Journal of Software: Evolution and Process

JOURNAL OF SOFTWARE: EVOLUTION AND PROCESS
J. Softw. Evol. and Proc. 2014; **26**:654–691
Published online 12 February 2014 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/smr.1643

Classification and comparison of architecture evolution reuse knowledge—a systematic review

Aakash Ahmad[1,2], Pooyan Jamshidi[1,2,3,*,†] and Claus Pahl[1,2,3]

[1]*School of Computing, Dublin City University, Dublin, Ireland*
[2]*Lero—the Irish Software Engineering Research Centre, Ireland*
[3]*Irish Centre for Cloud Computing and Commerce (IC4), Ireland*

ABSTRACT

Context: Architecture-centric software evolution (ACSE) enables changes in system's structure and behaviour while maintaining a global view of the software to address evolution-centric trade-offs. The existing research and practices for ACSE primarily focus on *design-time evolution* and *runtime adaptations* to accommodate changing requirements in existing architectures.

Objectives: We aim to *identify*, taxonomically *classify* and systematically *compare* the existing research focused on enabling or enhancing change reuse to support ACSE.

Method: We conducted a systematic literature review of 32 qualitatively selected studies and taxonomically classified these studies based on solutions that enable (i) *empirical acquisition* and (ii) *systematic application* of architecture evolution reuse knowledge (AERK) to guide ACSE.

Results: We identified six distinct research themes that support acquisition and application of AERK. We investigated (i) *how* evolution reuse knowledge is defined, classified and represented in the existing research to support ACSE and (ii) *what* are the existing methods, techniques and solutions to support empirical acquisition and systematic application of AERK.

Conclusions: *Change patterns* (34% of selected studies) represent a predominant solution, followed by *evolution styles* (25%) and *adaptation strategies and policies* (22%) to enable application of reuse knowledge. Empirical methods for acquisition of reuse knowledge represent 19% including *pattern discovery*, *configuration analysis*, *evolution and maintenance prediction* techniques (approximately 6% each). A lack of focus on empirical acquisition of reuse knowledge suggests the need of solutions with *architecture change mining* as a complementary and integrated phase for *architecture change execution*. Copyright © 2014 John Wiley & Sons, Ltd.

Received 13 May 2013; Revised 23 September 2013; Accepted 27 December 2013

KEY WORDS: software architecture; architecture-centric software evolution; architecture evolution reuse knowledge; systematic literature review; evidence-based study in software evolution; research synthesis

1. INTRODUCTION

Modern software systems operate in a dynamic environment with frequent changes in stakeholder needs, business and technical requirements and operating environments [1, 2]. These changing requirements trigger a continuous evolution in existing software to prolong its productive life and economic value over time [1, 3]. During the design, development and evolution of software systems, the role of an architecture

*Correspondence to: Pooyan Jamshidi, School of Computing, Dublin City University, Dublin, Ireland.
†E-mail: pooyan.jamshidi@computing.dcu.ie

as a blueprint of software is central to map the changes in requirements [4] and their implementations in source code [5]. Architecture abstracts the implementation specific details of a software by modelling lines-of-code as architectural components and their interconnections. As a result, an architecture model enables planning, modelling and executing both design-time evolution [3, 6] and runtime adaptation [7, 8]—at higher abstraction levels such as software components and connectors [3, 7, 9, 10].

Lehman's law of continuing change [2] poses a challenge to research and practices that aim to support long-living and continuously evolving architectures [7, 11, 12] under frequently varying requirements [4, 10]. The law states, 'systems must be continually adapted or they become progressively less satisfactory'. To support a continuous change [2], existing solutions focused on exploiting reusable knowledge and expertise to address recurring evolution [13] and adaptation [12] of software architectures. However, there has been no attempt to analyse the existing research with a systematic study of active trends, limitations and future dimensions for evolution reuse in software architectures [9, 14]. Furthermore, considering the growing demand for autonomic computing [15, 16] or specifically self-adaptive architectures [6, 11, 12], we must distinguish the effects of reuse on *design-time* [3] (also static or off-line) as well as on *runtime* (also dynamic adaptation or online) evolution [7].

Recently, we conducted a systematic review [9] to classify and compare state of the research and practices that enable architecture-centric software evolution (ACSE). An evaluation of this review suggested,

> given the increasing importance of reuse in ACSE, a dedicated effort is required to systematically classify and compare available evidences that support reuse in evolution and adaptation to address architecture-based change management.

Existing studies of architecture evolution research are focused on analysing [14], characterising [10] and comparing [9, 6] ACSE approaches. In contrast to the existing reviews on architectural evolution [6, 9, 10, 14], our focus in this review is to classify and compare research that enables acquisition and application of reuse knowledge to support ACSE.

In recent years, interest in the area of *architecture knowledge* (AK) research [17] has grown—in books [18], research conferences [19], workshops [20] and dedicated body of knowledge [21]. Although *architecture evolution reuse knowledge* (AERK)[‡] could be classified as a sub-domain of AK, a survey of AK research [17] identifies architectural maintenance and evolution as an independent concern. This allows us to conclude that in the general context of AK, there is a need to explicitly classify and compare research on evolution knowledge to address recurring evolution in architectures [12, 13, 22]. Thus, we shift architectural knowledge application focus from the reuse of design-time artefacts to the *reuse of evolution-centric artefacts*, although the progress of architecture evolution reuse research [13, 22, 23] is reflected over more than a decade starting in 2001 [23]. However, we did not find any evidence to systematically synthesise the collective impact of existing research focused on AERK.

To carry out this review, we followed the guidelines in [24] to conduct a systematic literature review (SLR) of evolution reuse in architectures. SLRs help to identify, classify and synthesise a comparative overview of state-of-the-art research and enable knowledge transfer among the research community. The objective of this research is to *systematically identify and classify the available evidence about evolution reuse in software architectures and provide a comparison of existing research to highlight its potential, limitations and future dimensions*.

This SLR includes 32 qualitatively selected studies that are classified as research that supports *acquisition* or *application* of reuse knowledge to support ACSE. To assess the contribution of each study, we provide a comparison among all the studies and synthesise our results using 12 *comparison attributes*. The comparison attributes are derived and refined by following the guidelines in [10, 14, 25], our experience with SLRs [9, 26], a qualitative assessment of the studies and an external validation of the review protocol.

In Figure 1, a classification of existing literature highlights primary contributions of this paper, focussing on the following:

[‡]Please note that we use the terms 'AERK' and 'evolution reuse knowledge' and 'reuse knowledge' interchangeably—all referring to the same concept.

DOI: 10.1002/smr

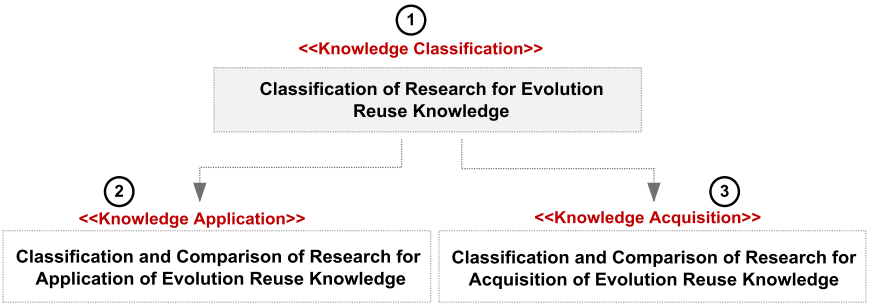

Figure 1. An overview of the contribution of systematic review.

- *How* AERK is defined, classified and expressed in existing research to support ACSE. The contribution is a *taxonomic classification* scheme to identify and categorise research with overlapping and disjoint themes on evolution reuse.
- *What* existing methods and techniques enable or enhance evolution reuse in software architectures. We classify and compare the state of research and analyse the research impact based on 26 (81%) of qualitatively assessed studies.
- *What* existing methods and techniques enable an empirical acquisition of evolution reuse knowledge. We analyse the role of existing methodologies to *discover* and *share* evolution *reuse* based on six (19%) selected studies.

We identified three distinct research themes that enable reuse in architecture evolution. *Change patterns* [4, 27], as the most prominent solution address corrective, perfective and adaptive changes [10] for design-time evolution [3, 13] as well as runtime adaptations [7, 12]. In contrast, *evolution styles* [13, 22] only support design-time evolution as corrective and perfective type changes [10], whereas *adaptation strategies and policies* [12, 28] enable self-adaptation in running architectures. In general, we observed non-complementary and solution-specific representation and expression of AERK. In the knowledge acquisition context, we identified three research themes—*pattern discovery* [29, 30], *configuration analysis* [23, 31] and *evolution and maintenance prediction* [32, 33]. We observed a lack of research on empirical approaches to analyse and discover knowledge [30, 32, 33] that can be shared and reused to guide ACSE.

On the basis of a taxonomic classification of studies, we provide a definition of AERK. We propose a framework, REVOLVE, that supports architecture *change mining* (for reuse knowledge acquisition) as a complementary and integrated phase to *architecture change execution* (for reuse knowledge application). This framework guides the systematic review. The literature base we provide in [34] is itself subject to a continuous evolution (adding newly published studies over time) and helps in knowledge sharing with ACSE community [35, 36]. In particular, the results of this SLR are beneficial for the following:

- Researchers in software engineering and software architecture in particular, who require an identification of relevant studies. A systematic presentation of research provides a foundational body of knowledge to develop theory and solutions, analyse research implications and to establish future dimensions.
- Practitioners interested in understanding the methods and solutions with formalism and tool support to model, analyse and implement evolution reuse in software architectures.

In general, this SLR provides a literature base to identify emerging trends or formulating hypotheses as a complement to existing studies [6, 9, 10, 14, 17]. The collected data in [34]—as an online literature base—provides a detailed insight and objective interpretation of the results.

The remainder of this paper is organised as follows. Section 2 presents background details and related research. Section 3 describes details about the research methodology we followed to plan, conduct and document the SLR. Section 4 highlights the results based on a taxonomical classification of the literature. Sections 5 and 6 present two separate concerns—application and acquisition of reuse knowledge, respectively. Research implications and validity threats are discussed in Section 7 with conclusions in Section 8.

DOI: 10.1002/smr

2. BACKGROUND

Architectural maintenance [32], evolution [3, 22] and adaptation [12] represent different views of change implementation determined by the types, means, times and frequency of changes in software architectures. We highlight existing secondary studies in the context of ACSE that justifies the needs and scope of this review. In contrast to the existing systematic reviews on ACSE [6, 9, 10, 14, 17], this SLR specifically focuses on a taxonomical classification and comparison of research that supports evolution reuse in architectures.

2.1. Architecture-centric software maintenance and evolution

The implications of software maintenance and evolution in the context of system life cycle became obvious with the emergence of Lehman's laws of software evolution [2] and the International Organization for Standardization/ International Electrotechnical Commission (ISO/IEC) 14764 standard for software maintenance [37]. Since then, maintenance and evolution represent a critical activity in system life cycle to prolong the productive life, economic value and operational reliability of existing software [1, 2, 25]. However, beyond these abstract laws and theoretical standardisations, a critical decision is to select an appropriate abstraction to implement changes in software [3, 13, 22]. In contrast to source-code refactoring [5], architecture models—as topological configurations of components and their connectors—represent an appropriate abstraction of software to enable maintenance and evolution in a controllable fashion [3, 9, 14]. The software engineering literature in general and theory of software architectures in particular treated maintenance and evolution as virtually synonymous, interchangeable concepts [1, 8]. However, in this review, we must maintain a distinction between the two based on the time of change implementation. More specifically, *architectural maintenance* refers to post-deployment changes implemented as static or off-line modifications of architecture. In contrast, *architectural evolution* refers to consequential changes in architectures usually implemented as dynamic or online modifications of architecture. Furthermore, in order to consider the needs for autonomic computing [15] and self-adaptive architectures [6, 7, 11], we must distinguish between design-time maintenance [32, 33] and runtime evolution or adaptations [12, 28]. In the taxonomy of software change [25], the factors influencing evolution are the following:

- *Time of evolution*: To operate in a dynamic and open world [16], modern software systems need to evolve their architecture while maintaining system execution. This highlights a critical factor as a change (either design-time or runtime evolution) that must be implemented in a timely and consistent fashion [12].
- *Frequency of change*: It determines the rate at which software must evolve in order to keep its utility [2]. Therefore, evolution reuse knowledge could provide assistance to effectively address frequent (business and technical) change cycles in architecture of software systems.
- *Evolution reuse*: To support a frequent evolution and adaptation in a timely fashion, solutions must follow 'build-once, use-often' philosophy to support reuse of recurring architectural changes [4, 31]. In recent years, solution for architectural evolution promoted evolution styles [13, 22] to enable change reuse. However, systematic reviews in ACSE [9, 10, 14, 17] suggest the needs for solutions that enable a continuous empirical *discovery* of reuse knowledge that can be *shared* and *reused* to enable or enhance ACSE.

2.2. Secondary studies on software architecture evolution

In recent years, the SLRs on ACSE have focused on architecture evolution analysis [14], characterisation of architectural changes [10] and classification and comparison of architecture evolution research [9, 6]. We summarise the existing SLRs [9, 10, 14] (in Section 2.2.1) and survey-based studies [6] (in Section 2.2.2) to justify the needs and scope for this review (in Section 2.2.3).

2.2.1. Systematic literature reviews of software architecture evolution.

A. *Review of architecture change characterisation*—A systematic review (Williams and Carver [10] in Table I) investigated a total of 130 peer-reviewed studies—published from 1976 to 2008—to characterise design-time and runtime evolution as corrective, perfective, adaptive and preventive type changes in architectures. The SLR [10] proposed a comprehensive change

DOI: 10.1002/smr

Table I. A summary of secondary studies on architecture-centric software evolution.

Study type	Study reference	Study focus	Year of publication	Time constraints	Total reviewed	Years of studies
Systematic literature review(s)	Williams and Carver [10]	Change characterisation	2010	Design-time, runtime	130	1976–2008
	Breivold *et al.* [14]	Evolvability analysis	2011	Design-time	82	1992–2010
	Jamshidi *et al.* [9]	Classification and comparison	2013	Design-time, runtime	60	1995–2011
	Ahmad and Jamshidi [34]	Reuse-driven evolution	N/A	Design-time, runtime	32	1999–2012
Surveys	Bradbury *et al.* [6]	Dynamic evolution	2004	Runtime	14	1992–2002
Mapping studies	Li *et al.* [17]	Architecture knowledge	2013	N/A	55	2000–2011

characterisation scheme to systematically classify different approaches on how to distinguish and characterise software architecture changes and change impact analysis. The scheme works as a decision tree to provide support for system developers to assess the impact and feasibility of desired changes.

B. *Review of architecture evolution analysis*—A systematic review (Breivold *et al.* [14] in Table I) investigates 82 peer-reviewed studies—published from 1992 to 2012—focused on design-time evolution of software architectures. The SLR in [14] is focused on analysing the evolvability of a software architecture. The primary objective of this review is to provide an overview of existing approaches for analysing and improving software architecture evolution and to identify critical factors influencing software architecture evolvability.

C. *Classification and comparison of ACSE research*—We conducted a systematic review (Jamshidi *et al.* [9] in Table I) of 60 peer-reviewed studies—published from 1995 to 2011—focused on design-time and runtime evolution of software architectures. In the SLR [9], we qualitatively investigated the state of the art to classify and compare of formalisms and tool support that enable or enhance software architecture evolution.

2.2.2. Survey-based and taxonomic studies on software architecture evolution.

A. *Survey of self-management in dynamic software*—A survey-based study (Bradbury *et al.* [6] in Table I) reviewed 14 studies—published from 1992 to 2002—focused on runtime evolution of software architectures. The survey [6] synthesises formal specifications for dynamic adaptation of software architectures. The authors present a set of classification criteria for the comparison of dynamic software architectures based on the types, processes and infrastructure for dynamic adaptation of architectures.

B. *Mapping study on knowledge-based approaches in software architectures*—A mapping study (Li *et al.* [17] in Table I) provides a systematic mapping of research on knowledge-based approaches in software architecture according to 55 peer-reviewed studies—published from 2000 to 2011. The mapping study [17] identifies gaps in the application of knowledge-based approaches to five architecting activities that include architectural *analysis*, *synthesis*, *evaluation*, *implementation*, and *maintenance and evolution*. The study shows an increasing interest in the application of knowledge-based approaches in software architecture with only 5/55 studies on architectural knowledge for maintenance and evolution.

C. *Industrial survey and taxonomic study on architecture evolution*—Stammel *et al.* [35] provided an overview of various approaches evaluated based on real-world industrial scenarios on the evolution of sustainable systems. The study targets practitioners because it is a general and live document based on a growing number of industrial experience reports. Slyngstad *et al.* [38] performed a survey among software architects from software industry in order to capture a more complete picture of risk and management issues in software architecture evolution.

DOI: 10.1002/smr

Although not directly related to the ACSE, some taxonomies of software change [25, 8] try to answer the questions like why, how, what, when and where aspects of software evolution that have also acted as a guideline for us to define the comparison attributes (detailed in Section 3).

2.3. *A systematic review of architecture evolution reuse knowledge*

The review in this paper (Ahmad and Jamshidi [34] in Table I) is focused on a systematic *identification*, *classification* and *comparison* of the existing research that supports application and acquisition of reuse knowledge to support ACSE. In contrast to the mapping study on AK [35] that identifies only five studies on design-time maintenance and evolution, our SLR is comprised of 32 studies published from 1999 to 2011 and is focused on both design-time and runtime evolution. As presented in Table I, the proposed SLR complements the existing body of secondary studies on ACSE [6, 10, 14] and extends our previous review [9]. Given the importance of reuse in ACSE [14], it exclusively focuses on classification and comparison of evolution reuse knowledge.

In order to ensure that a similar review or any study has not already been performed, we searched the Compendex, IEEE Xplore, ACM and Google Scholar digital libraries (on 23/10/2012). None of the retrieved publications were related to any of our research questions detailed in Section 3. Considering the importance of reuse in ACSE [9, 14] and the relative maturity of AK approaches [18, 21], a consolidation of existing evidence about application and acquisition of reuse knowledge to support ACSE is timely.

3. RESEARCH METHODOLOGY

In contrast to a non-structured review process, a SLR [24, 39] reduces bias by following a precise and rigorous sequence of methodological steps to investigate the state of research. More specifically, an SLR relies on a well-defined and evaluated review protocol to extract, analyse and document the results as illustrated in Figure 2. We adopted the guidelines in [24] with a three-step review process that includes *planning*, *conducting* and *documenting* the SLR. The review is complemented by evaluation of the outcome of each step, as illustrated in Figure 2. We also provide a taxonomical classification and comparison of the reviewed studies. A taxonomical classification is the foundation for comparative analysis of studies based on our defined comparison attributes (Section 3.5) that are subject to external evaluation prior to results reporting in [34].

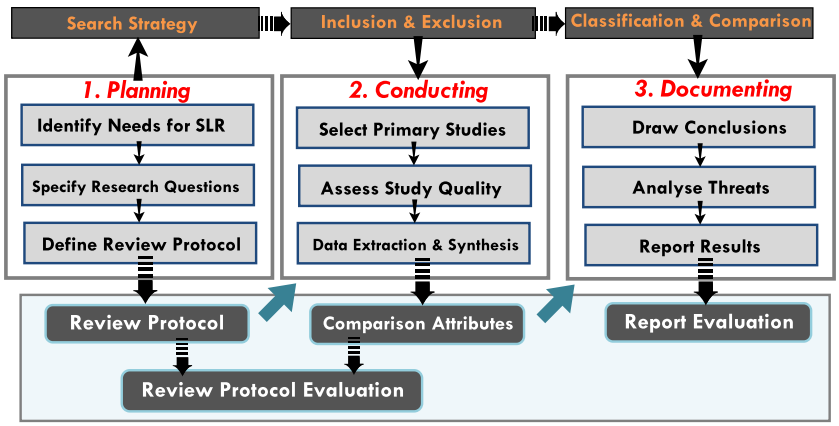

Figure 2. Systematic review process for classification and comparison of reuse knowledge in architecture-centric software evolution.

DOI: 10.1002/smr

3.1. Definition and evaluation of the protocol for systematic review

According to the guidelines in [24], the review protocol drives the *planning*, *conducting* and *documenting* phases of the systematic review as illustrated in Figure 2. The *protocol definition* is provided in the reminder of Section 3. More specifically, protocol for SLR includes (i) identification of the needs and objectives for SLR (Section 3.2), (ii) definition of search strategies to identify, include and exclude and qualitatively analyse the relevant literature (Sections 3.3 and 3.4), (iii) data extraction and results synthesis (Section 3.5) and (iv) results classification (Section 3.6). We developed the review protocol by following the guidelines in [24, 39, 40] and our experience with conducting the systematic review [9, 26]. Additional details about the review protocol are provided in [34].

As suggested by [9] and [24], we externally *evaluated the protocol* before its execution. We asked two external experts for feedback, who had experience in conducting SLRs in an area that overlaps with software architecture research (see Acknowledgement section). The feedback made by the expert resulted in a refined protocol. We also performed a pilot study of the systematic review with 15 (approximately 50%) of the included studies. The objective for conducting a pilot study was to first reduce the bias for (i) identification of primary studies, (ii) extraction of data from these studies and (iii) synthesising the results of review. On the basis of the external review of the protocol, we expanded the review scope, improved search strategies and refined the inclusion/exclusion criteria during the pilot studies (see Section 3.3 for details).

3.2. Planning the review

The review plan consists of three steps as (i) identifying the needs for SLR, (ii) specifying the research questions and (iii) defining and evaluating the review protocol, as illustrated in Figure 2.

3.2.1. Identify the needs for systematic literature review. The needs for SLR have been identified in [9] and its contribution already justified in Section 2. This SLR complements the existing reviews on ACSE [6, 9, 10, 14, 17] to investigate the state of research for application and acquisition of AERK (cf. Table I). Although the progress of research on architecture evolution reuse [13, 22, 23] is reflected over more than a decade [23], we did not find any evidence to systematically synthesise the collective impact of existing research on reuse knowledge (Section 2.3). Therefore, in this SLR, we aim to classify and compare existing research; identify the research potential and its limitations; and outline future dimensions for methods, techniques and solution that enable evolution reuse in software architectures. In addition, the research questions help us to (i) outline the scope and contributions of SLR and (ii) define and evaluate the review protocol to conduct the SLR.

3.2.2. Specify the research questions. The research questions are based on our motivation to conduct the SLR, that is, the answers provide us with an evidence-based overview of the definition, application and acquisition of reuse knowledge to support ACSE methods and techniques. We define three *research questions* that represent the foundation for deriving the search strategy for literature extraction. The *objective* outlines the primary intent of investigation for each question. In addition, a comparative analysis allows us to analyse the collective impact of research, represented in terms of comparison attributes (in Section 3.5, Table V).

- *Research question 1—How evolution reuse knowledge is defined, classified and expressed in existing literature to enable architecture-based software change management?*
- Objective—To understand the existing *classification* and *representation* of AERK that provides a foundation for a detailed comparison of solutions to enable ACSE.

- *Research question 2—What are the existing methodologies and techniques that support application of reuse knowledge to evolve software architectures?*
- Objective—To *identify* and *compare* existing solutions that support an explicit reuse of change implementation mechanisms to enable design-time evolution and runtime adaptations in architectures.

- *Research question 3—What empirical approaches are employed to discover evolution reuse knowledge?*

DOI: 10.1002/smr

- Objective—To *investigate* and *compare* the available support for empirical acquisition/discovery of reuse knowledge and expertise that can be shared to guide architecture evolution.

3.3. Conducting the review

To conduct the review, we follow a three-step process as (i) searching the studies for review, (ii) selection and qualitative assessment of studies and (iii) extraction and synthesis of data from studies, as illustrated in Figure 2.

3.3.1. Selection of primary studies for review. The search terms used to identify primary studies were developed using suggestions in [39] and guided by the research questions (cf. Section 3.2). Our search process comprises of *primary* and *secondary* search.

- *Primary search* is a five-step process to identify and retrieve the relevant literature. The summary of each step involved in primary search is presented in Table II.
- *Secondary search* includes (i) review of references/bibliography section in the selected primary studies to find other relevant articles, (ii) review of citations to the selected primary studies to find any relevant articles, also known as a backward pass [24, 39] and (c) identify and contact authors of selected primary studies for extended versions of the research, if required. The secondary search did not lead to identification of any relevant studies. The secondary search and study selection was performed iteratively until no new studies were found.

The research question resulted in a composition of search string applied to six databases as illustrated in Figure 3. We extracted published peer-reviewed literature from years 1999 to 2012 (inclusive). The year 1999 was chosen as the preliminary search that found no earlier results related to any of the research questions with 1550 manuscripts extracted. Because we used our primary search criteria on title and abstract, the results provided a relatively high number of irrelevant studies, which were further refined with secondary search.

Note that we have decomposed the search string for illustrative reasons in Figure 3. To search the primary studies, the sub-strings in Figure 3 were combined and represented as a single search string.

Table II. A summary of the steps in the literature search.

Search step	Description
1. Derive search strings	From RQs (cf. Table I) in Section 3.1
2. Consider synonyms and alternatives	Consider the alternative spellings and synonyms while composing search strings as follows: **Evolution** as (change, restructure, update, extension, adaptation, reconfiguration, migration, transformation and modification) **Methods and techniques to enable reuse** as (customise, pattern, plan, styles, framework and strategies) **Empirical methods for discovery** (identification, extraction, tracing, mining, discovery and acquisition) **Architecture or software architecture** (we only consider the term software architecture as only using architecture resulted in a large amount of irrelevant studies focusing on hardware, network or system architecture)
3. Search-term combinations	Boolean ***OR*** to incorporate alternative spellings and synonyms Boolean ***AND*** to link the major terms. Number of unique search string depends on a multiplier: ([AND] clause) × (<OR>-keywords)
4. Search string division	Dividing strings so that they could be applied to different databases. Assigning unique IDs to every (sub-) search string and customising them for all selected resources
5. Reference management	Citations with Zotero

RQs, research questions.

DOI: 10.1002/smr

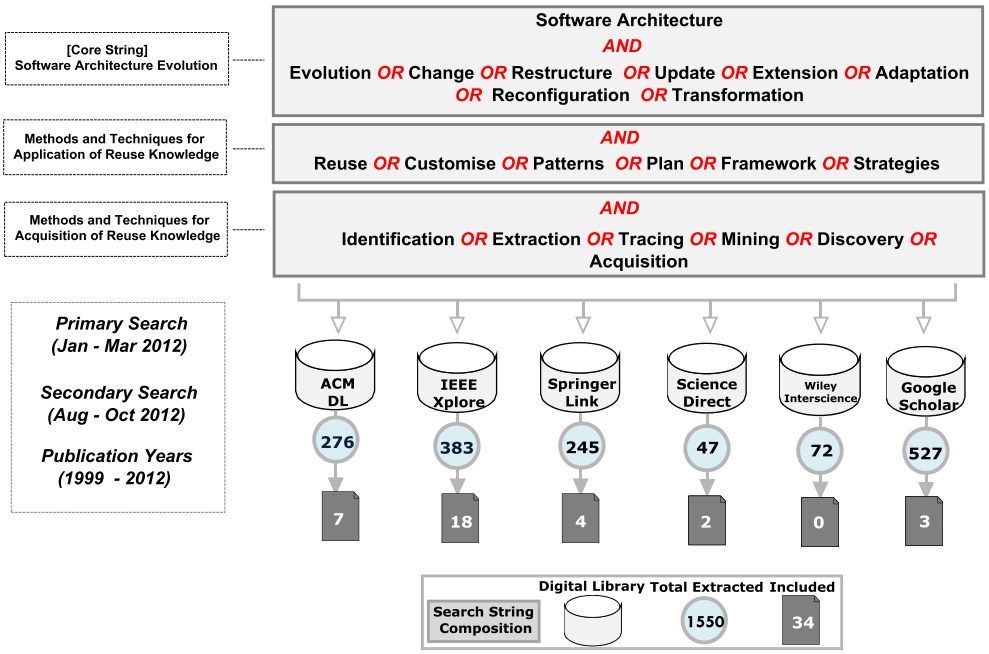

Figure 3. Summary of the primary search process.

3.4. *Selection and qualitative assessment of primary studies*

The study selection comprises of a four-step process that includes *screening*, *initial selection, final selection and qualitative assessment* as presented in Tables III and IV. In Table IV, the qualitative assessment helps us to include/exclude studies and rank the selected studies based on their quality score in the Appendix.

3.4.1. Screening of identified literature. An initial screening is performed for all the studies based on four criteria as presented in Table III. The screening ensures that each of the selected study represents a (i) *peer-reviewed research*, is (ii) *written in English language*, is (iii) *not a secondary* study and is (iv) *not a book*. If the answer to all of these four criteria is [YES], the study is included for initial selection. Otherwise, if the answer to any of the four criteria is [NO], the study is excluded.

3.4.2. Initial selection. This process comprises screening of *titles* and *abstracts* of the potential primary studies. For almost 35% of studies, no decision could be made just on title and abstract, as these papers did not make a clear distinction between an explicit *representation and application* (RQ1 and RQ2) or *acquisition* (RQ1 and RQ3) of reuse knowledge. During initial selection, the decision to exclude [NO] or proceed to the final selection [YES] was based on an examination of the full text for each study.

3.4.3. Final selection. This process is based on a brief validation of the studies, the use of formalisms and tool support and details of the experimental setup. After performing this step, 34 studies were selected. During the secondary search process, references and citations for the 34 selected studies were also reviewed, but this did not lead to the inclusion of any other relevant studies. As a result, 34 studies were included for qualitative assessment.

3.4.4. Qualitative assessment of included studies. For the 34 included studies, we primarily focused on the technical rigor of content presented in the study. We based our qualitative assessment on two factors as *general assessment* (G) and *specific assessment* (S), as summarised in Table IV. Additional details about the quality checklist are provided in [34]. Quality scores provided us with a numerical quantification to rank the selected studies in the Appendix.

On the basis of the quality assessment checklist in Table IV, the quality ranking formula is given as follows. G represents five factors as general assessment criteria from Table IV, providing a maximum

DOI: 10.1002/smr

Table III. Summary of the study selection process (without qualitative assessment).

Step I. Screening			
Is the study in English language?		Yes	No
Is the study a scientific peer-reviewed published research (no white papers or technical reports)?		Yes	No
Is the study not a secondary study?		Yes	No
Is the study not a book or a book chapter?		Yes	No
If [YES] to all four criteria, then go to Step II; otherwise, exclude study			
Step II. Initial selection			
RQ1, RQ2	Does the study presents a method, technique or a solution for application of evolution reuse knowledge?	RQ2, RQ3	Does the study presents a method, technique or a solution for an empirical acquisition of evolution reuse knowledge?
	If [YES], go to Step III; otherwise, exclude study		If [YES], go to Step III; otherwise, exclude study
Step III. Final selection			
RQ1, RQ2	A. Are evaluations for application of reuse knowledge and architecture evolution are provided?	RQ1, RQ2	A. Are the source(s) of reuse knowledge and its discovery/ acquisition presented?
	B. Are formalism and tool support for reuse knowledge application provided?		B. Are the details about the experimental setup of reuse knowledge discovery/acquisition provided?
If [YES] to both A and B, then include study; otherwise, exclude study			

score of 1 (25% weight), S represents a total of five factors as specific items providing a maximum score of 3. S is weighted as three times more than G (75% weight) as specific contributions of a study are more important than general factors for assessment. On the basis of a consensus among the researchers and suggestions from the external reviewers, the criteria for qualitative assessment maximum score was $G + S = 4$, where a 3–4 score represented quality *papers*, a score less than 3 and greater than or equal to 1.5 was *acceptable* and a score less than 1.5 resulted in *study exclusion*.

$$\text{Quality score} = \left[\frac{\sum_{G=1}^{5}}{5} + \left(\frac{\sum_{S=1}^{5}}{5} \times 3 \right) \right]$$

On the basis of the qualitative assessment of *34* studies, we excluded *two* studies to finally select *32* studies for the review. Two studies were excluded because their quality scored was less than 1.5 according to the criteria in Table IV. The selected studies are listed with title, authors, quality score and citation count in the *Appendix*. Please note that quality ranking is an internal metric only that helps us to choose most related studies and does not reflect any comparison or objective interpretation of selected studies.

3.5. Data extraction and synthesis

In order to record the extracted data from the selected studies, we followed [24, 39] and designed a structured format as presented in Table V. The format in Table V records the data as *generic and documentation specific items* and *comparison attributes* for a collective and comparative analysis of research to answer RQ1–RQ3. The data were extracted by locating evidence for each item in the selected studies. Self-explanatory comparison attributes (CA1–CA12 in Table V) are the smallest

DOI: 10.1002/smr

Table IV. Summary of quality assessment checklist.

	General items for quality assessment (G)			
	Score for general items $\sum_{G=1}^{5}$ =	Yes = 1	Partially = 0.5	No = 0
G1	Are *problem definition* and *motivation* of the study clearly presented?			
G2	Is the *research environment* in which the study was carried out properly explained?			
G3	Are *research methodology* and its *organisation* clearly stated?			
G4	Are the *contributions* of the in-line with presented *results*?			
G5	Are the *insights* and *lessons learnt* from the study explicitly mentioned?			
	Specific items for quality assessment (S)			
	Score for specific items $\sum_{G=1}^{5}$ =	Yes = 1	Partially = 0.5	No = 0
S1	Is the research clearly *focused* on *application* or *acquisition* of evolution reuse?			
S2	Are the details about *related research* clearly addressing *evolution reuse* in architectures?			
S3	Is the *research validation* clearly illustrates *application* or *acquisition* of evolution reuse?			
S4	Are the results *clearly validated* in a real (industrial case study) evaluation context?			
S5	Are limitations and future implications for architecture evolution reuse clearly positioned?			

unit of data that we extracted from the literature for comparison purposes and provided for external evaluation. These attributes provide the base for follow-up syntheses, that is, mainly classification and comparison of claims and supporting evidence of evolution reuse detailed in the paper. Instead of reading through detailed results (Sections 4, 5 and 6), external reviewers examined a summary of results provided in [34] to evaluate the protocol and suggestions about documentation of SLR results.

3.3. Classifying the results

To discuss the results, first, we need to provide a conceptual framework to systematically present the existing literature and to identify the required steps that enable ACSE. With the help of a framework, we can organise the reviewed studies in terms of framework processes and activities that (a process-centric view to) support application and acquisition of evolution reuse knowledge. Section 3.3.1 highlights some established models and frameworks for architectural evolution and adaptation, whereas Section 3.3.2 presents the proposed framework to consolidate the existing research on application and acquisition of AERK.

3.3.1. Models and frameworks for architectural migration and evolution. We introduce some established reference models and frameworks based on their impact and relevance for design-time evolution [3] and runtime adaptations of software architectures [7]. The selected models and frameworks are acknowledged through high citations in the research community, provide a detailed documentation and cover a process-based view for design-time evolution as well runtime adaptations detailed as follows:

- *Horseshoe model for architectural extraction and transformation*

The horseshoe model [41] (proposed by Software Engineering Institute in 1999) represents one of the classical approaches for architecture-based reverse and forward engineering. The horseshoe model follows a three-step process including *architectural extraction*, *architectural transformation* and *architecture-based development*. In recent years, a number of solutions have extended the classical horseshoe model that includes (i) SOA Migration Horseshoe [42], (ii) SOA Migration

DOI: 10.1002/smr

Table V. Extracted data and comparison attributes.

ID	Data item	Aim
		Generic and documentation specific data
1	Study ID	Unique ID of study _____
2	Bibliography	a) List of author(s) _____ b) Year of publication _____ c) Source of publication Book chapter ☐ Journal ☐ Conference ☐ Workshop ☐
3	Focus of study	Theme, concepts, motivation clearly presented: Yes ○ No ○
4	Research method	Design and evaluation ☐ Case study ☐ Survey ☐ Experiments ☐ Other _____
5	Application context	Context and application domain: Academic ☐ Industrial ☐ Both ☐ Other ☐
6	Limitations	Constraints, limitations, future research clearly stated: Yes ○ No ○
7	Related research	Positioning and novelty of the research _____
8	Future dimensions	Implications on future research or ideas clearly stated: Yes ○ No ○
		Comparison attributes for RQ1 and RQ2 (derived from [7,11,13, 16])
CA1	Knowledge support	Solutions to support reuse knowledge in ACSE.
CA2	Type of change	Adaptive ☐ Perfective ☐ Corrective ☐ Preventive ☐
CA3	Time of change	Design-time ☐ Runtime ☐
CA4	Means of change	Type of operational support to implement change
CA5	Formalism support	Application of a specific formal approaches in modeling, analysing and executing evolution _____
CA6	Architecture descriptions	UML ☐ ADL ☐ Graph models ☐ State transition ☐ Other _____
		Comparison attributes for RQ1 and RQ3 (derived from [7,11,13, 16])
CA7	Knowledge source	The type of collection—real data set for change instances _____
CA8	Type of analysis	Type of analysis to discover evolutionary knowledge _____
CA9	Type of formalism	Type of formalised methods and for empirical discovery _____
CA10	Time of discovery	Run time extraction ☐ Off-line mining ☐ Other _____
		Comparison attributes for both RQ1, RQ2 and RQ3 (derived from [7,11,13, 16])
CA11	Tool support	Automation support for reuse-driven evolution. Yes ○ No ○
CA12	Evaluation method	Design and evaluation ☐ Case study ☐ Survey ☐ Experiments ☐ Other _____

ADL, architecture description language;
ACSE, architecture - centric software evolution.

Framework [43] and (iii) Architecture Driven Modernisation (ADM) model [44]. The SOA Migration Horseshoe [42] and SOA Migration Framework [43] support migration of a legacy software to service-oriented architectures. In contrast, the ADM [44] model is a more generic model that supports a process-based approach to architecture-based evolution. In Section 3.3.2, we further explain how ADM model helps us to develop the proposed framework to support reuse in ACSE.

- *IBM autonomic framework*

The International Business Machine (IBM's) autonomic framework provides a number of solutions to support autonomic computing [15, 16] by means of dynamic and self-adaptive architectures [6, 11]. Two of the well-established autonomic frameworks are software tuning panels for autonomic control (STAC) [45] and monitor, analyse, plan, execute knowledge (MAPE-K) frameworks [15]. STAC aims to automatically re-architecture the (system source code) to facilitate autonomic adaptation of a software. In contrast, the MAPE-K model supports application of adaptation knowledge to execute dynamic adaptation of a software and ultimately its underlying architecture. In Section 3.3.2, we further explain how

DOI: 10.1002/smr

MAPE-K framework helps us to identify and develop the framework to highlight research on runtime adaptations of architectures.

3.3.2. A framework to classify research on acquisition and application of evolution reuse knowledge. We derive our proposed framework from two well-known reference models: the Object Management Group (OMG's) ADM method [44] and the IBM MAPE-K reference model [15], presented in Figure 4. ADM represents architectural modernisation and evolution at design-time [13, 22], whereas MAPE-K loop supports runtime adaptations [12, 28]. Using established reference models and practices validates the adequacy of our classification and comparison framework. In the following, we briefly present the details for the ADM and MAPE-K models as a basis for our proposed framework to organise research on reuse knowledge. Additional details about the proposed framework provided in [46].

The *ADM horseshoe model* consists of three architectural views: *business architecture*, *application and data architecture* and *technical architecture* (Figure 4A). The existing system with a three-layer architecture is on left, whereas the target system with evolved architectural view on right. The transformation from legacy to target represents the path of evolution.

Therefore, the ADM method involves transformation of the existing legacy architectures in an incremental fashion to the target architectures. The evolution involves the transformation of legacy (procedural) code to new (object-oriented) code. In summary, transformation at any architectural layer relies on three elements:

- *Knowledge discovery* of the legacy system,
- *Definition* of target architecture and
- *Transformation* steps for source to target evolution.

The *MAPE-K reference framework* describes dynamic adaptation process of software. The MAPE-K reference model in Figure 4B is used to communicate the architectural aspects of autonomic systems. Although MAPE-K does not entirely focus on architecture of dynamic software, it provides a reference model to monitor, analyse, plan and execute runtime adaptation of architectures [6, 12, 28]. The MAPE-K reference model relies on the following:

- *Monitoring* monitors the system and measure attributes related to architectural configurations and properties for possible runtime reconfigurations of architecture.
- *Analysing* analyses the measured runtime data and detects violations of the requirements.
- *Planning* generates a change plan for architectural reconfigurations.
- *Execute enacts structural and behavioural* changes to the running system based on the actions recommended by the plan function.
- *Knowledge* includes shared data such as topology information, metrics and policies for dynamic adaptation.

After discussing the ADM and MAPE-K reference frameworks, we propose an integrated framework called *REVOLVE* presented in Figure 5. The reviewed studies are organised (in Section 4) according to the methods and techniques for evolution history analysis (i.e. *change mining* for reuse knowledge

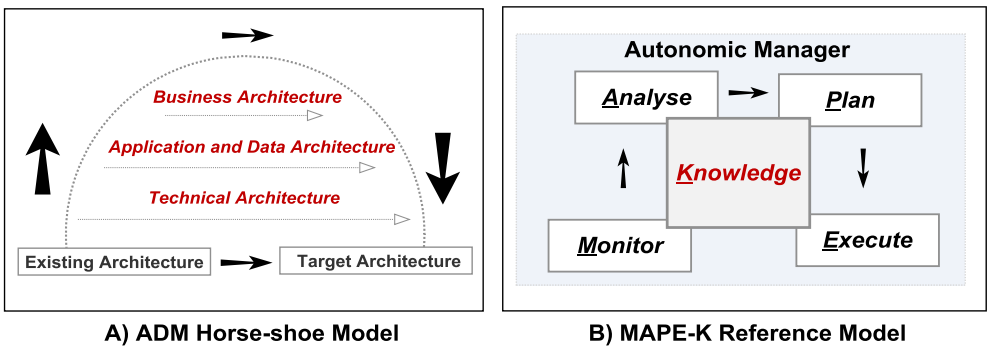

Figure 4. Reference models for knowledge in architecture evolution and adaptation.

DOI: 10.1002/smr

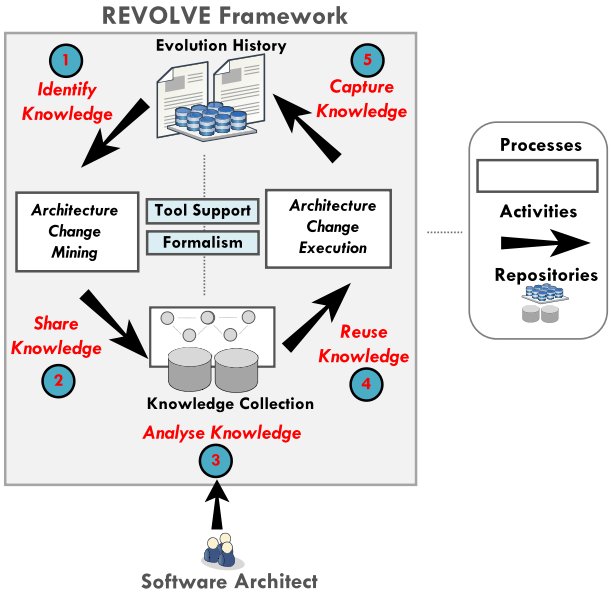

Figure 5. REVOLVE—an integrated view of architecture *change mining* and *change execution*.

acquisition) and change implementation (i.e. *change execution* for reuse knowledge application), which form the two core activities of REVOLVE in Figure 5, each covered by ADM and MAPE-K separately. Additional technical details about the proposed REVOLVE framework in terms of framework activities and framework process are provided in [46].

The concepts and methods used in ADM and MAPE-K reference models can be reused or possibly extended to develop the processes and activities in REVOLVE framework. Method engineering [47] enables us to reuse the existing concepts from existing methods (frameworks, models or solutions) to develop new methods by reusing existing methodologies with reduced efforts and time to derive or develop new solutions. More specifically, during architecture change mining process in the REVOLVE framework, we exploit the knowledge discovery concepts from ADM [44] model for acquisition of evolutionary knowledge from architecture evolution histories. Moreover, the discovered knowledge can be shared and reused as in the MAPE-K framework [15] to analyse, plan and execute architectural adaptation.

The REVOLVE framework in Figure 5 along with the presentation of its processes, activities and their corresponding studies in Table VI is beneficial for ACSE researchers and practitioners. The framework assists ACSE researchers with quick identification of relevant studies. A systematic presentation of state of research provides a foundational body of knowledge to develop theory and

Table VI. Processes, activities and repositories of framework to represent reviewed studies.

	Process	Activity	Repository	Research evidences
1.	Architecture change mining	Identify evolution reuse knowledge	Evolution history	[S7, S9, S10, S17, S29, S31]
		Share evolution reuse knowledge	Knowledge collection	[S8, S9, S10, S17, S31]
		Analyse evolution reuse knowledge	Knowledge collection	[S9, S10, S17, S29]
2.	Architecture change execution	Reuse evolution knowledge	Knowledge collection	[S1, S2, S3, S4, S5, S6, S8, S11, S12, S13, S14, S15, S16, S18, S19, S20, S21, S22, S23, S24, S25, S26, S27, S28, 30, S32]
		Capture evolution reuse knowledge	Evolution history	[S7, S29, S31]

DOI: 10.1002/smr

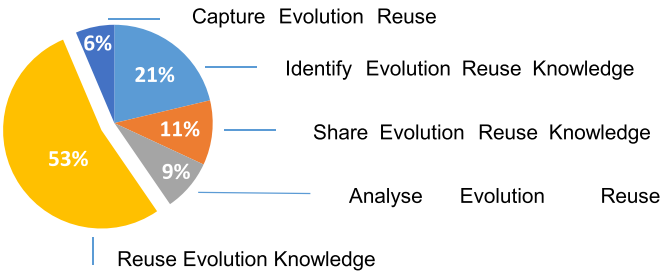

Figure 6. Percentage distribution of studies based on REVOLVE framework activities.

solutions, analyse research implications and to establish future dimensions. In addition, the framework can be beneficial for practitioners interested in understanding the methods and solutions with formalism and tool support to model, analyse, and implement evolution reuse in software architectures. The framework provides a process-centric view of a collection of existing solutions for acquisition and application of reuse knowledge to evolve software architectures.

We conceptualised the logical relationship between individual research elements as a framework in Figure 5. It defines an iterative mechanism to continuously discover reuse knowledge that can be shared and reused to guide ACSE in a semi-automated way. The framework provides an aggregated representation of existing literature. In Section 4, we further discuss the framework processes and activities from Figure 5. The results highlight a lack of solutions that integrate the concept of empirical acquisition of reuse knowledge to guide ACSE with reuse knowledge application. Beyond this review, this framework can assist researchers and practitioners to objectively identify and interpret potential and limitations in state-of-the-art research [34].

4. RESULTS CATEGORISATION AND REUSE KNOWLEDGE TAXONOMY

Our discussion of *results* uses the REVOLVE framework for reuse knowledge from Section 3.3. Central to this framework is a set of *processes*, *activities* and *repositories* (in Table VI, which complements Figure 6). The processes encompass architecture change mining as a complementary and integrated phase to change execution—a concept partially realised in only one of the reviewed studies [S7].[§] We also present the relative distribution of the five activities of the REVOLVE framework. Figure 6 highlights a significant portion (53%) of studies focussing on methods and techniques for application of evolution reuse knowledge. On the other hand, only 9% of studies focus on analysing reuse knowledge. Please note that some of the studies cover different activities of the REVOLVE framework. For example, studies [S9, S10, S17, S13] both represent research on *identifying* and *sharing* reuse knowledge. Similarly, studies [S7, S29, S31] represent *capturing* and *identifying* reuse knowledge.

Table VI summarises the involved processes, their corresponding activities, associated repositories and identified studies—concrete research evidence of the claims. In Figure 5, it is vital to highlight the complementary role of tool support and formalism to support reuse in ACSE. In recent years, there is a growing need for tool and automation support to model and execute architecture evolution in a (semi-) automated way [48, 13].

For example, in Figure 5, to support automation of the activity for *reuse knowledge identification*, the solution must provide a tool or a prototype to analyse architecture evolution histories that contain evolutionary data of significant size and complexity [S29, S31]. A lack of tool support results in an increase in the complexity of architecture evolution process, process scalability (changes from small to large systems) and error proneness in change implementation.

[§]The notation [S_n] (*n* is a number) represents a reference to studies included in the SLR, which are listed in the Appendix. The notation also maintains a distinction between the bibliography and list of selected for SLR.

DOI: 10.1002/smr

4.1. A taxonomical classification of architecture evolution reuse knowledge

The *taxonomy* defines a systematic *identification*, *naming* and *organisation* of reuse approaches into groups that share, overlap or are distinguished by various attributes. A taxonomical classification provides an insight into the commonality or distinction of research themes as denoted in Figure 7. We explicitly discuss three distinct classification types of reuse knowledge research as *generic* and *thematic*. A *solution-specific* classification is introduced in Sections 5 and 6 when we provide a comparison of existing research.

A. *Generic classification* is derived on the basis of a review of studies and our experience with previous SLRs [9, 26] that helped us to refine classification attributes based on studies for analysing the role of reuse knowledge in ACSE. In Figure 7, the literature is classified into *methods and techniques that enables change reuse in architectural evolution* (26 studies, i.e. 81%) and *empirical acquisition or discovery* (six studies, i.e. 19%) *of reuse knowledge and expertise* by exploiting evolution histories. Dotted rectangles in Figure 7 represent the comparison attributes extracted by full-text investigation of the selected studies, as explained in Table V—data extraction and synthesis.

B. *Thematic classification* provides details about the predominant research themes based on *time* and *type of evolution*. In the following, we focus on taxonomy of identified research themes based on a mapping of activities in REVOLVE framework to identified research themes in Figure 7.

1. *Evolution styles* [S1, S5, S8, S11, S13, S21, S23] are inspired by a conventional concept of *architecture styles* that represent a reusable vocabulary of architectural elements (component or connectors) and a set of constraints on them to express a style [49]. Evolution styles focus on defining, classifying, representing and reusing frequent evolution plans [S1, S11] and architecture change expertise [S5, S8, S13, S21]. *Style-based* approaches represent 22% of the reviewed studies addressing *corrective* and *perfective* changes implemented as *design-time evolution*. In the *style-driven* approaches, we observed a trend towards structural evolution-off-the-shelf [S13, S21] and evolution planning [S1, S8] with time, cost and risk analysis to derive evolution plans.
2. *Change patterns* [S2, S6, S12, S14, S15, S16, S17, S20, S21, S27, S29, S30, S24] exploit the same idea as *design patterns* [50] that aim at providing a generic, repeatable solution to recurring design problems. In contrast, *change patterns* follow reuse-driven methods and techniques to offer a generic solution to frequent evolution problems. *Pattern-based* solutions accounted for 41% of total reviewed literature, focusing on *corrective, adaptive* and *perfective* changes supporting both design-time and runtime evolution. *Adaptation and reconfiguration patterns* [S16, S19] are the

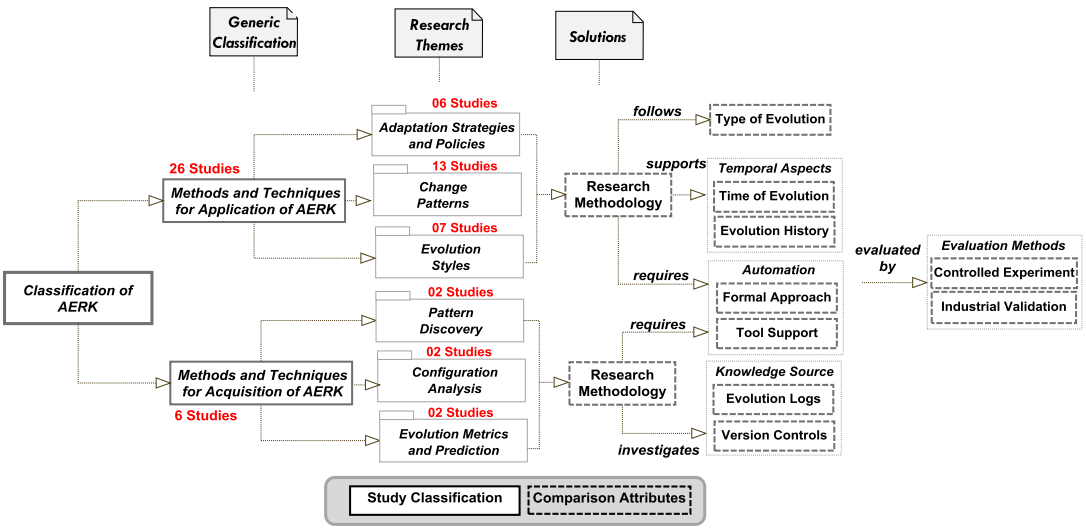

Figure 7. A taxonomical classification of architecture evolution reuse knowledge.

DOI: 10.1002/smr

runtime evolution solutions. The solutions also address the *co-evolution of processes* [S30], *requirements* [S2] and underlying *architecture models*. In addition, a number of studies propose *language-based formalism* [S6, S12, S14, S15] to enable reuse in architectural migration and integration. Unlike styles that only use model-driven evolution, pattern-based changes are expressed as different techniques using model transformations [S2, S30], state transitions [S16, S19] and change operationalisation [S27].

3. *Adaptation strategies and policies* [S3, S4, S25, S26, S28, S32] focus on reuse and customisation of *adaptation policies* [S3, S4], reusable and *knowledge-driven strategies* [S25, S26, S32] and *aspects* [S28] to support the reuse of policies in self-adaptive architectures. With a recent emphasis on autonomic computing, and growing demand for highly available architectures, reuse-driven strategies aim to support knowledge-driven reuse at runtime. These accounted for 19% of reviewed literature with a focus on *adaptive* change. Runtime reconfigurations of architectures are also highlighted in the MAPE-K reference model [5, S4].
4. *Pattern discovery* [S19, S29] represents methods and techniques for post-mortem analysis of evolution history (change logs [S29] and version control [S19]) to discover recurring changes as pattern instances. Pattern-based knowledge acquisition/discovery mechanisms represented a 6% of the total study population.
5. *Architecture configuration analysis* [S7, S31] exploits configuration management techniques to analyse architectural configurations [S7]. It focuses on mining architecture revision histories to capture *evolution* and *variability* in order to represent crosscutting relationships among evolving architecture elements. This is particularly beneficial to classify changes as *atomic* and *composite* types and allows determining the extent to which architectural change can be parallelised (*commutative* and *dependent* changes) [S31]. *Architecture configuration analysis* represented 6% of total study population.
6. *Evolution and maintenance prediction* [S9, S10] focuses on prediction of maintenance and evolution efforts for software architectures. We included two studies in which [S9] represents a set of change scenarios for predicting perfective and adaptive evolution tasks in architectures. In [S10], on the basis of an architectural evaluation and maintenance prediction, the required maintenance and evolution effort for a software system can be estimated [S10].

4.2. A mapping of identified research themes to activities in REVOLVE framework

Although the REVOLVE framework has provided a broader categorisation of research, some observations and interpretation of the results suggested an explicit mapping among the identified research themes and the activities of REVOLVE framework. Figure 8 provides a mapping of the framework's activities (cf. Figure 6) and the identified research themes (cf. Figure 7) to classify and compare application (Section 5) and acquisition (Section 6) of AERK. The circles on the right axis in Figure 8 represent mapping between framework activities and identified research themes for a study reference (e.g. '8' represents 'S8' in the Appendix list of selected studies). Alternatively, the circles on the left axis represent publication map (providing a temporal distribution, 1999 to 2012) for framework activities and identified research themes.

In this section, an iterative mapping process has been employed to present the identified research themes and to provide an answer to the first research question (RQ1). The map as bubble plot is depicted in Figure 8 to enable a mapping of research themes to activities of REVOLVE based on the following:

- Five activities of the REVOLVE framework (cf. Figure 6) along the horizontal axis.
- Six identified research themes (cf. Figure 7) along the vertical axis.

For example, in Figure 8, the bubble at the right axis and at the intersection of 'research theme' *change pattern* (CP) and 'framework activity' *knowledge reuse* (KR) represents the studies [S2, S6, S12, S14, S15, S16, S19, 20, 22, 27, 30] that support change patterns to apply reuse knowledge in ACSE. Alternatively, the bubble at the left axis that intersects 'CP' and 2012 represents the studies [2, 17, 48] published in 2012 and focus on change patterns. The relative size of the bubble indicates the total number of studies (bigger the size, more studies a theme represents).

DOI: 10.1002/smr

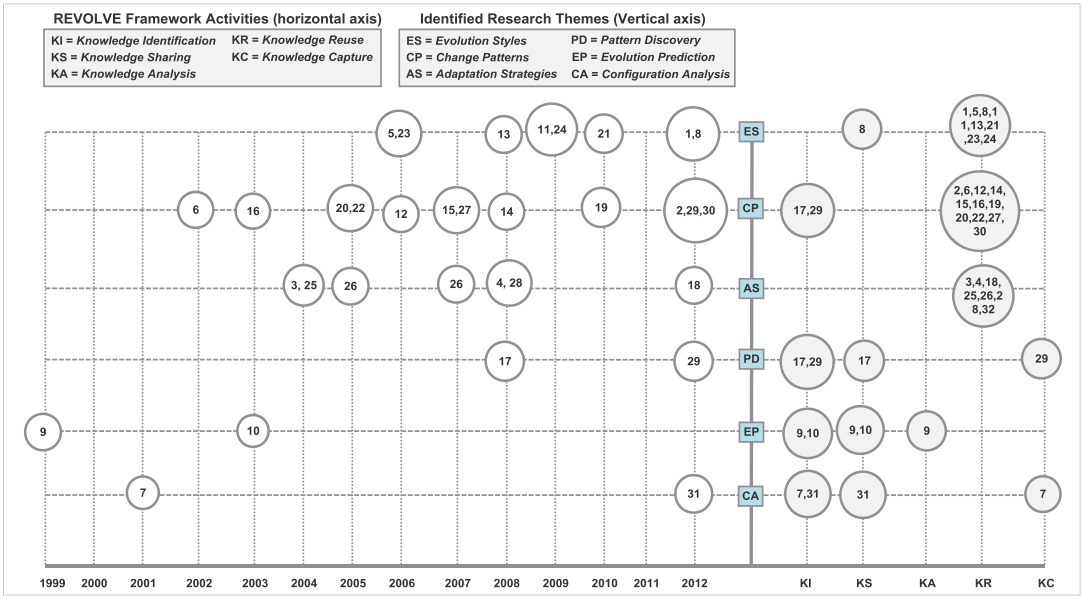

Figure 8. Study mapping over the range of research themes, REVOLVE activities and time period.

4.3. *Definition of architecture evolution reuse knowledge*

Research Question RQ1 addresses how AERK is defined and expressed in the context of ACSE and is answered in this section. After we have defined AERK here, we answer RQ2 (application of reuse knowledge in Section 5) and RQ3 (acquisition of reuse knowledge in Section 6).

In the reviewed studies, we observed that interpreting and assessing individual studies as isolated solutions to a specific research problem lacks consistency in representing what exactly defines AERK and how it is classified and expressed in literature. This could be a direct consequence of the respective author views on *how* to achieve reuse in a solution-specific context. For example, the concept of *evolution style* has distinct and diverse interpretations as Garlan *et al*. who define *evolutionary plans* [S1, S11] following a style, whereas Tamzalit *et al*. exploit styles as *evolution patterns* [S13, S21, S23]. Moreover, Cuesta *et al*. express *evolution styles as an integrated part of architectural knowledge* [S8] that drives architecture evolution. In addition, Yskout *et al*. utilised *change patterns* for architecture co-evolution [S2], Côté *et al*. for *pattern-to-pattern integration* [S27], Gomma *et al*. for *runtime adaptations* [S16, S19] and Zdun *et al*. exploited *language-based formalism* for *evolution and integration* patterns [S6, S12, S15]. This reflects a lack of consideration of *what* existing methods could be leveraged, extended or refined to achieve reuse that drives ACSE [3, 9, 11].

Evolution in the reviewed literature refers to design-time changes [S1, S2, S6, S13, S20] or runtime adaptations [S3, S4, S16, S25] as perfections, reconfigurations or corrections in architectural structure and behaviour [10]. We observed that the term *evolution* (also including evolving, evolve and co-evolution) has six variations as *change* (also including changing): reconfiguration, adaptation, restructuring, update, transformation and migration. The reasons for distinctive terminologies are as follows:

- *Type of architecture change* refers to *corrective*, *adaptive* (also *reconfigurative* [S16, S19]) and *perfective* (also *updative* [S23], *restructurive* [S21], *transformative* [S5] and *migrative* [S6]). With a more conventional interpretation of ISO/IEC 14764 and architectural change characterisation [10], we did not find any study to support *preventive* changes. This indicates that existing work lacks support for reuse in pre-emptive and pro-active evolution of architectures [11, 15].
- *Time Constraint of Change* refers to *evolution*, *change*, *update* and *restructure* for design-time evolution [3], whereas *reconfiguration* and *adaptation* refer to *runtime evolution* [S3, 6]. In Figure 9, there is a clear inclination (53% of total studies) towards style-driven approaches, evolutionary plans and model co-evolution for design-time (also known as static evolution). In contrast, runtime (also known as dynamic evolution) comprises of 28% of studies focussing on self-adaptation and runtime

DOI: 10.1002/smr

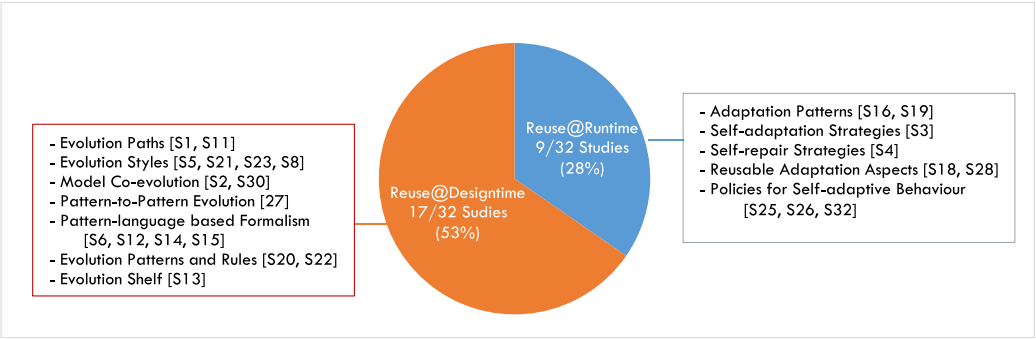

Figure 9. Study distribution—time constraints of architectural evolution reuse.

reconfigurations reflected by studies published in 2004 and 2009. However, with a growing importance of autonomic computing [15] and the context of high-availability architectures [S3, S4, S16, S19, S25, S26, S32], there is a need to realise the potential of *reuse at runtime* reflected by the MAPE-K model.

This suggests that evolution is an unclear term in the context of *types* and *time* of architectural changes making it hard to implicitly derive a unified or aggregated definition for evolution reuse knowledge. In the study titles, the keyword 'evolution' occurs 10 times, 'change' five times and 'adaptation' six times (i.e. approximately 34%, 15% and 19%, respectively). Because of a characterisation of architectural change types [10] and times of evolution [25], a clear consensus or unified definition is not possible. In fact, it would only limit the acceptance of the concept with a narrow view based on available evidence. However, an aggregated definition of evolution reuse knowledge is important to classify and compare the existing research. We further discuss the types and time or architectural changes in Section 5, while answering RQ2—a comparison of method for application of reuse knowledge.

Reuse in the reviewed studies is expressed as evolution styles (seven studies, 22%), change patterns (13 studies, 40%) and *adaptation strategies and policies* (six studies, 19%) in Figure 7. An interesting observation is that although they are novel as methodical approaches, both evolution styles and change patterns conceptually extend the more conventional concepts of *architecture styles* [49] and *design patterns* [50] to represent evolution expertise. Evolution styles [S1, S13, S21] primarily aims at defining, classifying, representing and reusing frequent corrective and perfective changes as a design-time activity. In contrast, *change patterns* [S2, S16, S19] promote the 'build-once, use-often' philosophy to offer a generic, repeatable solution to frequent adaptive, corrective and perfective changes as design-time and runtime-time evolution. The concept of reusable *adaptation strategies and policies* is only represented in the context of reuse plans [S3, S4, S25] and aspects [S28] for self-adaptive architectures.

Once we have identified the relative representation and expression of *architecture evolution* and *evolution reuse*, we can provide a consolidated view of AERK in the context of ACSE. We provide an aggregated definition of AERK as

> A collection and integrated representation (problem-solution mapping) of empirically discovered generic and repeatable change implementation expertise that can be shared and reused as a solution to frequent (architecture) evolution problems.

In the existing literature, the generic and repetitive solutions are predominantly expressed as evolution styles and patterns. In addition, frequent evolution operations represent addition, removal or modification of architecture elements as design-time change or runtime adaptation. Some studies [S1, S11, S13, S20] implicitly denoted reuse as a first-class abstraction—by operationalising and parameterising changes—to resolve recurring evolution tasks. In summary, to answer RQ1, we provided a definition of a generic and thematic classification scheme and organised research about reuse knowledge in ACSE along this scheme. A *classification*, *definition* and *representation* of reuse knowledge are missing in the existing literature to reflect a consolidated impact of research that has progressed for more than a decade (1999–2012). This classification is not meant to be exhaustive and might need to be adapted to

DOI: 10.1002/smr

consider future developments. Figures 7 and 8, however, provide a foundation for a more fine-granular classification and comparison of studies as discussed in Section 5 in the succeeding text.

5. APPLICATION OF ARCHITECTURE EVOLUTION REUSE KNOWLEDGE

On the basis of the generic and thematic classification in Section 4, to answer to RQ2, we classify and compare the existing methods and techniques that support application of evolution reuse knowledge based on the generic and thematic classification in Section 4. A systematic identification and comparison of existing research are particularly beneficial to gain an insight into aspects of *problem-solution mapping* and *architecture evolution characterisation* or to assess *formalisms and tool support*. The comparative analysis is presented as a number of structured tables (Tables VII and VIII). Additional details of synthesised data are available in [34]. In this section, a thematic coding process has been employed to identify the comparison attributes (cf. Table V) and to provide an answer to the RQ2. More specifically, *what are the existing methods and techniques that enable application of reuse knowledge to support architecture evolution* is answered in Section 5.1 and *how to compare the existing techniques to analyse a collective impact of existing research that enhance evolution reuse* is answered in Section 5.2.

5.1. Methods and techniques for application of evolution reuse knowledge

On the basis of the classification of research themes, we focus on answering RQ2 with Table VII. It has three columns associated with the following aspects:

- *Problem view—Why* there is a need for reuse knowledge to address recurring evolution problems?
- *Solution view—How* do solutions provide methods and techniques to address these research problems?
- *Comparison view—What* are the trends, type, means and time of evolution, formalism and tool support, architectural description notations and evaluation methods? See Table VIII for details.

For each reviewed study, the problem and solution views are captured in Table V (with ID 5, 6 in generic and documentation specific items) and represented in Table VII. Whereas the comparison view is represented with a set of comparison attributes in Table V. Note that because of the classification scheme (*styles versus patterns versus strategies and policies*), we denote adaptation patterns [S16, S19] as a sub-theme of change patterns [S2, S17]. For example, Table VII highlights *change patterns* as a solution to *address the problems of continuous runtime adaptations* of software architectures. More specifically, the studies [S16, S19] propose adaptation patterns to support reuse of architectural configurations and adaptations. Furthermore, Table VII serves as a catalogue for problem-solution map along with the available evidence to support application of reuse knowledge.

5.2. Comparison of methods and techniques for application of evolution reuse knowledge

In order to go beyond the analysis of individual studies, a holistic comparison of existing research based on comparison attributes including their objective and concrete evidence is provided in Table VIII. We compare available methods and techniques based on comparison attributes CA1 to CA12 (cf. Table V). The comparison of research methodologies to support the application of evolution reuse knowledge is based on eight distinct comparison attributes CA1–CA6, CA11 and CA12 from the full list (remaining ones will be covered in Section 6).

CA1: What are the identified research trends for reuse in architecture-based evolution and adaptation?

Objective: The aim is to identify available solutions that support reuse knowledge for ACSE. In addition, an overview of research builds the foundation for a comparative analysis of individual methodologies as discussed in the succeeding text and mapped out later on in Figure 10. Each

DOI: 10.1002/smr

Table VII. Methods and techniques to enable application of evolution reuse knowledge.

Research problem	Solution (method and techniques)	Studies
	Evolution styles	
How to o enable evolution planning and trade-off analysis?	**Evolution paths**—to plan and apply reusable evolution strategies.	[S1, 11]
How to achieve recurring structural evolution of architecture?	**Evolution shelf**—library of reusable and reliable evolution expertise.	[S13]
How to enhance change reusability and architecture consistency?	**Update styles**—reuse expertise for restructuring and updating architectures.	[S21, S23]
How to exploit architecture knowledge as an asset for architecture evolution?	**AK-driven evolution styles**—use of AK as evolution styles to constrain and trigger evolution	[S8]
How to reuse in transformation and refinement of component-model to service-driven architectures?	**Style-based transformations**—to achieve migration from component-based architecture to business-driven service architecture.	[S5]
Change patterns		
How to co-evolve process, requirements with architectures?	**Co-evolving models**—reusable patterns to enable co-evolution in process and requirements to their underlying architectures.	[S2, S30]
How to enable a continuous runtime adaptation of architectures?	**Adaptation patterns**—reuse@runtime to support architectural reconfigurations and self-adaptations.	[S16, S19]
How to exploit the reuse of design methods, documents and process for architecture migration and evolution?	**Pattern-to-pattern evolution and integration**—evolution operators and design documents to tackle requirement and architecture changes [S27]. Model-based migration and integration of process-centric architecture models [S12, S15].	[S27, S12, S15]
How to enable an incremental migration of legacy architecture by means of reusable decision models?	**Pattern language-based formalism**—to facilitate a piecemeal migration of architecture models.	[S6, S14]
How to effectively manage evolution at different architectural abstractions?	**Evolution patterns and rules**—to model, analyse and execute architectural transformations at different abstraction levels.	[S20, S22]
Adaptation strategies and policies		
How to provide mechanisms for architecture to adapt at runtime in order to accommodate varying resources, system errors and changing requirements?	**Strategies for self-adaptation**—supported with stylised architectural design models for automatically monitoring system behavior falling outside of acceptable ranges, and then a high-level repair strategy is selected.	[S3, S4]
How to utilise reusable aspects to develop self-adaptive architectures?	**Reusable adaptation aspects**—to reusable aspects and policies to develop self-adaptive architectures.	[S28]
How to efficiently construct system global adaptation behaviour according to the dynamic adaptation requirements?	**Composable adaptation planning** that provides a systematic coordination mechanism to achieve effective and correct composition. It also allows prototyping, testing, evaluation and injection of new adaptation behaviours for component-based adaptable architectures.	[S18]
How to specifying and enact architectural adaptation policies that drive self-adaptive behavior?	**Knowledge-based adaptation management**—for reasoning and decision-making about the timing and nature of specific adaptations grounded on knowledge-based adaptation policies.	[S25, S26, S32]

AK, architectural knowledge.

DOI: 10.1002/smr

Table VIII. A holistic comparison of methods and techniques to support application of reuse knowledge.

Methods/techniques		Comparison attributes							
		Research trends (CA1)	Type of change (CA2)	Time of change (CA3)	Means of change (CA4)	Evolution support formalism (CA5)	Architecture description (CA6)	Tool support (CA11)	Evaluation method (CA12)
Evolution styles	Evolution paths [S1, S11]	Evolution plans	Corrective and perfective	Design-time	Change operations, model transformation^{++}	QVT-based model evolution	Acme ADL and UML 2.0^{++}	AEvol	Case study
	Evolution shelf [S13]	Evolution styles	Corrective^{++} and perfective	Design-time	Model transformation	QVT-based model evolution^{++}	Acme ADL and UML 2.0^{++}	--	Case Study
	Update styles [S21, S23]	Updating styles [S11], architecture style [S10]	Corrective^{++} and perfective	Design-time	Model transformation	Graph transformation rules^{++}	ADL^{++} and UML 2.0,	AGG [S11] and USE[S10]	Case study
	AK-driven evolution styles [S8]	AKdES	Corrective^{++} and perfective	Runtime	Model t ransformation	QVT-based model evolution	ATRIUM metamodel	ATRIUM	Case study
	Style-based transformations [S5]	Style-based evolution and refinement	Corrective^{++} and perfective	Design-time	Model transformation	Graph t ransformation rules	UML profile for SOA, graph model	Poseidon and GTXL	Case study
Change patterns	Model co-evolution [S2, S30]	Requirements [S2], business process [S30]	Adaptive and corrective	Design-time	Model transformation	--	UML 2.0 [S2] and graph model [S15]	VIATRA[S2] -- [S15]	Industrial validation [S2] Case study [S2]
	Adaptation patterns [S16, S19]	Adaptation state-machines	Adaptive and perfective	Runtime	Reconfiguration^{+} $^{+}$ operations	State transition	XTEAM, xADL and UML 2.0	REPLUSSE [S6] and SASSY [S9]	Case study
	Pattern-to-pattern evolution	Pattern-to-pattern evolution and integration	Corrective and perfective^{++}	Design-time	Change operations^{++}	Jackson's framework [S27]	Context diagram,	--	Case study
	Pattern language-based formalism [S6, S12, S15, S14]	Pattern-based migration [S6], integration [S12, S15] and evolution [S14]	Corrective, perfective and adaptive	Design-time	--[S6, S14], Model transformation [S12, S15]	--[6] Model-driven software development [S12, S15] and RADM [S14]	IDL [6], UML 2.0 and XMI [S12, S15], [S14]	-- [6], MDSD tool chain [S12, S15] and ArchPad[S14]	Case study [12, 15], migration of document archival system [S6] and industrial case study[S14]

(Continues)

J. Softw. Evol. and Proc. 2014; **26**:654–691
DOI: 10.1002/smr

Table VIII. *Continued*

Methods/techniques		Comparison attributes							
		Research trends (CA1)	Type of change (CA2)	Time of change (CA3)	Means of change (CA4)	Evolution support formalism (CA5)	Architecture description (CA6)	Tool support (CA11)	Evaluation method (CA12)
	Evolution patterns and rules [S20, S22]	SAEV [S20], TranSAT[S22]	Corrective and perfective	Design-time	Change operation, evolution rules [S20] and model transformation [S22]	SAEV, ECA [S20] and AOSD [S22]	ADL [20] and AgroUML [S22]	--[s20] SafArchie [S22]	Case study
Adaptation strategies and policies	Strategies for self-adaptation and self-repair [S3, S4]	Rainbow framework [S3], style-based adaptation [S4]	Adaptive, perfective and corrective^{++}	Runtime	Adaptation operators and repair strategies [S4]	--	ADL [S3]$^{++}$ and ACME [S4]	Rainbowand stitch language [S3]	Case study
	Reusable and composable adaptation aspects [S28, S18]	Aspect-oriented architecture [S28], composable adaptation planning [S18]	Adaptive and corrective^{++}	Runtime	Aspect generation and weaving [S28]$^{++}$ Composable adaptation plans [S18]	CaesarJ AO-programming language [S28], --[S18	--[S28], Component architecture model [S18]	--	Case study
	Adaptation policies for self-adaptive behaviour [S25, S26, S32]	Knowledge-based adaptation management	Adaptive and corrective^{++}	Runtime	Knowledge-based adaptation policies	Architectural adaptation manager	xADL	KBAAM	Case study

'--' represents an attribute not discussed in the reviewed study; '++' represents an implicit discussion of the attribute, the remaining is all explicit in the literature.
AK, architectural knowledge; ADL, architecture description language; IDL, interface description language; AGG, attributed graph grammar; XMI, XML metadata interchange; GTXL, graph transformation eXchange language; SAEV, software architecture evolution; ECA, event condition action; AOSD, aspect oriented software development; KBAAM, knowledge-based architectural adaptation management; XTEAM, eXtensible tool-chain for evaluation of architectural models; MDSD, model driven software development; RADM, reusable architecture description models.

DOI: 10.1002/smr

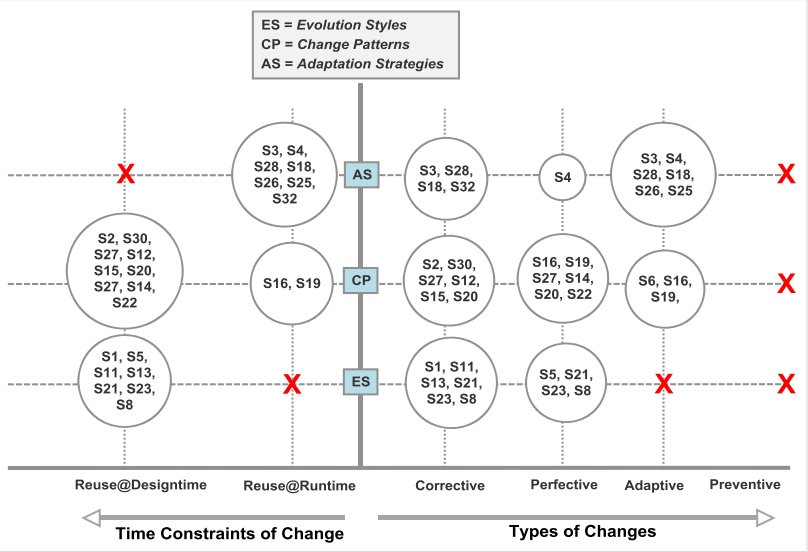

Figure 10. A comparison map of research trends—based on time and types of changes.

theme (from Section 4.1) contains one or more trends. Figure 10 provides a mapping of the research themes (*y*-axis) to types and time of architectural changes (*x*-axis). For example, in Figure 10, the study [S4] represents *adaptation strategies* for *perfective changes* at runtime.

1. *Evolution-off-the-shelf*—We observed a trend following evolution styles for structural evolution [S1, S11, S13] in component-based architectures and *evolution planning* [S1, S11] based on time, cost and risk of changes to define alternative evolution strategies. An interesting observation is a recent emergence of *evolution style* [S8] that exploits AK as an asset to drive evolution-off-the-shelf [S13]. In Figure 10, our comparison suggests that evolution style-based approaches only focus on corrective and perfective type changes [10]. We could not find any evidence to support adaptive or preventive type evolution. Evolution styles are limited to supporting only design-time evolution in software architectures.
2. *Pattern and language-based formalisms*—Pattern-based solutions address the co-evolution of *business processes* [S30] and *requirements* [S2] along with their underlying architecture models. *Adaptation* [S19] and reconfiguration *patterns* [S16] support dynamic adaptations as well. Pattern language-based solutions aim at building a system-of-patterns to support *migration* [S6], *integration* [S12, S15] and *evolution* [S14] of component-based architectures. On the basis of the comparison map in Figure 10, we can conclude that pattern-based techniques enable corrective, adaptive and perfective type changes but do not address preventive change. Pattern-based solutions are heavily biased towards design-time evolution. However, studies on reconfiguration and adaptation patterns suggest a potential for future research to address dynamic adaptation by leveraging change patterns [S19].
3. *Reuse knowledge for self-adaptation and self-repair*—In particular, *self-adaptive* and *self-repair* techniques reflect the recent emphasis on autonomic computing and growing demands for high-availability architectures.

Reuse-driven self-adaptation enables dynamic evolution reflected as *reusable adaptation strategies* for adaptive architectures [S3, S25]. In addition, knowledge-based *adaptation policies* [S4, S26, S32] enhance self-organisation and repair of dynamic adaptive architectures. Self-adaptation strategies are the key to supporting dynamic and high-availability architectures. Unlike styles and patterns, reusable adaptation strategies focus on runtime reuse of adaptation expertise. Moreover, self-repair [S4, S26] policies promise to tackle preventive type of changes. However, on the basis of the mapping in Figure 10, we did not find explicit evidence to address preventive changes that corresponds to unanticipated evolution [11].

DOI: 10.1002/smr

CA2: What types of architectural changes are supported to achieve evolution reuse?

Objectives: To investigate the type of change support offered by existing ACSE solutions: corrective, perfective, adaptive and preventive changes [10]. This change typology is based on the ISO/IEC 14764 standard and architecture change characterisation in [10].

In Figure 10, style-driven approaches focus on corrective and perfective changes (also reported as *updative* [S23], *restructurive* [S21], *transformative* [S5] and *migrative* [S6]). Pattern-based solutions support *corrective* [S27], *perfective* [S12, S15, S6, S14] *and adaptive* change support (also called *reconfigurative* [S16, S19]). *Adaptation strategies and policies*, as the name indicates, primarily focus on runtime *adaptive* [S3, S4, S28] changes. Note that none of the reviewed studies addresses preventive change that aims to prevent problems before they occur. This suggests a lack of focus on tackling unanticipated evolution [6, S4] in software architectures.

CA3: How do time aspects affect change implementation during architecture evolution?

Objectives: To analyse the temporal aspects [25] in terms of the time (or stage) associated to architecture evolution in Figure 10. The existing evidence suggests the following:

- *Reuse@Runtime* enables application of reuse knowledge at runtime to achieve dynamic adaptation. Reconfiguration patterns reflect reusable strategies as a consequence of growing demands for autonomic and self-adaptive architectures for runtime evolution [S2, S4, S25, S26, S27]. We could not find evidence of style-based approaches that facilitate runtime reuse.
- *Reuse@Designtime* enables application of reuse knowledge at design-time to achieve evolution. Style-driven approaches [S1, S13, S8] are heavily oriented towards design-time evolution. In contrast, pattern-driven reuse is aimed primarily at design-time changes [S2, S30, S27] but also support runtime reconfigurations [S16, S19]. However, adaptation strategies lack explicit support for design-time reuse.

CA4: What are the existing means of architectural change to achieve evolution reuse?

Objectives: To study and compare the change implementation mechanisms and to analyse if there exist any recurring themes among them. We only present the predominant means of change as (at least indicated in five or more studies) individual methods, and techniques are already summarised in Table VIII.*Evolution operators* as the most utilised means of change that could be further classified as *change* [S1, S11, S20, S22, S27], *adaptation* [S19] and *reconfiguration operators* [S16]. *Model transformation* enables design-time evolution as discussed in [S1, S13, S21, S23, S5, S2, S30]. Furthermore, *adaptation plans* exploit *repair strategies* and *aspect weaving mechanism* [S4, S18, S26, S28, S32] for runtime adaptation.

CA5: What types of formal methodologies are exploited to support reuse in ACSE?

Objectives: To analyse the extent to which formal techniques facilitate modelling, analysing and executing evolution reuse. We only present predominant formal methods (at least indicated in three or more studies).

We observed an overwhelming bias towards model-based architecture evolution that is primarily achieved through model transformation with Query/View/Transformation (QVT) [S1, S11, S13] and also graph-based specifications [S11, S10, S5, S12, S15]. This observation is also reported in [9]. The only exceptions are adaptation patterns [S16, S19, S12, S27] that exploit state transition and pattern-to-pattern integration using or architecture evolution.

CA6: What are the notations used for architectural descriptions in evolving architecture models?

Objectives: To identify the modelling notation used to support architecture evolution. We primarily focus on investigating the role of architecture descriptions in enabling and enhancing architecture evolution (at least three studies).

The three commonly used architectural description notation are *UML 2.0* [S11, S13, S23, S2, S19, S12], *architecture description languages* (ADLs) [S11, S13, S21, S16, S20, S3, S25, S26] and *UML profiles* [S5, S22, S18]. The primary motive to use ADLs or UML is the availability of extensive

DOI: 10.1002/smr

research literature and tool support to specify architecture models with model-based verification and transformation to support evolution. Most notable ADLs are ACME and xADL.

CA11: What is the available tool support to enable or enhance reuse in architectural evolution and adaptation?

Objectives: to analyse the role of automation and tool support in enabling the architect to model, analyse and execute reuse in ACSE.

Tool support is significant to assist the architects in decisions-making and automating complex tasks, especially where there is a need to model and choose among alternative evolution paths [S1, S11]. In the reviewed studies, tool support is generally provided in terms of research prototypes. Automation allows an architect to model [S1, S21], analyse and execute generic, reusable strategies for evolution [S2, S1, S21]. However, there is a mandatory user intervention through appropriate parameterisation and customisation of evolution process to accommodate the human perspective before and after evolution [S6, S9, S11, S12]. Some practical issues and lessons learned regarding tool support for architecture evolution reuse have been reported in [48].

CA12: What is the context of evaluation methods to validate research hypotheses or results?

Objectives: The aim is to analyse the context of evaluation, where evaluation context defines the research environment in which the results are evaluated.

The comparative analysis suggests that validation of the proposed solutions or generated results are heavily based on *surveys*, *controlled experimentation* with case studies [S1, S21, S8] or *evaluation in an industrial context* [S2, S6, S14]. It is evident that solutions are heavily oriented towards case study based evaluation, usually in a lab experimentation context. The only exceptions are the studies [S2, S6, S14] that focus on co-evolution of requirements and architectures evaluated in industrial settings.

6. ACQUISITION OF ARCHITECTURE EVOLUTION REUSE KNOWLEDGE

In this section, we investigate the methods and techniques for acquisition of reuse knowledge to answer RQ3, that is, *what are the existing methods and techniques for acquisition of evolution reuse knowledge* (Section 6.1) and *how these methods and techniques can be compared* to consolidate the impact of existing research (Section 6.2). Note that the solutions for this research question (i.e. RQ3) are complementary to the methods and techniques that support application of reuse knowledge in ACSE.

6.1. Methods and techniques for acquisition of evolution reuse knowledge

In Section 5, we identified *change pattern discovery* [S17, S29], *evolution and maintenance prediction* [S9, S10] and *architecture configuration analysis* [S7, S31] as the three research themes to support reuse knowledge acquisition. More specifically,

- *Change pattern discovery* techniques focus on investigating evolution histories for an experimental identification of recurring change sequences as potential change patterns.
- *Evolution and maintenance prediction* methods focus on *maintenance profiles* [S9] and *scenario-based* [S10] prediction of maintenance efforts to enhance or enable architecture evolution.
- *Architecture configuration analysis* deals with architectural system model that tightly integrates architectural concepts with concepts from configuration management. Change composition analysis [S31] focuses on analysing change operationalisation based on a hierarchical composition of change instances, that is, defining and reusing atomic change operations to build up composite change operations.

Solutions for reuse knowledge acquisition primarily focus on the post-mortem analysis of architecture evolution histories to discover evolutionary knowledge. In Table IX, we summarise the

DOI: 10.1002/smr

Table IX. A summary of methods and techniques for acquisition of reuse knowledge.

Research problem	Solution (knowledge acquisition techniques)	Included studies
Change pattern discovery		
How to empirically discover reusable change operators and patterns?	**Evolution history analysis**—post-mortem analysis of architecture *evolution logs* [S29] and *version histories* [S17] to identify change patterns.	[S17, S29]
Maintenance and evolution prediction		
How to predict the efforts of architecture-based maintenance and evolution?	**Maintenance profiling**—the architecture is evaluated using the so-called scenario scripting and the expected maintenance effort for each change scenario is evaluated for perfective and adaptive changes [S9].	
	Scenario-based change prediction—of complex changes during initial analysis of existing architecture, and how and to what extent the process to elicit and assess the impact of such changes might be improved [S10].	[S9, S10]
Configuration analysis		
How to capture and relate changes for architecture configurations?	**Revision history mining**—captures evolution and variability to represent crosscutting relationships among evolving architecture elements [S7].	[S7, S31]
	Dependency analysis—analyse change classification and to dependency analysis [S31].	

problem-solution mapping to highlight research on knowledge discovery. In this section, the problem-solution views are presented and captured in Table V (generic and documentation specific items), whereas attributes CA7–CA12 are presented in Table V as for comparison purposes. We can observe a relative lack of focus on establishing and exploiting experimental foundation for a continuous and incremental acquisition of reuse knowledge.

We have identified only a relatively limited number of studies (6/32 of included studies, i.e. 19% approximately), which do not allow us for any stronger judgments. However, we believe that highlighting the existing literature based on a problem-solution mapping helps us to analyse the current state of research and possible future directions as detailed in Table IX. In addition, summarised results in Table IX allow us to assess methodologies for a collective impact of existing research on acquisition of reuse knowledge.

6.2. A comparison of methods and techniques for acquisition of evolution reuse knowledge

We provide a comparison of existing techniques in Table X that enables reuse knowledge acquisition based on six comparison attributes CA6–CA11 from Table V. The comparative analysis highlights the *sources of knowledge*; the adoption of *empirical approaches*; and the role of *formalisms* and *tool support, type of knowledge discovery* and *evaluation methods*.

We now describe the comparison attributes in detail including their objective and concrete evidence as comparison options used in the columns of Table X.

CA6: What types of knowledge sources are investigated for acquisition of reuse knowledge?

Objective: In order to discover evolutionary knowledge, existing knowledge sources need to be considered. A knowledge source represents a repository that maintains historical ACSE data for knowledge acquisition.

- *Pattern discovery techniques* exploit *change logs* [S29] and *version controls* [S17]) as centrally managed repositories of evolution history. Change logs and version controls contain fine-grained

DOI: 10.1002/smr

Table X. Comparison of methods and techniques for reuse knowledge acquisition.

Comparison attributes						
Methods and techniques	Knowledge source (CA7)	Type of analysis (CA8)	Type of formalism (CA9)	Time of discovery (CA10)	Tool support (CA11)	Evaluation method (CA12)
Change patterns discovery [S17, S29]	Change logs [S29] and version control [S17]	Post-mortem analysis [S29] and architecture snapshots [S17]	Graph mining [S29] and version snapshot [S17]	Design-time	G-Pride [S29] and HEAT [17]	Case study
Evolution and maintenance prediction [S9, S10]	Maintenance profiles [S9] and change scenarios [S10]	Change scenarios based evaluation	Not explicitly mentioned	Design-time	Not explicitly mentioned	Case study
Configuration analysis [S7, S31]	Revision histories [S7] and Change Logs [S31]	configuration management analysis [S7]	Not explicitly mentioned [S7] and graph matching [S31]	Design-time	Mae [S7] and G-Pride [S31]	Case study

DOI: 10.1002/smr

traces of evolution data sets that can be queried and searched to analyse architecture-centric evolution history over time.

- *Evolution and maintenance prediction* utilise maintenance profiles [S9] that represent a set of change scenarios for perfective and adaptive maintenance tasks. More specifically, by exploiting maintenance profile, the architecture is evaluated using the so-called scenario scripting. The expected maintenance effort for each change scenario is assessed. On the basis of architectural evaluation and maintenance prediction, the required maintenance and evolution effort for a software system and its underlying architecture can be estimated.
- *Architecture configuration analysis* investigates architecture *revision histories* [S7] and *change logs* [S31]. Revision histories contain datasets for architectural configuration analysis, reflecting evolution and variability of architectures. These are necessary to represent crosscutting relationships among evolving architectural elements [S7]. In long-term analyses [S31], dependencies among change operations determine if evolution operations could be parallelised on the basis of identified *commutative* and *dependent* change operations.

CA7: What types of analyses are performed on knowledge sources for acquisition of reuse knowledge?

Objective: To analyse the application of knowledge discovery/acquisition mechanisms on knowledge sources.*Post-mortem* analysis [S29] and *version control snapshots* [S17] techniques are employed to discover change patterns. In the context of architecture evolution prediction, scenario-based analyses are used as well as techniques from configuration analysis and management.

CA8: What type of formal methods and techniques are utilised for reuse knowledge acquisition?

Objective: To identify the types of formal methods used for knowledge acquisition.

The role of the formalism, detailing the application of formal techniques, is discussed in three studies. In particular, graph-based formalisms are exploited for *sub-graph mining* [18], [S29] to identify recurring change patterns and *graph matching* [S31] techniques that are used to discover change composition and dependencies among operations. Snapshots of architecture versions are used to discover patterns and possible drifts in architecture from one version to another [S17].

CA9: Is knowledge acquisition performed at design-time or runtime?

Objective: To distinguish between the techniques for runtime and/or design-time discovery or acquisition of reuse knowledge. In all of the reviewed studies, evolution reuse-knowledge discovery is performed as a design-time activity. We did not find any evidence that highlights maintaining and analysing traces of runtime architectural adaptations.

CA10: How are the knowledge acquisition techniques evaluated?

Objective: To compare the type of evaluation methodologies used to validate the knowledge acquisition techniques.

The evaluation of knowledge acquisition techniques is primarily based on surveys, controlled experimentation with case studies or evaluation in an industrial context. Existing solutions mainly apply evaluations based on case study and usually in a controlled lab experimentation.

CA11: What is the tool support for analysing and discovering reuse knowledge from evolution knowledge sources?

Objective: To investigate the extent to which the existing research supports automation and customisation of the knowledge acquisition process with support by prototypes and tools.

Tool support is critical, especially where the amount of data or the complexity of the knowledge source is substantial. It is difficult, time consuming and error prone to perform analyses manually. In most cases, prototypes enable efficient pattern analysis, discovery [S17, S29] and composition analysis [S31, S7].

DOI: 10.1002/smr

7. RESEARCH IMPLICATIONS AND DISCUSSIONS

In this paper, we present the results of a systematic review to analyse the collective coverage and impact of existing research that enable or enhance architecture evolution with reuse knowledge. We classified existing work (Section 4) and provided a comparative analysis for methods and technique that enable application (Section 5) and acquisition (Section 6) of reuse knowledge to guide architecture evolution. In this section, we present a summary of research progress and principle findings of the SLR to highlight trends and possible future research. A yearly distribution of reviewed studies (research progression to-date) and associated research trends are presented in Figure 11. The year 1999 was chosen as the preliminary search that found no earlier results related to any of the research questions.

7.1. Research trends and future directions

In the context of software evolution, research on architecture evolution reuse is continuously growing over more than a decade (as observed in the reviewed studies from 1999 to 2012). As indicated in Figure 11, we did not set a lower boundary for the year of publication in the search process, yet the timeframe of identified studies reflects also the timeframe of emergence and maturation of solutions. The trend curve starts in 1999 with a study on predicting architecture maintenance and evolution [S9]. Since 2004, an interesting observation (cf. Table XI) is a continuous exploitation of the concept evolution styles to support planning [S1, S11], operationalising [S21] and fostering [S13] of reuse knowledge.

A reflection on research trends and possible future directions is presented in Table XI along with the aspects of *methods and techniques to support application and acquisition of reuse knowledge*.

7.1.1. Research trends in application of reuse knowledge. The identified research themes to express reuse knowledge in architecture evolution are primarily classified as *evolution styles*, *change patterns* and *adaptation strategies*. Evolution styles [S1] are focused on deriving generic evolution plans [S11, S8, S21] to support design-time evolution of architectures. In contrast, adaptation strategies [S3] aim to support reusable adaptation strategies [S18, S28] to support runtime evolution. Only change patterns [S2, S16] could support both design-time and runtime evolution in architectures. More specifically, pattern languages [S6, S12] and architecture co-evolution [S2, S30] are the most notable trends for enabling pattern-driven reusable evolution. Although we only identified two studies, adaptation patterns promote reuse in runtime evolution [S16, S19].

- *Future research dimensions*—We can identify the need for future research based on time aspects of evolution reuse that includes the following:
- *Reuse@Runtime* refers to application of reuse to support reuse-driven dynamic adaptation in software architectures (also known as on-line evolution). In an architectural context for high

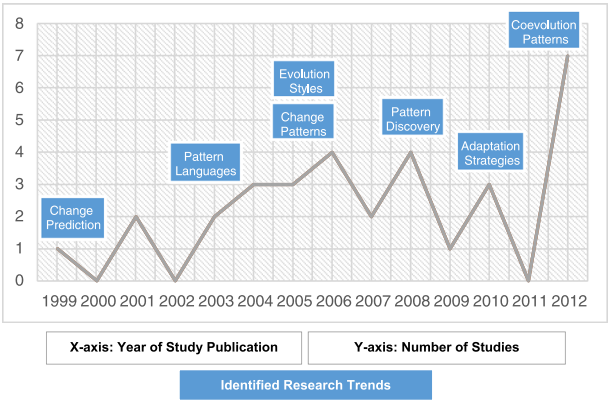

Figure 11. Temporal distribution of primary studies (1999–2012).

DOI: 10.1002/smr

Table XI. A summary of identified research trends and future research dimensions.

Classification	Methods and techniques for application of reuse knowledge			Methods and techniques for acquisition of reuse knowledge		
Solutions	Evolution styles	Change patterns	Adaptation strategies	Pattern discovery	Evolution prediction	Configuration analysis
Identified research trends	Evolution planning [S1, S11, S8]	Model co-evolution [S2, S30]	Self-adaptation and repair [S3, S4]	Log-based post-mortem analysis [S29]	Evolution scenario analysis [S10]	Change configuration analysis [S7]
	Evolution paths [S21, S23]	Adaptation patterns [S16, S19]	Composable adaptations [S18, S28]	Change version mining [S17]	Maintenance profile analysis [S9]	Change composition analysis [S31]
	Evolution shelf [S13, S21]	Pattern languages [S6, S12, S15]	Adaptation knowledge [S25, S26, S32]			
Potential for future dimensions	Reuse@Runtime	Reuse@Designtime		Evolution mining		
	Reconfiguration patterns	Knowledge-driven migration, integration and evolution		Analysing evolution-centric couplings		
	Adaptation plans and REUSABLE Infrastructure	Reuse-driven co-evolution of architectures		Evolution dependency analysis		

DOI: 10.1002/smr

availability, there is an obvious need to capitalise on generic and off-the-shelf expertise to support reuse-driven self-adaptation [S3, S4, S18, S25]. The IBM autonomic framework [15]—*MAPE* loop—embodies the *topology*, *policy* and *problem determination* knowledge to derive configuration plans and to enforce adaptation policies to monitor and execute software adaptations. In contrast to studies [S4, S25, S32], we argue that augmenting the conventional MAPE loop with explicit evolution reuse knowledge can systematically address frequent adaptation tasks. The existing solutions either allow customisation of reusable infrastructure [S3], self-repair [S4] or adaptation aspects [S28] to existing software. However, they lack support for evolution reuse to guide dynamic adaptations. When addressing recurring evolution, the potential lies with *fostering* and *reusing* off-the-shelf dynamic adaptations to enable evolution reuse at runtime.

- *Reuse@Designtime* refers to application of reuse to support generic and reusable evolution in software architectures (also known as off-line evolution). Existing research clearly focuses on styles and patterns for the reuse of generic evolution plans, change operationalisation and model-based architecture co-evolution. With the REVOLVE framework, our review suggests the need to augment styles [S1, S11, S13, S21] and pattern-driven solutions [S2, S30] with repository mining techniques [S17, S29, S31] to discover reusable evolution strategies.

7.2.2. Research trends in acquisition of reuse knowledge. In contrast to reuse knowledge application, we can observe a clear lack of research on knowledge discovery/acquisition techniques (only six studies) despite an acknowledged need. The primary themes for evolution-centric knowledge acquisition represent *pattern discovery*, *evolution prediction* and *architecture configuration analysis*. Change *pattern discovery* aims at investigating change logs [S29] and version control [S17] systems for post-mortem analysis of evolution histories. Frequent change instances from evolution histories are identified and represented as change patterns. Architecture-based prediction of software evolution aims to exploit scenario-based analysis to estimate the efforts of software evolution [S9, S10]. Configuration analysis techniques aim to investigate the evolution-centric dependencies for software architectures [S7, S31].

- *Future research dimension*—The comparative analysis for knowledge acquisition techniques suggests an investigation of evolution-centric dependencies. In particular, we believe in a need for *evolution mining* that aims at *analysing*, *discovering* and *sharing* explicit knowledge to be *reused* to anticipate and guide architecture change management. In the reviewed studies, there is little evidence of architecture change mining. Our review suggests the needs for empirically derived evolution plans and the need to analyse evolution dependencies. Such dependency analysis is significant to identify the *commutative* and *dependent* changes in order to investigate parallelisation of evolution operations.

7.2. Benefits of the systematic review for researchers and practitioners

The classification framework (in Section 5) provides a holistic view of different evolution reuse aspects to be considered in the context of the REVOLVE framework (Figure 5). The trends in Table XI reiterate the fact that among prominent concerns to tackle ACSE are time aspects of evolution. It reflects on the role of formalisms and tool support that can be exploited to leverage conventional data mining techniques for post-mortem analysis of architecture evolution histories. There is a need to develop a tool chain that could automate the REVOLVE framework with appropriate and minimal user intervention.

The classification and comparison and its accompanying templates [34] contain 12 comparison attributes that provides a moderate amount of information. For instance, for the 32 papers and 12 comparison attributes, it creates a collection with $32 * 12 = 384$ data points. As a result, the user can, for example, query and analyse the database based on <Subject: *architecture model evolution*> [Object: using *graph transformation*] (Implications: for *change reuse and architecture consistency*). This is beneficial for the following:

- Researchers who require a quick identification of relevant studies and detailed insight into state of the art that supports application and acquisition of reuse knowledge in ACSE.

DOI: 10.1002/smr

- Practitioners interested in understanding the existing methods with supporting formalism and tool support to analyse and execute evolution reuse.

7.3. Threats to validity of the systematic review

This SLR provides a classification of existing evidence of reuse in ACSE by reviewing and analysing peer-reviewed literature. Apart from addressing the research questions and providing an overview in the field according to the REVOLVE framework, we also identified areas that are not covered in the literature body. This work has been performed on the basis of the review protocol explained in Section 3.

Although the observations and results of systematic reviews are considered to be reliable [39, 40], this type of review work has its own limitations that should be considered [36]. We discuss each of the validity threats associated to different steps in our SLR (cf. Figure 2).

- *Threats to the identification of primary studies*. In our search strategies, the key idea was to retrieve as much as possible the available literature to avoid any possible bias. Another critical challenge in addressing these threats was to determine the scope of our study, because the notion of reuse knowledge means different things to different research communities including software architecture, software product lines and self-adaptive software. Therefore, to cover all and avoid bias, we searched for common terms and combined them in our search string (cf. Figure 3). While this approach decreases the bias, it also significantly increases the search work. To identify relevant studies and ensure the process of selection was unbiased, a review protocol was developed and evaluated.
- *Threats to selection and data extraction consistency*. We have identified a lack of consistent terminologies for reuse knowledge (Section 4). This poses difficulties for the composition of the search queries and the inclusion/exclusion criteria. Such difficulties led us to analyse the terms concerning reuse knowledge that were found on the selected studies. However, because the notion of 'reuse knowledge' is used in numerous studies, but we are specifically concerned with 'architecture (-based) evolution reuse knowledge', we had to exclude a majority of retrieved studies that affected the low precision of our search. In addition, we performed quality assessment (Section 3.4 for details) on the studies to ensure that the identified findings and implications came from credible sources.
- *Threats to data synthesis and results*. The threat to the reliability of results is mitigated as far as possible by involving multiple researchers, having a unified scheme for data synthesis and having several steps where the scheme and process were piloted and externally evaluated. Although as a general practice, we were determined to use the guidelines provided in [24] to perform our systematic review, and we had *deviations* from their procedures as we have detailed in Section 3.

To summarise, we believe that the validity of the study is high, given the use of a systematic procedure, the involvement and discussion among the researchers and external evaluations. The openness of our review by exposing our data in [34] allows other researchers to judge the trustworthiness of the results objectively. This initiative is suggested by the evidence-based software engineering community (e.g. http://www.dur.ac.uk/ebse/).

8. CONCLUSION

Our focus in this SLR was AERK, that is, knowledge specific to reuse in the evolution of software architecture. As such, it forms part of the wider AK research in the software architecture community. The AERK perspective presented in this work shifts the reuse focus from artefacts (such as software architectures) to processes (here, the evolution of architectures).

On the basis of a qualitative selection of 32 studies, we investigated the coverage and concerns of reuse knowledge in ACSE. More specifically, we provide a taxonomical classification and holistic comparison of existing research based on 12 comparisons attributes to derive conclusions about central aspects, gaps and possible future research directions.

DOI: 10.1002/smr

We define what exactly constitutes reuse knowledge in the context of architecture evolution based on the systematic review. Moreover, we derived a taxonomy that aims to assist the researchers in classifying existing and future approaches for reuse-driven evolution that reflects a continuous progression of research over the last decade. We presented the research implications organised by the REVOLVE framework to consolidate the existing work with reflections on future research. The comparative analyses are presented in a number of structured tables. The reported results aim to facilitate knowledge transfer among researchers and practitioners to promote the 'build-once, use-often' philosophy to address recurring evolution. On the basis of the proposed conceptual framework, we distinguish between research efforts on architecture change discovery and mining (6/32 studies, i.e. 19% of the reviewed literature) and architecture change execution (26/32, 81%). Five distinct research activities—*identifying*, *sharing*, *analysing*, *reusing* and *capturing reuse knowledge*—frame the scope of reuse-driven architecture evolution. We also identified a number of research gaps and potential future trends:

- *Reuse knowledge mining and discovery*. In this evolution reuse context, the most frequent research focus is change patterns to promote reuse for both the design-time evolution and runtime adaptation of architectures. Knowledge capturing and identification represent the activities that have received significantly less research effort.
- *Dynamic, runtime evolution*. The solutions for reuse of design-time changes show a relative maturation with change patterns. However, with growing needs for autonomic computing and self-adaptive architectures, more efforts are required to systematically address dynamic evolution. We believe that architecture evolution mining is particularly helpful to discover reuse knowledge that can be shared and reused to address anticipated and unanticipated evolution problems. A relative lack of focus on empirical identification of reuse knowledge suggests the need of solutions with *architecture change mining* as a complementary and integrated phase for *architecture change execution*.

APPENDIX

List of studies for systematic literature review.

Study ID	Author(s), title, channel of publication	Year of publication	Citation count	Quality score
[S1]	J. M. Barnes, D. Garlan and B. Schmerl. **Evolution Styles**: **Foundations and Models for Software Architecture Evolution**. *In Journal of Software and Systems Modeling*.	2012	0	3.8
[S2]	K. Yskout, R. Scandariato, W. Joosen. **Change Patterns**: **Co-evolving Requirements and Architecture**. *In Journal of Software and Systems Modeling*.	2012	04	3.6
[S3]	D. Garlan, S. Cheng, A. Huang, B. Schmerl, P. Steenkiste. **Rainbow**: **Architecture-Based Self-Adaptation with Reusable Infrastructure**. *In IEEE Computer*	2004	665	3.6
[S4]	D. Garlan, S.W. Cheng, B. Schmerl. **Increasing System Dependability through Architecture-Based Self-Repair**. *In Architecting Dependable Systems*.	2008	138	3.6
[S5]	L. Baresi, R. Heckel, S. Thöne and D. Varró. **Style-based Modeling and Refinement of Service-oriented Architectures**. *In Journal of Software and Systems Modeling*.	2006	82	3.5
[S6]	M. Goedicke and U. Zdun. **Piecemeal Legacy Migrating with an Architectural Pattern Language**. In *Journal of Software Maintenance: Research and Practice*.	2002	29	3.4
[S7]	A. Hoek, M. Rakic, R. Roshandel and N. Medvidovic. **Taming Architectural Evolution**. *In Joint 8th European Software Engineering Conference and 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering*.	2001	72	3.3
[S8]	C. E. Cuesta, E. Navarro, D. E. Perry, C. Roda. **Evolution Styles**: **Using Architectural Knowledge as an Evolution Driver**. *In Journal of Software: Evolution and Process*.	2012	0	3.3
[S9]	P. Bengtsson and Jan Bosch. **Architecture Level Prediction of Software Maintenance**. *In 3rd European Conference on Software Maintenance and Reengineering*.	1999	107	3.3

(Continues)

DOI: 10.1002/smr

APPENDIX
(Continued).

[S10]	N. Lassing, D. Rijsenbrij, H. v. Vliet. **How Well can we Predict Changes at Architecture Design Time**. *In Journal of Systems and Software*.	2003	31	3.3
[S11]	D. Garlan, J. M. Barnes, B. Schmerl, O. Celiku. **Evolution Styles**: **Foundations and Tool Support for Software Architecture Evolution**. *In Joint Working IEEE/IFIP Conference on Software Architecture 2009 & European Conference on Software Architecture*.	2009	47	3.2
[S12]	C. Hentrich and U. Zdun. **Patterns for Process-Oriented Integration in Service-Oriented Architectures**. *In 11th European Conference on Pattern Languages of Programs*.	2006	42	3.2
[S13]	O. L. Goaer. D. Tamzalit, M. Oussalah, A. D. Seriai. **Evolution Shelf**: **Reusing Evolution Expertise within Component-Based Software Architectures**. *In IEEE International Computer Software and Applications Conference*.	2008	13	3.0
[S14]	O. Zimmermann, U. Zdun, T. Gschwind, F. Leymann. **Combining Pattern Languages and Reusable Architectural Decision Models into a Comprehensive and Comprehensible Design Method**. *In 7th Working IEEE/IFIP Conference on Software Architecture*.	2008	27	3.0
[S15]	U. Zdun and S. Dustdar. **Model-Driven and Pattern-Based Integration of Process-Driven SOA Models**. *In International Journal Business Process Integration and Management, 2007*.	2007	41	2.9
[S16]	H. Gomaa, M. Hussein. **Software Reconfiguration Patterns for Dynamic Evolution of Software Architectures**. *In 4th Working IEEE/IFIP Conference on Software Architecture*.	2004	50	2.8
[S17]	X. Dong, M. W. Godfrey. **Identifying Architectural Change Patterns in Object-Oriented Systems**. *In 16th IEEE International Conference on Program Comprehension*.	2008	08	2.8
[S18]	N. Gui and V. De. Florio, **Towards Meta-Adaptation Support with Reusable and Composable Adaptation Components**. *In EEE Sixth International Conference on Self-Adaptive and Self-Organizing Systems*.	2012	0	2.8
[S19]	H. Gomaa, K. Hashimoto, M. Kim, S. Malek, D. A. Menascé. **Software Adaptation Patterns for Service-oriented Architectures**. *In ACM Symposium on Applied Computing*.	2010	19	2.7
[S20]	N. Sadou, D. Tamzalit, M. Oussalah. **How to Manage Uniformly Software Architecture at Different Abstraction Levels**. *In 24th International Conference on Conceptual Modeling*.	2005	08	2.4
[S21]	D. Tamzalit, T. Mens. **Guiding Architectural Restructuring through Architectural Styles**. *In 17th IEEE International Conference and Workshops on Engineering of Computer-Based Systems*.	2010	10	2.1
[S22]	O. Barais, L. Duchien, A. Le Meur. **A Framework to Specify Incremental Software Architecture Transformations**. *In 31st EUROMICRO Conference on Software Engineering and Advanced Applications*.	2005	17	2.1
[S23]	D. Tamzalit, M. Oussalah, O. L. Goaer, A. d. Seriai. **Updating Software Architectures**: **A Style-based Approach**. In International Conference on Software Engineering Research and Practice.	2006	07	2.0
[S24]	Le. Goaer, M. Oussalah, D. Tamzalit. **Reusing Evolution Practices onto Object-Oriented Designs**: **An Experiment with Evolution Styles**. *In 19th International Conference on Software Engineering and Data Engineering*.	2010	0	2.0
[S25]	J. C. Georgas R. N. Taylor. **Towards a Knowledge-Based Approach to Architectural Adaptation Management**. *In 1st ACM SIGSOFT Workshop on Self-managed Systems*.	2004	41	1.9
[S26]	J. C. Georgas, A. v.d. Hoek, R. N. Taylor. **Architectural Runtime Configuration Management in Support of Dependable Self-Adaptive Software**. *In Workshop on Architecting Dependable Systems*.	2005	17	1.9
[S27]	I. Côté, M. Heisel, I. Wentzlaff. **Pattern-Based Evolution of Software Architectures**. *In European Conference on Software Architecture*.	2007	02	1.8
[S28]	E. Truyen and W. Joosen. **Towards an Aspect-oriented Architecture for Self-adaptive Frameworks**. *In Workshop on Aspects, Components, and Patterns for Infrastructure Software*.	2008	04	1.7

(Continues)

DOI: 10.1002/smr

APPENDIX
(Continued).

[S29]	A. Ahmad. P. Jamshidi, C. Pahl. **Graph-based Pattern Identification from Architecture Change Logs**. *In 10th International Workshop on System/Software Architectures*.	2012	02	1.6
[S30]	P. Jamshidi, C. Pahl. **Business Process and Software Architecture Model Co-evolution Patterns**. *In Workshop on Modeling in Software Engineering*.	2012	01	1.5
[S31]	A. Ahmad. P. Jamshidi, M. Arshad, C. Pahl. **Graph-based Implicit Knowledge Discovery from Architecture Change Logs**. *In 7th Workshop on SHaring and Reusing Architectural Knowledge*.	2012	0	1.5
[S32]	J. C. Georgas R. N. Taylor. **An Architectural Style Perspective on Dynamic Robotic Architectures**. *In IEEE 2nd International Workshop on Software Development and Integration in Robotics*.	2007	02	1.5

ACKNOWLEDGEMENTS

The authors would like to thank Dr Jim Buckely (affiliated with: Lero—the Irish Software Engineering Research Centre, University of Limerick, Ireland) and Bardia Mohabbati (affiliated with: Simon Fraser University, Canada) for their feedback and thoughtful suggestions throughout the development and evaluation of the review protocol. This work was supported, in part, by Science Foundation Ireland grant 10/CE/I1855 to Lero—the Irish Software Engineering Research Centre (www.lero.ie). The research work described in this paper was partly supported by the Irish Centre for Cloud Computing and Commerce (IC4), a national technology centre funded by Enterprise Ireland and the Irish Industrial Development Authority

REFERENCES

1. Mens T, Demeyer S. Software Evolution. Springer: Springer Berlin Heidelberg, 2008: 1–11.
2. Lehman M, Laws of software evolution revisited. *Software Process Technology*, LNCS 1996.
3. Medvidovic N, Rosenblum D, Taylor RN. A language and environment for architecture-based software development and evolution. *International Conference on Software Engineering*, 1999.
4. Yskout K, Scandariato R, Joosen W. Change patterns: co-evolving requirements and architecture. *Journal of Software and Systems Modeling* 2012.
5. Moghadam I, Cinnéide M. Automated refactoring using design differencing. *16th European Conference on Software Maintenance and Reengineering*, 2012.
6. Bradbury J, Cordy J, Dingel J, Wermelinger M. A classification of formal specifications for dynamic software architectures. *International Workshop on Self-Managed Systems*, 2004.
7. Oreizy P, Gorlick M, Taylor RN, Heimbigner D, Johnson G, Medvidovic N, Quilici A, Rosenblum DS, Wolf A. An architecture-based approach to self-adaptive software. *IEEE Journal of Intelligent Systems and Their Applications*.
8. Chapin N, Hale JE, Kham KM, Ramil JF, Tan WG. Types of software evolution and software maintenance. *Journal of Software Maintenance Research and Practice* 2001; **13**(1).
9. Jamshidi P, Ghafari M, Ahmad A, Pahl C. A framework for classifying and comparing architecture centric software evolution. *17th European Conference on Software Maintenanace and Reengineering*, 2013.
10. Williams BJ, Carver JC. Characterizing software architecture changes: a systematic review. *Information and Software Technology* 2010; **52**(1):31–51.
11. Mens K, Mens T, Wouters B, Wuyts R. Managing unanticipated evolution of software architectures. *Workshop on Object-Oriented Technology*, LNCS, 1999.
12. Garlan D, Cheng S, Huang A, Schmerl B, Steenkiste P. Rainbow: architecture-based self-adaptation with reusable infrastructure. *IEEE Computer* 2004; **37**:46–54.
13. Barnes JM, Garlan D, Schmerl B. Evolution styles: foundations and models for software architecture evolution. *Journal of Software and Systems Modeling* 2012. DOI: 10.1007/s10270-012-0301-9
14. Breivold HP, Crnkovic I, Larsson M. A systematic review of software architecture evolution research. *Information and Software Technology* 2012; **54**(1):16–40.
15. Ganek AG, Corbi TA. The dawning of the autonomic computing era. *IBM Systems Journal* 2003; **42**(1):5–18
16. Baresi L, Nitto ED, Ghezzi C. Toward open-world software: issue and challenges. *IEEE Computer* 2006:36–43.
17. Li Z, Liang P, Avgeriou P. Application of knowledge-based approaches in software architecture: a systematic mapping study. *In Information and Software Technology* 2013; **55**(5):777–794.
18. Babar MA, Dingsøyr T, Lago P, Vliet HV. Software architecture knowledge management: theory and practice, Springer Heidelberg, 2009.
19. Joint 10th Working IEEE/IFIP Conference on Software Architecture and 6th European Conference on Software Architecture. Available from: http://www.wicsa.net/ [Accessed on 22 February 2013]
20. Workshop on sharing and reusing architectural knowledge. Available from: http://www.shark-workshop.org/ [Accessed on 22 February 2013]

21. WICSA Wiki. Available from: http://wwwp.dnsalias.org/wiki/WICSA_2011. [Accessed on 22 February 2013]
22. Goaer OL. Tamzalit D, Oussalah M, Seriai AD. Evolution shelf: reusing evolution expertise within component-based software architectures. *IEEE International Computer Software and Applications Conference*, IEEE: Los Alamitos 2008; 311–318.
23. Hoek A, Rakic M, Roshandel R, Medvidovic N. Taming architectural evolution. *In Joint 8th European Software Engineering Conference and 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, Vienna: Austria, 2001.
24. Brereton P, Kitchenham B, Budgen D, Turner M, Khalil M. Lessons from applying the systematic literature review process within the software engineering domain. *Journal of Systems and Software* 2007; **80**(4):571–583.
25. Buckley J, Mens T, Zenger M, Rashid A, Kniesel G. Towards a taxonomy of software change. *Journal of Software Maintenance and Evolution: Research and Practice* 2005:309–332.
26. Jamshidi P, Ahmad A, Pahl C. Cloud migration research: a systematic review. IEEE Transactions on Cloud Computing, 2013.
27. Côté I, Heisel M, Wentzlaff I. Pattern-based evolution of software architectures. *In European Conference on Software Architecture*, 2007: 29–43.
28. Gui N, De Florio V. Towards meta-adaptation support with reusable and composable adaptation components. *In IEEE Sixth International Conference on Self-Adaptive and Self-Organizing Systems*, IEEE ,2012.
29. Dong X, Godfrey MW. Identifying architectural change patterns in object-oriented systems. *16th IEEE International Conference on Program Comprehension*, Amsterdam: The Netherlands, 2008.
30. Ahmad A, Jamshidi P, Pahl C. Graph-based pattern identification from architecture change logs. *In 10th International Workshop on System/Software Architectures*, CAiSE workshops: Gdansk, Poland, 2012.
31. Ahmad A, Jamshidi P, Arshad M, Pahl C. Graph-based implicit knowledge discovery from architecture change logs. *In 7th Workshop on SHaring and Reusing Architectural Knowledge*, WICSA/ECSA Companion Volume: Helsinki, Finland, 2012.
32. Bengtsson P, Bosch J. Architecture level prediction of software maintenance. *In 3rd European Conference on Software Maintenance and Reengineering*, the Netherlands, 1999.
33. Lassing N, Rijsenbrij D, Vliet H. How well can we predict changes at architecture design time. *Journal of Systems and Software* 2003; **65**(2):141–153.
34. Ahmad A, Jamshidi P. A classification and comparison of software architecture evolution reuse-knowledge. Available from: http://www.computing.dcu.ie/~pjamshidi/SLR/SLR-ERK.html.
35. Stammel J, Durdik Z, Krogmann K, Weiss R, Koziolek H. Software evolution for industrial automation systems: literature overview. *Karlsruhe Reports in Informatics* 2011. Available from: http://sdqweb.ipd.kit.edu/publications/pdfs/stammel2011a.pdf, [Accessed on 12 March 2013]
36. Garg AX, Hackam D, Tonelli M. Systematic review and meta-analysis: when one study is just not enough. *Clinical Journal of the American Society of Nephrology* 2008; **3**(1):253–260.
37. Bennett KH, Rajlich V. Software maintenance and evolution: a roadmap. In ICSE'2000 - Future of Software Engineering, Limerick, 2000, 73–87.
38. Slyngstad OPN, Conradi R, Babar MA, Clerc V, van Vliet H. Risks and risk management in software architecture evolution: an industrial survey. *In 15th Asia-Pacific Software Engineering Conference*, 2008.
39. Zhang H, Babar MA. Systematic reviews in software engineering: an empirical investigation. *Information and Software Technology* 2013; **5**(7):1341–1354.
40. Petticrew M, Roberts H. Systematic Reviews in the Social Sciences: A Practical Guide. Blackwell: Oxford, 2006.
41. Kazman R, Woods S, Carriere J. Requirements for integrating software architecture and reengineering models: CORUM II. *In Working Conference on Reverse Engineering*, 1998; 154–163.
42. Winter A, Ziemann J. Model-based migration to service-oriented architectures. *In International Workshop on SOA Maintenance and Evolution*, In H. Sneed (ed.). CSMR 2007 Workshops, 2007; 107–110.
43. Razavian M, Lago P. Towards a conceptual framework for legacy to SOA migration. *In Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops*, WESOA 2009.
44. Ulrich WM, Newcomb P. Information Systems Transformation: Architecture-Driven Modernization Case Studies. Morgan Kaufmann Publishers Inc: Elsevier, 2010. ISBN: 978-0-12-374913-0
45. Dancy E, Cordy JR, James R. STAC: software tuning panels for autonomic control. *In 2006 conference of the Center for Advanced Studies on Collaborative Research*. 2006; 146–160.
46. Ahmad A, Jamshidi P, Pahl C. A framework for acquisition and application of software architecture evolution knowledge. *ACM SIGSOFT Software Engineering Notes* 2013; **38**(4).
47. Brinkkemper S. Method engineering: engineering of information systems development methods and tools. In Information and Software Technology, 1996; **38**(4):275–280.
48. Barnes JM, Garlan D. Challenges in developing a software architecture evolution tool as a plug-in. *In 3rd Workshop on Developing Tools as Plugin-Ins*, ICSE WorkshopsL: USA, 2013.
49. Pahl C, Giesecke S, Hasselbring W. Ontology-based modelling of architectural styles. *Information and Software Technology* 2009; (12):1739–1749.
50. Gamma E, Helm R, Johnson R, Vlissides J. Design patterns: abstraction and reuse of object-oriented design. *In Object-Oriented Programming (ECOOP)*, 1993.

DOI: 10.1002/smr

AUTHORS' BIOGRAPHIES

Aakash Ahmad is a PhD candidate in Software Engineering at Lero—the Irish Software Engineering Research Centre, Dublin City University, Ireland. He received the BSc degree in Software Engineering from the International Islamic University, Islamabad, Pakistan. He was a tutor in Dublin City University, Ireland, and a software engineer at Elixir Technologies, Pakistan. His research interests include architecture-centric software evolution, and acquisition and application of reuse knowledge and expertise to evolve software systems.

Pooyan Jamshidi is a PhD candidate in the School of Computing, Faculty of Engineering and Computing at Dublin City University. He received the BS and MS degrees in Computer Science and Systems Engineering from Amirkabir University of Technology (Tehran Polytechnic) in 2003 and 2006, respectively. He is currently with IC4–The Irish Centre for Cloud Computing and Commerce. His general research interests are in the field of software engineering and his focus lies predominantly in the areas of software architecture, software evolution and self-adaptive software. He is a reviewer for five software engineering journals and conferences.

Dr. Claus Pahl is a senior lecturer at the School of Computing at Dublin City University. He is the architecture theme lead at the National Cloud Computing Technology Centre IC4. His research focuses on software architecture and cloud service engineering. His specific interests include dynamic architectures (dynamic context dependent composition, service models at runtime; constraints monitoring), business process and service architecture integration (layered architectures for process and architecture configuration), and mediation and integration in cloud computing (interoperability for on-demand architectures; multitenancy SOA and policy-based governance). He has published more than 240 papers. He is a reviewer for more than 30 journals and helped organizing more than 100 conferences and workshops. He is on the editorial board of four journals, and he has acted as a panel member and an evaluator for various research schemes worldwide. He is a member of the IEEE.

DOI: 10.1002/smr