-defined document structure, which facilitates maintenance of structural and internal consistency of SD [S37]. Twenty-six editors, from more than five companies, have successfully applied this tool in a large process engineering project for one year; consequently the evidence level of [S37] is obtained from industrial practice (Evidence level 5). From Fig. 9, we find that more than 50% (33 out of 60) studies on using knowledge-based approaches in SDt are supported by academic studies (Evidence level 3). Only 13.3% (8 out of 60) studies have been validated in industrial practice (Evidence level 5). The results are understandable because of the high cost and risk of evaluating knowledge-based approaches in SDt through industrial practices. No evidence is obtained from expert opinions or observation (Evidence level 2). The reason is that there are no consistent criteria for experts to evaluate the quality of SDt.

Fig. 10 presents the distribution of selected studies in two dimensions: evidence level and knowledge-based approach employed in SDt. The number in a bubble represents the number of studies that use certain knowledge-based approach and are supported by a specific evidence level (e.g., the biggest bubble denotes

Table 9
Classification of studies by cost categories of using knowledge-based approaches.

Cost categories	Selected studies	%
Information retrieval from documents	S2, S3, S9, S14, S15, S20, S23, S24, S25, S26, S29, S31, S34, S36, S38, S40, S43, S44, S51, S53, S54, S58, S60	38.3
Document creation	S1, S4, S5, S6, S11, S12, S17, S18, S20, S21, S22, S30, S43, S45, S48, S50, S54, S55, S57	31.7
Document maintenance and evolution	S4, S6, S7, S14, S16, S19, S27, S28, S35, S39, S41, S42, S46, S52, S56, S59, S60	28.3
Document distribution	S13, S19, S31, S32, S33, S37, S45	11.7

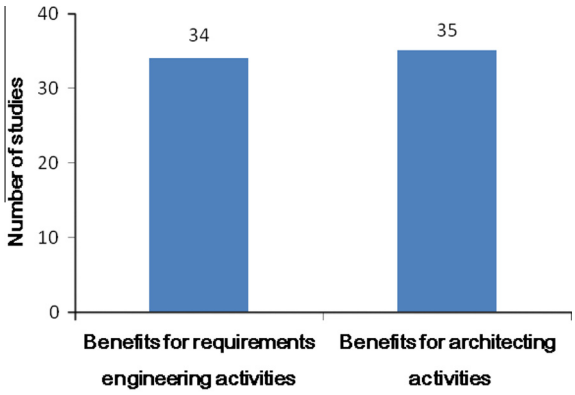

Fig. 8. Distribution of studies over the benefit categories for requirements engineering and architecting activities.

that 27 studies use KCR in SDt and are evaluated by academic studies). Note that there is a difference between the sum of all the bubbles (i.e., numbers of studies) in Fig. 10 (138) and number of selected studies (60), because one study may employ several knowledge-based approaches in SDt. From Fig. 10 we find that all knowledge-based approaches are evaluated in at least four studies with industrial evidence (i.e., industrial studies or industrial practice), but all the knowledge-based approaches used in SDt are mainly supported by evidence obtained from academic studies, except for KRt.

4. Discussion

4.1. Scope of the systematic review

This SLR focuses on how knowledge-based approaches are employed in SDt, in terms of improving the quality of SDt and use SD to support software development activities.

When conducting this SLR, we consider the QA of SD in the search process as part of the search terms (i.e., the outcome in the PICO criteria of a SLR) collected from standards and literature on SD, SAD, and SRD, so that the results and conclusions of this SLR can be applied to SAD and SRD. Meanwhile, we use the most general term "knowledge" as part of the search terms to maximize the coverage of potentially-relevant studies retrieval and consequently ensure that the results of this SLR can cover all studies that use certain knowledge or knowledge-based approaches.

This SLR focuses on studies that elaborate knowledge-based approaches, not on studies that only introduce knowledge-based tools. For instance, *KaitoroBase* is an architecture documentation tool, which provides support for non-linear navigation and visualization of SAD through an underlying conceptual model [64]. This work only introduces the structure of the tool and its application, without any description of the employed knowledge-based approach that the tool implements. This study is therefore excluded from this SLR.

4.2. Study quality assessment

The quality of data extraction and synthesis of the selected studies of this SLR are assessed in this section. An assessment instrument is presented in Table 11, which is adapted from the criteria for study quality assessment proposed in [9,28]. We include five questions in this instrument to assess the quality of extracted data. Q1, Q2, and Q5 are adopted from [9,28] while Q3 and Q4 are proposed according to the scope and RQs of this SLR. This instrument uses a three-grade scale score (Yes = 1 point, No = 0 point, and Partially = 0.5 point) to answer Q2 to Q5. The score of Q1 is directly collected from the data item D8 of Table 4, i.e., evidence level. The sum of the scores of all the assessment questions for a study can reflect the quality of a study.

The quality assessment on the selected studies is also useful to increase the accuracy of the data extraction results. The quality assessment results are showed in Table 12 according to the assessment questions described in Table 11. The scores of all the studies are no less than 3.10 and the average score is 4.17. The overall quality of the selected studies is acceptable. Since we use the inclusion and exclusion criteria specified in Section 2.2 when selecting studies, the scores of Q2, Q3, and Q4 are high. i.e., the average scores of Q2 and Q3 are both greater than 0.93 and all the studies get full scores on Q4. The high scores of Q2 and Q3 also show that the results of data extraction are in line with the two key concepts of this SLR, i.e., impact to software documentation and knowledge-based approaches employed. Meanwhile, the studies that get low total score in Table 12 are checked again against the inclusion and exclusion criteria, e.g., [S32], to guarantee the quality of study selection results.

4.3. Validity threats

According to the guidelines for analyzing the validity threats to SE methods and processes [73], four types of validity threats are identified. We discuss these potential threats that influence the data extraction and the findings of this SLR in this section.

Construct validity: The main constructs in this review are the two basic concepts "knowledge-based approaches" and "software documentation". For the first concept, we use term "knowledge" and its synonyms to make sure that all selected studies are relevant to knowledge-based approaches or using knowledge. For the second concept, QA of SD can reflect the effect of intervention to SDt. QA of SD and their synonyms,

Table 10
Classification of studies by benefit categories of using knowledge-based approaches.

Benefit categories	Selected studies	%
Architecture understanding	S1, S3, S4, S5, S6, S9, S10, S13, S18, S19, S24, S27, S31, S38, S40, S42, S43, S46, S47, S48, S51, S52, S57, S58, S60	41.7
Requirements elicitation	S15, S17, S22, S23, S30, S33, S37, S41, S44, S45, S49, S50, S56,	21.7
Co-evolution of requirements and architecture	S3, S11, S12, S18, S24, S36, S53, S54, S58	15.0
Requirements comprehension	S2, S8, S32, S33, S34, S35, S39, S55	13.3
Architecture evolution	S1, S6, S16, S19, S21, S52, S59	11.7
Requirements analysis	S7, S15, S25, S26, S50, S56	10.0
Requirements evolution	S11, S28, S39, S45, S50	8.3
Architecture recovery	S14, S20, S60	5.0
Requirements traceability	S29, S36	3.3

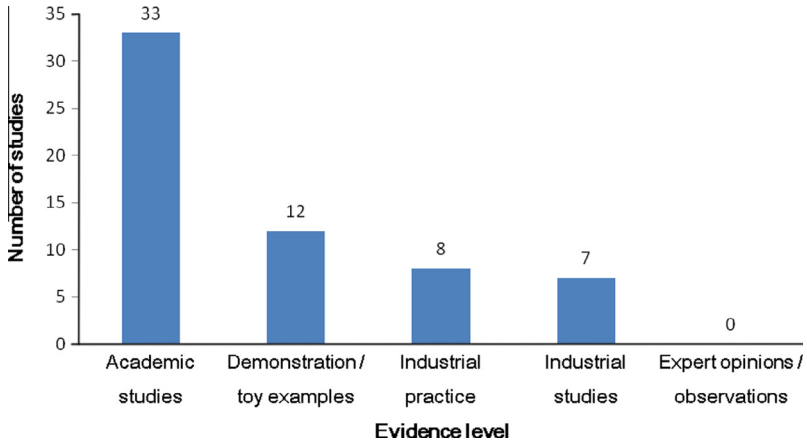

Fig. 9. Distribution of selected studies over evidence levels.

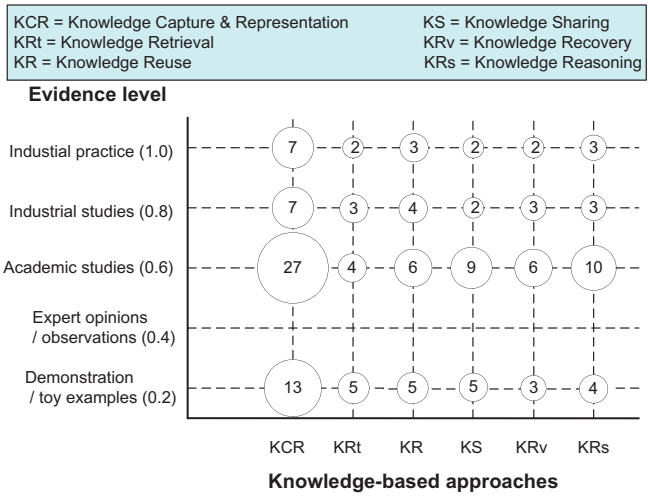

Fig. 10. Distribution of studies by evidence level and knowledge-based approach.

collected and refined from SDt standards and literature, are used to ensure high coverage of potentially-relevant studies on the influence to SDt from database search. Meanwhile, manual search from literature sources is performed complementary to database search to ensure that relevant studies are covered as much as possible. Specific journals and conferences on KM are not included in the literature sources for the manual search due to the limitation of our knowledge, which may cause the missing of related studies. This threat is partially mitigated by including the general intervention term "knowledge" in the search terms for the database search.

Internal validity in a SLR focuses on whether a research is adequately designed and executed to produce reliable findings, and particularly whether the results really follow from the data [30]. As a threat to the internal validity, researchers may end up with different data extraction and analysis results. The data extraction is performed collaboratively by two authors, and any conflicts are discussed and resolved by all the authors. In this way, we try to mitigate the threats due to personal bias on study understanding.

External validity is concerned with establishing the generalizability of the SLR results, which is related to the degree to which the primary studies are representative for the review topic. In order to mitigate external threats, the search process described in Section 2.3 is defined after several trial searches. We tested the coverage and representativeness of retrieved studies, including automatic database search, manual search, and references scan.

Reliability: It is possible that some studies excluded in this review should have been included. To mitigate the threats to reliability, the selection process and the inclusion and exclusion criteria are carefully designed and discussed by authors to minimize the risk of exclusion of relevant studies.

4.4. Further research

This SLR has illuminated several promising research directions that are critical but underexplored in current research and practice:

(1) How to employ knowledge-based approaches to improve the QA of SD. This area has not received much attention and the claims lack evidential support. For instance, the assertions to use knowledge-based approaches to improve *credibility*, *conciseness*, and *unambiguity* of SD are hardly supported. There are still many open questions to be answered, e.g., how to define the quantitative metrics for evaluating *credibility*, *conciseness*, and *unambiguity* of SD. Furthermore, text retrieval approaches have been used to evaluate *conciseness* of queries to software artifacts from a system [35], and the approaches may be adapted to evaluate *conciseness* of

Table 11
Questions on study quality assessment.

#	Questions
Q1	In which evidence level the proposed approach of the study is evaluated? (the answer of this question can be collected from data item D8 of Table 4)
Q2	Is there a clear statement of the benefits and costs for software documentation in this study?
Q3	Is there a clear statement of what the knowledge-based approach employed is in the study?
Q4	Is there an adequate description of what QA of SD are improved by the knowledge-based approach employed?
Q5	Are the limitations of this study discussed explicitly?

Table 12
Quality assessment results of selected studies.

Study ID	Q1	Q2	Q3	Q4	Q5	Total score	Study ID	Q1	Q2	Q3	Q4	Q5	Total score
S1	0.2	1.0	1.0	1.0	0.0	3.2	S31	1.0	1.0	1.0	1.0	1.0	5.0
S2	0.2	1.0	1.0	1.0	0.0	3.2	S32	0.6	0.5	1.0	1.0	0.0	3.1
S3	0.6	1.0	1.0	1.0	1.0	4.6	S33	1.0	0.5	1.0	1.0	1.0	4.5
S4	0.6	1.0	1.0	1.0	1.0	4.6	S34	0.6	1.0	1.0	1.0	1.0	4.6
S5	0.6	1.0	1.0	1.0	0.0	3.6	S35	0.6	0.5	1.0	1.0	1.0	4.1
S6	0.8	1.0	0.5	1.0	1.0	4.3	S36	0.6	1.0	1.0	1.0	1.0	4.6
S7	0.2	0.5	1.0	1.0	1.0	3.7	S37	1.0	1.0	1.0	1.0	1.0	5.0
S8	0.2	1.0	0.5	1.0	0.5	3.2	S38	1.0	0.5	1.0	1.0	0.0	3.5
S9	0.6	1.0	1.0	1.0	1.0	4.6	S39	0.6	0.5	1.0	1.0	1.0	4.1
S10	0.8	1.0	1.0	1.0	0.5	4.3	S40	0.6	1.0	1.0	1.0	0.0	3.6
S11	0.6	1.0	0.5	1.0	1.0	4.1	S41	0.6	1.0	1.0	1.0	1.0	4.6
S12	0.6	1.0	1.0	1.0	0.0	3.6	S42	0.8	1.0	1.0	1.0	1.0	4.8
S13	0.6	1.0	1.0	1.0	1.0	4.6	S43	0.2	1.0	1.0	1.0	1.0	4.2
S14	0.8	0.5	1.0	1.0	1.0	4.3	S44	0.6	1.0	1.0	1.0	1.0	4.6
S15	0.6	0.5	0.5	1.0	1.0	3.6	S45	0.2	1.0	1.0	1.0	0.0	3.2
S16	0.2	1.0	1.0	1.0	0.0	3.2	S46	0.6	1.0	1.0	1.0	1.0	4.6
S17	0.2	1.0	1.0	1.0	0.0	3.2	S47	0.6	1.0	1.0	1.0	1.0	4.6
S18	0.2	1.0	1.0	1.0	0.5	3.7	S48	0.6	1.0	0.5	1.0	1.0	4.1
S19	0.8	1.0	1.0	1.0	1.0	4.8	S49	0.6	1.0	1.0	1.0	1.0	4.6
S20	0.6	1.0	1.0	1.0	1.0	4.6	S50	0.6	1.0	1.0	1.0	1.0	4.6
S21	0.6	1.0	1.0	1.0	0.5	4.1	S51	0.6	1.0	1.0	1.0	1.0	4.6
S22	0.6	1.0	1.0	1.0	1.0	4.6	S52	1.0	1.0	1.0	1.0	1.0	5.0
S23	0.6	1.0	1.0	1.0	1.0	4.6	S53	0.6	1.0	1.0	1.0	0.0	3.6
S24	0.2	1.0	1.0	1.0	1.0	4.2	S54	0.2	1.0	1.0	1.0	0.0	3.2
S25	0.2	1.0	1.0	1.0	0.0	3.2	S55	0.2	1.0	1.0	1.0	0.0	3.2
S26	0.6	1.0	0.5	1.0	1.0	4.1	S56	1.0	1.0	1.0	1.0	1.0	5.0
S27	0.2	1.0	1.0	1.0	1.0	4.2	S57	0.8	1.0	1.0	1.0	1.0	4.8
S28	0.8	1.0	1.0	1.0	1.0	4.8	S58	0.8	1.0	1.0	1.0	0.0	3.8
S29	1.0	1.0	1.0	1.0	1.0	5.0	S59	0.6	1.0	1.0	1.0	1.0	4.6
S30	0.6	1.0	1.0	1.0	0.0	3.6	S60	1.0	1.0	1.0	1.0	1.0	5.0

	Q1	Q2	Q3	Q4	Q5	Total
Average score	0.59	0.93	0.95	1.00	0.70	4.17

content in SD. We may also use the definition of *conciseness* for ontology evaluation [32] to evaluate *conciseness* of e.g., ontology-based SDt [25].

(2) How knowledge-based approaches can improve design practice by better use of knowledge content in SD. For instance, architectural patterns and architecturally significant requirements do not receive much attention in the current applications of knowledge-based approaches in SDt (see Table 8). To further improve the practice of SDt activity, for example how KRv can be used to recover architecturally significant requirements in SAD, which can be made explicit in the current project or reused in other projects. A decision-centric approach is presented in [48] to recover design decisions and their semantically rich traceability links from architecturally significant requirements to architectural components, in which knowledge recovery is supported by machine learning techniques (e.g., classifier training). This approach may be employed and adapted to recover other knowledge content in SD.

(3) How to better apply SD. As shown in Table 7 and Table 8, we can see that much work has been done on using KCR to capture and represent knowledge in SD. However, the application of SD largely depends on the knowledge-based approaches: KR, KRt, KRs, and KS. The amount of research on applying SD knowledge is much less than SD knowledge capture and representation. This is to be expected since knowledge capture and representation is a prerequisite to using SD knowledge. The research community needs to focus more on how SD application can facilitate cost-effective software development with these knowledge-based approaches in order to make the most use of SD.

(4) How to measure the costs and benefits of using knowledge-based approaches in SDt in a qualitative or quantitative way. The cost and benefit categories identified in this SLR are classified without qualitative or quantitative comparison since most studies did not explicitly discuss this issue or provide such information. Hence we need more research on the qualitative or quantitative measurement of the costs and benefits of using knowledge-based approaches in SDt. For example, Dzidek et al. study the costs and benefits of using UML documentation in software maintenance [29]. They introduce six dependent variables (e.g., time, correctness, and design quality in maintenance tasks) to evaluate quantitatively and qualitatively the costs and benefits of using UML documentation (i.e., the treatment in controlled experiments). These dependent variables and extension of them can be potentially used for the measurement of the costs and benefits of using knowledge-based approaches (the treatment) in SDt.

5. Conclusions

Software documentation (SDt) is a core artifact as well as an important and prevalent activity in the software lifecycle [62], even in agile practices [59]. When SDt improves in quality, the software quality will improve too [55], but the costs and benefits of SDt determine how much documentation is needed [15,17,41]. Knowledge-based approaches have been extensively employed in software development for decades, as well as in SDt. In this work, we try to understand how knowledge-based approaches are used in SDt through a SLR. More specifically, the major objective of this SLR is to understand how knowledge-based approaches are employed in SDt, their influences, and the costs and benefits of

using knowledge-based approaches in SDt, especially in the context of architecture design.

Sixty studies on the review topic are finally included, in which twelve QA of SD, four cost categories, and nine benefit categories of using knowledge-based approaches in SDt are identified. Three categories of benefits out of the nine identified categories are achieved by using knowledge-based approaches in SDt: architecture understanding, requirements elicitation, and co-evolution of requirements and architecture. The cost of retrieving information from documents is the major concern when using knowledge-based approaches in SDt. The review results show that knowledge-based approaches are promising and appropriate to support the documentation from requirements to architecture.

In this review, we find an increasing trend in the number of studies on using knowledge-based approaches in SDt over the last decade. Among various knowledge-based approaches, KCR is the most frequently studied approach in SDt, which can be employed to improve all the twelve QA of SD. The usefulness of KRt and KRv in SDt is demonstrated, but these two approaches require more research work. The results of this SLR are also beneficial for practitioners. They can compare various knowledge-based approaches in their focused content, applications, and evidence levels in SDt, and then select or combine the approaches that are suitable for specific SDt (e.g., requirements or architecture documents) in their context.

Acknowledgements

This work has been partially sponsored by the Natural Science Foundation of China (NSFC) under the Grant No. 61170025, KeSRAD: Knowledge-enabled Software Requirements to Architecture Documentation and the Dutch "Regeling Kenniswerkers" Project KWR09164, "Stephenson: Architecture knowledge sharing practices in software product lines for print systems".

Appendix A. Primary studies in the review

[S1] A. Alti, A. Boukerram, A. Smeda, S. Maillard, M. Oussalah, COSABuilder and COSAInstantiator: an extensible tool for architectural description, International Journal of Software Engineering and Knowledge Engineering 20(3) (2010) 423–455.
[S2] E. Astesiano, G. Reggio, Knowledge structuring and representation in requirement specification, in: Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering (SEKE), 2002, pp. 143–150.
[S3] G. Buchageher, R. Weinreich, Automatic tracing of decisions to architecture and implementation, in: Proceedings of the 9th Working IEEE/IFIP Conference on Software Architecture (WICSA), 2011, pp. 46–55.
[S4] R. Capilla, J.C. Duenas, F. Nava, Viability for codifying and documenting architectural design decisions with tool support, Journal of Software Maintenance and Evolution: Research and Practice 22(2) (2010) 81–119.
[S5] M.C. Carignano, S. Gonnet, H. Leone, A model to represent architectural design rationale, in: Proceedings of the 7th Working IEEE/IFIP Conference on Software Architecture (WICSA), 2009, pp. 301–304.
[S6] M. Che, D.E. Perry, Scenario-based architectural design decisions documentation and evolution, in: Proceedings of the 18th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS), 2011, pp. 216–225.
[S7] W.C. Chu, C.H. Chang, C.W. Lu, Model-based object-oriented requirement engineering and its support to software documents integration, in: Proceedings of the 6th International Conference on Software Engineering Research and Practice (SERP), 2008, pp. 431–436.
[S8] A.L. Correa, C.M.L. Werner, Precise specification and validation of transactional business software, in: Proceedings of the 12th International Requirements Engineering Conference (RE), 2004, pp. 16–25.
[S9] R.C. de Boer, 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) (2008) 1456–1469.
[S10] R.C. de Boer, H. van Vliet, Writing and reading software documentation: How the development process may affect understanding, in: Proceedings of the 2nd ICSE Workshop on Cooperative and Human Aspects on Software Engineering (CHASE), 2009, pp. 40–48.
[S11] D. Falessi, G. Cantone, M. Becker, Documenting design decision rationale to improve individual and team design decision making: an experimental evaluation, in: Proceedings of the 5th ACM/IEEE International Symposium on Empirical Software Engineering (ISESE), 2006, pp. 134–143.
[S12] D. Falessi, G. Cantone, P. Kruchten, Value-based design decision rationale documentation: Principles and empirical feasibility study, in: Proceedings of the 7th Working IEEE/IFIP Conference on Software Architecture (WICSA), 2008, pp. 189–198.
[S13] R. Farenhorst, P. Lago, H. van Vliet, EAGLE: Effective tool support for sharing architectural knowledge, International Journal of Cooperative Information Systems 16(3&4) (2007) 413–437.
[S14] M. Feilkas, D. Ratiu, E. Jurgens, The loss of architectural knowledge during system evolution: an industrial case study, in: Proceedings of the 17th IEEE International Conference on Program Comprehension (ICPC), 2009, pp. 188–197.
[S15] R. Gacitua, P. Sawyer, V. Gervasi, On the effectiveness of abstraction identification in requirements engineering, in: Proceedings of the 18th International Requirements Engineering Conference (RE), 2010, pp. 5–14.
[S16] H.J. Happel, S. Seedorf, M. Schader, Ontology-enabled documentation of service-oriented architectures with ontobrowse semantic wiki, in: Proceedings of Innovation for Enterprise Software (PRIMIUM), 2009, pp. 61–80.
[S17] M.A. Hissen, Facilitating tacit-knowledge acquisition within requirements engineering, in: Proceedings of the 10th WSEAS International Conference on Applied Computer Science (ACS), 2010, pp. 27–32.
[S18] T. Holmes, H. Tran, U. Zdun, S. Dustdar, Model-driven and domain-specific architectural knowledge view for compliance meta-data in process-driven SOAs, in: Proceedings of the 5th Workshop on SHAring and Reusing Architectural Knowledge (SHARK), 2010, pp. 1–7.
[S19] A. Jansen, P. Avgeriou, J.S. van der Ven, Enriching software architecture documentation, Journal of System and Software 82(8) (2009) 1232–1248.
[S20] A. Jansen, J. Bosch, P. Avgeriou, Documenting after the fact: Recovering architectural design decisions. Journal of Systems and Software 81(4) (2008) 536–557.
[S21] A. Jansen, J. van der Ven, P. Avgeriou, D.K. Hammer, Tool support for architectural decisions, in: Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture (WICSA), 2007, pp. 4–13.
[S22] H. Kaiya, M. Saeki, Using domain ontology as domain knowledge for requirements elicitation, in: Proceedings of the 14th International Requirements Engineering Conference (RE), 2006, pp. 189–198.
[S23] H. Kaiya, Y. Shimizu, H. Yasui, K. Kaijiri, M. Saeki, Enhancing domain knowledge for requirements elicitation with web

mining, in: Proceedings of the 17th Asia–Pacific Software Engineering Conference (APSEC), 2010, pp. 3–12.
[S24] A.W. Kiwelekar, R.K. Joshi, Ontological analysis for generating baseline architectural descriptions, in: Proceedings of the 4th European Conference on Software Architecture (ECSA), 2010, pp. 417–424.
[S25] A. Kozlenkov, A. Zisman, Are their design specifications consistent with our requirements? in: Proceedings of the 10th International Requirements Engineering Conference (RE), 2002, pp. 145–154.
[S26] P. Kroha, R. Janetzko, J.E. Labra, Ontologies in checking for inconsistency of requirements specification, in: Proceedings of the 3rd International Conference on Advances in Semantic Processing (SEMAPRO), 2009, pp. 32–37.
[S27] P. Kruchten, P. Lago, H. van Vliet, Building up and reasoning about architectural knowledge, in: Proceedings of the 2nd International Conference on the Quality of Software Architectures (QoSA), 2006, pp. 43–58.
[S28] M. Kumar, N. Ajmeri, S. Ghaisas, Towards knowledge assisted agile requirements evolution, in: Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering (RSSE), 2010, pp. 16–20.
[S29] P. Lago, E. Niemela, H. van Vliet, Tool support for traceable product evolution, in: Proceedings of the 8th European Conference on Software Maintenance and Reengineering (CSMR), 2004, pp. 261–269.
[S30] Y. Li, N. Narayan, J. Helming, M. Koegel, A domain specific requirements model for scientific computing, in: Proceedings of the 33rd International Conference on Software Engineering (ICSE), 2011, pp. 848–851.
[S31] P. Liang, A. Jansen, P. Avgeriou, A. Tang, L. Xu, Advanced quality prediction model for software architectural knowledge sharing, Journal of Systems and Software 84(5) (2011) 786–802.
[S32] C. Lopez, L.M. Cysneiros, H. Astudillo, NDR ontology: Sharing and reusing NFR and design rationale knowledge, in: Proceedings of the 1st International Workshop on Managing Requirements Knowledge (MaRK), 2008, pp. 1–10.
[S33] M. Luckey, A. Baumann, D. Méndez, Reusing security requirements using an extended quality model, in: Proceedings of the 6th ICSE Workshop on Software Engineering for Secure Systems (SESS), 2010, pp. 1–7.
[S34] A. Marcus, J.I. Maletic, A. Sergeyev, Recovery of traceability links between software documentation and source code, International Journal of Software Engineering and Knowledge Engineering 15(5) (2005) 811–836.
[S35] A.B.B. Martínez, J.J.P. Arias, A.F. Vilas, On the interplay between inconsistency and incompleteness in multi-perspective requirements specifications, Information and Software Technology 50(4) (2008) 296–321.
[S36] C. McMillan, D. Poshyvanyk, M. Revelle, Combining textual and structural analysis of software artifacts for traceability link recovery, in: Proceedings of the 31st International Conference on Software Engineering (ICSE), 2009, pp. 41–48.
[S37] M. Meisinger, A. Rausch, M. Sihling, 4everedit – team-based process documentation management, Software Process: Improvement and Practice 11(6) (2006) 627–642.
[S38] M. Mirakhorli, J. Cleland-Huang, Transforming trace information in architectural documents into re-usable and effective traceability links, in: Proceedings of the 6th Workshop on SHAring and Reusing Architectural Knowledge (SHARK), 2011, pp. 45–52.
[S39] K. Mohan, P. Xu, L. Cao, B. Ramesh, Improving change management in software development: Integrating traceability and software configuration management, Decision Support Systems 45(4) (2008) 922–936.
[S40] B. Orlic, R. Mak, I. David, J. Lukkien, Concepts and diagram elements for architectural knowledge management, in: Proceedings of the 5th European Conference on Software Architecture (ECSA), 2011, pp. 1–10.
[S41] A. Osada, D. Ozawa, H. Kaiya, K. Kaijiri, The role of domain knowledge representation in requirements elicitation, in: Proceedings of the 25th IASTED International Multi-Conference: Software Engineering (SE), 2007, pp. 84–92.
[S42] J.A.D. Pace, J.P. Carlino, M. Blech, A. Soria, M.R. Campo, Assisting the synchronization of UCM-based architectural documentation with implementation, in: Proceedings of the 7th Working IEEE/IFIP Conference on Software Architecture (WICSA), 2009, pp. 151–160.
[S43] D. Rambabu, T.V. Prabhakar, On archiving architecture documents, in: Proceedings of the 12th Asia–Pacific Software Engineering Conference (APSEC), 2005, pp. 351–358.
[S44] R. Rauf, M. Antkiewicz, K. Czarnecki, Logical structure extraction from software requirements documents, in: Proceedings of the 12th International Requirements Engineering Conference (RE), 2011, pp. 101–110.
[S45] E. Rubin, H. Rubin, Supporting agile software development through active documentation, Requirements Engineering 16(2) (2011) 117–132.
[S46] J. Savolainen, T. Mannisto, Conflict-centric software architectural views: Exposing trade-offs in quality requirements, IEEE Software 27(6) (2010) 33–37.
[S47] H.H. Schoonewille, W. Heijstek, R.V. Michel, K. Thomas, A cognitive perspective on developer comprehension of software design documentation, in: Proceedings of the 30th ACM International Conference on Design of Communication (SIGDOC), 2011, pp. 211–218.
[S48] M. Shahin, P. Liang, Z.Y. Li, Architectural design decision visualization for architecture design: Preliminary results of A controlled experiment, in: Proceedings of the 4th European Conference on Software Architecture: Companion Volume (ECSA), 2011.
[S49] E. Sharifi, R.A. Moghadam, F. Bobillo, M.M. Ebadzadeh, A fuzzy framework for semantic web service description, matchmaking, ranking and selection, in: Proceedings of the 8th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD), 2011, pp. 621–625.
[S50] R. Sharma, and K.K. Biswas, Using courteous logic based representations for requirements specification, in: Proceedings of the 4th International Workshop on Managing Requirements Knowledge (MaRK), 2011, pp. 12–16.
[S51] M.T. Su, J. Hosking, 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), 2011, pp. 256–259.
[S52] A. Tang, Y. Jin, J. Han, A rationale-based architecture model for design traceability and reasoning, Journal of Systems and Software 80(6) (2007) 918–934.
[S53] A. Tang, P. Liang, V. Clerc, H. van Vliet, Traceability in the co-evolution of architectural requirements and design, in: Relating Software Requirements and Architectures, Springer, 2011, pp. 35–60.
[S54] A. Tang, P. Liang, H. van Vliet, Software architecture documentation: the road ahead, in: Proceedings of the 9th Working IEEE/IFIP Conference on Software Architecture (WICSA), 2011, pp. 252–255.
[S55] J.T.E. Timm, G.C. Gannod, Specifying semantic web service compositions using UML and OCL, in: Proceedings of the 14th International Conference of the Web Services (ICWS), 2007, pp. 521–528.
[S56] J.J.P. Tsai, A. Liu, Experience on knowledge-based software engineering: a logic-based requirements language and its

industrial applications, Journal of Systems and Software 82(10) (2009) 1578–1587.
[S57] U. van Heesch, P. Avgeriou, R. Hilliard, A documentation framework for architecture decisions, Journal of Systems and Software 85(4) (2011) 795–820.
[S58] R. Weinreich, G. Buchgeher, Towards supporting the software architecture life cycle, Journal of Systems and Software 85(3) (2011) 546–561.
[S59] D. Weyns, B. Michalik, Codifying architecture knowledge to support online evolution of software product lines, in: Proceedings of the 6th Workshop on SHAring and Reusing architectural Knowledge (SHARK), 2011, pp. 37–44.
[S60] Y. Zhang, R. Witte, J. Rilling, V. Haarslev, Ontological approach for the semantic recovery of traceability links between software artifacts, IET Software 2(3) (2008) 185–203.

Appendix B. Abbreviations used in the review

KCR	Knowledge capture and representation
KM	Knowledge management
KR	Knowledge reuse
KRs	Knowledge reasoning
KRt	Knowledge retrieval
KRv	Knowledge recovery
KS	Knowledge sharing
PICO	Population, intervention, comparison, and outcome
QA	Quality attribute(s)
RQ	Research question
SAD	Software architecture document(s)
SD	Software document(s)
SDt	Software documentation
SE	Software engineering
SLR	Systematic literature review
SRD	Software requirements document(s)

References

[1] IEEE, IEEE Std. 830-1984, Guide to Software Requirement Specifications, 1984.
[2] IEEE, IEEE Std. 830-1998, IEEE Recommended Practice for Software Requirements Specifications, 1998.
[3] IEEE, IEEE Std. 1016-1998, Recommended Practice for Software Design Description, 1998.
[4] IEEE, IEEE Std. 1471-2000, Recommended Practice for Architectural Description of Software Intensive Systems, 2000.
[5] IEEE, Guide to the Software Engineering Body of Knowledge (SWEBOK), IEEE Computer Society, 2004.
[6] ISO, ISO 9000-3:1991, Quality Management and Quality Assurance Standards – Part 3: Guidelines for the Application of ISO 9001 to the Development, Supply and Maintenance of Software, International Organization for Standardization, Geneva, Switzerland, 1991.
[7] V.S. Alagar, K. Periyasamy, Specification activities, in: Specification of Software Systems, second ed., Springer, New York, 2011, pp. 23–34.
[8] M. Alavi, D.E. Leidner, Review: knowledge management and knowledge management systems: conceptual foundations and research issues, MIS Quart. 25 (1) (2001) 107–136.
[9] M.S. Ali, M.A. Babar, L. Chen, K.J. Stol, A systematic review of comparative evidence of aspect-oriented programming, Inf. Softw. Technol. 52 (9) (2010) 871–887.
[10] V. Alves, N. Niu, C. Alves, G. Valenca, Requirements engineering for software product lines: a systematic literature review, Inf. Softw. Technol. 52 (8) (2010) 806–820.
[11] O. Barais, A.F. Le Meur, L. Duchien, J. Lawall, Software architecture evolution, in: Software Evolution, Springer-Verlag, Berlin, Heidelberg, 2008, pp. 233–262.
[12] T.T. Barker, Writing Software Documentation: A Task-Oriented Approach, second ed., Allyn and Bacon, 2003.
[13] Z.A. Barmi, A.H. Ebrahimi, R. Feldt, Alignment of requirements specification and testing: a systematic mapping study, in: Proceedings of the 4th International Conference on Software Testing, Verification and Validation Workshops (ICSTW), Berlin, Germany, 2011, pp. 476–485.
[14] M. Biehl, Literature Study on Design Rationale and Design Decision Documentation for Architecture Descriptions, Technical Report ISRN/KTH/MMK/R-10/06-SE, Royal Institute of Technology, Stockholm, Sweden, 2010.
[15] D.C. Blair, M.E. Maron, An evaluation of retrieval effectiveness for a full-text document-retrieval system, Commun. ACM 28 (3) (1985) 289–299.
[16] L.C. Briand, On the many ways software engineering can benefit from knowledge engineering, in: Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering (SEKE), Ischia, Italy, 2002, pp. 3–6.
[17] L.C. Briand, Software documentation: how much is enough, in: Proceedings of the 17th European Conference on Software Maintenance and Reengineering (CSMR), Benevento, Italy, 2003, pp. 13–15.
[18] L. Chen, M.A. Babar, H. Zhang, Towards an evidence-based understanding of electronic data sources, in: Proceedings of the 14th International Conference on Evaluation and Assessment in Software Engineering (EASE), Keele, UK, 2010, pp. 135–138.
[19] J. Cleland-Huang, O. Gotel, A. Zisman, Software and Systems Traceability, Springer, London, 2012.
[20] P. Clements, F. Bachmann, L. Bass, D. Garlan, Documenting Software Architecutre: Views and Beyond, second ed., Addison-Wesley Professional, 2010.
[21] R. Collobert, J. Weston, A unified architecture for natural language processing: deep neural networks with multitask learning, in: Proceedings of the 25th International Conference on Machine Learning (ICML), Helsinki, Finland, 2008, pp. 160–167.
[22] A. Davis, S. Overmyer, K. Jordan, J. Caruso, Identifying and measuring quality in a software requirements specification, in: Proceedings of the 1st International Software Metrics Symposium (METRICS), Baltimore, MD, USA, 1993, pp. 141–152.
[23] A. Davis, Software Requirments: Objects, Functions, and State, second ed., Prentice Hall, Englewood Cliffs, NJ, 1993.
[24] R.C. de Boer, R. Farenhorst, P. Lago, H. van vliet, V. Clerc, A. Jansen, Architectural knowledge: getting to the core, in: Proceedings of the 3rd International Conference on the Quality of Software Architectures (QoSA), Medford, USA, 2007, pp. 197–214.
[25] K.A. de Graaf, A. Tang, P. Liang, H. van Vliet, Ontology-based software architecture documentation, in: Proceedings of the Joint 10th Working IEEE/IFIP Conference on Software Architecture & 6th European Conference on Software Architecture (WICSA/ECSA), Helsinki, Finland, 2012, pp. 121–130.
[26] P. Devanbu, R. Brachman, P.G. Selfridge, B.W. Ballard, LaSSIE: a knowledge-based software information system, Commun. ACM 34 (5) (1991) 34–49.
[27] T. Dingsoyr, R. Conradi, A survey of case studies of the use of knowledge management in software engineering, Int. J. Softw. Eng. Knowl. Eng. 12 (4) (2002) 391–414.
[28] T. Dybå, T. Dingsøyr, Empirical studies of agile software development: a systematic review, Inf. Softw. Technol. 50 (9) (2008) 833–859.
[29] W.J. Dzidek, E. Arisholm, L.C. Briand, A realistic empirical evaluation of the costs and benefits of UML in software maintenance, IEEE Trans. Softw. Eng. 34 (3) (2008) 407–432.
[30] S. Easterbrook, J. Singer, M.A. Storey, D. Damian, Selecting empirical methods for software engineering research, in: Guide to Advanced Empirical Software Engineering, Springer, London, UK, 2008, pp. 285–311.
[31] A. Forward, T.C. Lethbridge, The relevance of software documentation, tools and technologies: a survey, in: Proceedings of the 2nd ACM Symposium on Document Engineering (DocEng), McLean, Virginia, 2002, pp. 26–33.
[32] A. Gómez-Pérez, Evaluation of ontologies, Int. J. Intell. Syst. 16 (3) (2001) 391–409.
[33] O.C.Z. Gotel, A.C.W. Finkelstein, An analysis of the requirements traceability problem, in: Proceedings of the 1st International Conference on Requirements Engineering (RE), London, UK, 1994, pp. 94–101.
[34] S.J. Greenspan, On the role of domain knowledge-based approaches to software development, ACM SIGSOFT Softw. Eng. Notes 11 (4) (1986) 34–35.
[35] S. Haiduc, G. Bavota, R. Oliveto, A. Marcus, A. de Lucia, 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, 2012, pp. 1273–1276.
[36] A. Jansen, P. Avgeriou, J.S. van der Ven, Enriching software architecture documentation, J. Syst. Softw. 82 (8) (2009) 1232–1248.
[37] B. Kitchenham, S. Charters, Guidelines for Performing Systematic Literature Reviews in Software Engineering, EBSE Technical Report EBSE-2007-01, Keele University & University of Durham, 2007.
[38] R.E. Kraut, L.A. Streeter, Coordination in software development, Commun. ACM 38 (3) (1995) 69–81.
[39] P. Kruchten, Documentation of software architecture from a knowledge management perspective – design representation, in: Software Architecture Knowledge Management, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 39–57.
[40] P. Kruchten, P. Lago, H. van Vliet, Building up and reasoning about architectural knowledge, in: Proceedings of the 2nd International Conference on the Quality of Software Architectures (QoSA), Berlin, Germany, 2006, pp. 43–58.
[41] T.C. Lethbridge, J. Singer, A. Forward, How software engineers use documentation: the state of the practice, IEEE Softw. 20 (6) (2003) 35–39.
[42] Z. Li, P. Liang, P. Avgeriou, Application of knowledge-based approaches in software architecture: a systematic mapping study, Inf. Softw. Technol. 55 (5) (2013) 777–794.
[43] P. Liang, P. Avgeriou, From Architectural Knowledge to Requirements Knowledge Management, Technical Report RUG-SEARCH-09-L02, SEARCH, University of Groningen, February, 2009.
[44] P. Liang, P. Avgeriou, K. He, Rationale management challenges in requirements engineering, in: Proceedings of the 3rd International Workshop on Managing Requirements Knowledge (MaRK), Sydney, Australia, 2010, pp. 16–21.

[45] M. Luckey, A. Baumann, D. Méndez, Reusing security requirements using an extended quality model, in: Proceedings of the 6th ICSE Workshop on Software Engineering for Secure Systems (SESS), Cape Town, South Africa, 2010, pp. 1–7.
[46] W. Maalej, M.P. Robillard, Patterns of knowledge in API reference documentation, IEEE Trans. Softw. Eng. 39 (9) (2013) 1264–1282.
[47] N. Maiden, Cherishing ambiguity, IEEE Softw. 29 (6) (2012) 16–17.
[48] M. Mirakhorli, Tracing architecturally significant requirements: a decision-centric approach, in: Proceedings of the 33rd International Conference on Software Engineering (ICSE), Hawaii, USA, 2011, pp. 1126–1127.
[49] M. Mirakhorli, J. Cleland-Huang, Transforming trace information in architectural documents into re-usable and effective traceability links, in: Proceedings of the 6th International Workshop on SHAring and Reusing Architectural Knowledge (SHARK), Hawaii, USA, 2011, pp. 45–52.
[50] J. Mylopoulos, Conceptual modeling and telos, in: Conceptual Modeling, Databases and CASE: An Integrated View of Information Systems Development, John Wiley & Sons Inc., New York, USA, 1992, pp. 49–68.
[51] E.Y. Nakagawa, D. Feitosa, K.R. Felizardo, Using systematic mapping to explore software architecture knowledge, in: Proceedings of the 5th Workshop on SHAring and Reusing Architectural Knowledge (SHARK), Cape Town, South Africa, 2010, pp. 29–36.
[52] J. Nicolás, A. Toval, On the generation of requirements specifications from software engineering models: a systematic literature review, Inf. Softw. Technol. 51 (9) (2009) 1291–1307.
[53] I. Nonaka, H. Takeuchi, The Knowledge-Creating Company: How Japanese Companies Create the Dynamics of Innovation, Oxford University Press, 1995.
[54] D.L. Parnas, Document based rational software development, Knowl.-Based Syst. 22 (3) (2009) 132–141.
[55] D.L. Parnas, Precise documentation: the key to better software, in: The Future of Software Engineering, Springer, Zürich, Switzerland, 2011, pp. 125–148.
[56] M.C. Paulk, The Capability Maturity Model: Guidelines for Improving the Software Process, Addison-Wesley, 1995.
[57] K. Petersen, R. Feldt, S. Mujtaba, M. Mattsson, Systematic mapping studies in software engineering, in: Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering (EASE), Bari, Italy, 2008, pp. 68–77.
[58] L. Reeve, H. Han, Survey of semantic annotation platforms, in: Proceedings of the 20th ACM Symposium on Applied Computing (SAC), New York, USA, 2005, pp. 1634–1638.
[59] E. Rubin, H. Rubin, Supporting agile software development through active documentation, Requirements Eng. 16 (2) (2011) 117–132.
[60] I. Rus, M. Lindvall, Knowledge management in software engineering, IEEE Softw. 19 (3) (2002) 26–38.
[61] M. Shahin, P. Liang, M.R. Khayyambashi, Architectural design decision: Existing models and tools, in: Proceedings of the Joint 8th Working IEEE/IFIP Conference on Software Architecture & 3rd European Conference on Software Architecture (WICSA/ECSA), Cambridge, UK, 2009, pp. 293–296.
[62] I. Sommerville, Software documentation, Software Engineering: The Supporting Processes, vol. 2, Wiley-IEEE Press, New York, USA, 2002, pp. 171–186.
[63] G. Stark, P. Oman, A. Skillicorn, C.R. Ameele, An examination of the effects of requirements changes on software maintenance releases, J. Softw. Maint.: Res. Pract. 11 (5) (1999) 293–309.
[64] M.T. Su, C. Hirsch, J. Hosking, KaitoroBase: Visual exploration of software architecture documents, in: Proceedings of the 24th IEEE/ACM International Conference on Automated Software Engineering (ASE), Auckland, New Zealand, 2009, pp. 657–659.
[65] A. Tang, P. Avgeriou, A. Jansen, R. Capilla, M.A. Babar, A comparative study of architecture knowledge management tools, J. Syst. Softw. 83 (3) (2010) 352–370.
[66] A. Tang, M.A. Babar, I. Gorton, J. Han, A survey of the use and documentation of architecture design rationale, in: Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture (WICSA), Pittsburgh, Pennsylvania, USA, 2005, pp. 89–98.
[67] A. Tang, Y. Jin, J. Han, A rationale-based architecture model for design traceability and reasoning, J. Syst. Softw. 80 (6) (2007) 918–934.
[68] A. Tang, P. Liang, H. van Vliet, Software architecture documentation: the road ahead, in: Proceedings of the 9th Working IEEE/IFIP Conference on Software Architecture (WICSA), Boulder, Colorado, USA, 2011, pp. 252–255.
[69] A. Tang, H. van Vliet, Modeling constraints improves software architecture design reasoning, in: Proceedings of the Joint 8th Working IEEE/IFIP Conference on Software Architecture & 3rd European Conference on Software Architecture (WICSA/ECSA), Cambridge, UK, 2009, pp. 253–256.
[70] J.S. van der Ven, A. Jansen, P. Avgeriou, D.K. Hammer, Using architectural decisions, in: Proceedings of the 2nd International Conference on the Quality of Software Architectures (QoSA), Västeras, Sweden, 2006, pp. 1–10.
[71] H. van Vliet, Software architecture knowledge management, in: Proceedings of the 19th Australian Conference on Software Engineering (ASWEC), Perth, WA, Australia, 2008, pp. 24–31.
[72] H. van Vliet, Knowledge sharing in software development, in: Proceedings of the 10th International Conference on Quality Software (QSIC), Zhangjiajie, China, 2010, p. 2-2.
[73] C. Wohlin, P. Runeson, M. Host, M.C. Ohlsson, B. Regnell, A. Wesslen, Experimentation in Software Engineering, Springer-Verlag, Berlin, Heidelberg, 2012.