The Journal of Systems and Software 111 (2016) 157–184

Contents lists available at ScienceDirect

The Journal of Systems and Software

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

A systematic mapping study on the combination of software architecture and agile development

Chen Yang [a,b], Peng Liang [a,*], Paris Avgeriou [b]

[a] State Key Lab of Software Engineering, School of Computer, Wuhan University, 430072 Wuhan, China
[b] Department of Mathematics and Computing Science, University of Groningen, Nijenborgh 9, 9747 AG Groningen, The Netherlands

ARTICLE INFO

Article history:
Received 18 February 2015
Revised 16 September 2015
Accepted 20 September 2015

Keywords:
Software architecture
Agile development
Architecting approach

ABSTRACT

Context: Combining software architecture and agile development has received significant attention in recent years. However, there exists no comprehensive overview of the state of research on the architecture-agility combination.
Objective: This work aims to analyze the combination of architecture and agile methods for the purpose of exploration and analysis with respect to architecting activities and approaches, agile methods and practices, costs, benefits, challenges, factors, tools, and lessons learned concerning the combination.
Method: A systematic mapping study (SMS) was conducted, covering the literature on the architecture-agility combination published between February 2001 and January 2014.
Results: Fifty-four studies were finally included in this SMS. Some of the highlights: (1) a significant difference exists in the proportion of various architecting activities, agile methods, and agile practices employed in the combination. (2) none of the architecting approaches has been widely used in the combination. (3) there is a lack of description and analysis regarding the costs and failure stories of the combination. (4) twenty challenges, twenty-nine factors, and twenty-five lessons learned were identified.
Conclusions: The results of this SMS help the software engineering community to reflect on the past thirteen years of research and practice on the architecture-agility combination with a number of implications.

1. Introduction

Software Architecture (SA)[1] represents "*the fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution*" (ISO, 2011). SA acts as a high-level design and as a means of performing complicated trade-offs (e.g., quality requirements) between stakeholders of software-intensive systems (Bass et al., 2012). Critics of traditional architecting processes argue that they tend to employ a Big Design Up-Front (BDUF) approach leading to excessive documentation and implementation of possibly unneeded features, which introduce additional development effort (Abrahamsson et al., 2010). As an alternative to BDUF, agile development is proposed, which mainly focuses on adapting to change and delivering products of high quality through simple work-processes (Dingsøyr et al., 2010).

Freudenberg and Sharp listed the top 10 burning research questions in the agile development community collected from about 300 practitioners at the XP 2010 conference, and the question "*Architecture and agile - how much design is enough for different classes of problem?*" is ranked in the second position (Freudenberg and Sharp, 2010). Many approaches, techniques, processes, and tools have been proposed and developed to support either the use of SA or the use of agile methods in software development. However, ways to combine them is a challenging issue, which has been heavily debated over the past years (Abrahamsson et al., 2010). To understand how SA and agile development can be used together, we conducted a systematic mapping study (SMS) to collect primary studies on **using software architecture in agile development as well as using agile methods in architecture-centric development**. This study aims at identifying available evidence on various aspects of this topic (for details see Section 3.1), and spotting gaps in the application of architecture in agile development and the other way round, i.e., the application of agile methods and practices used within architecture-centric development.

An SMS aims at mapping the evidence at a high level for a specific topic, and is particularly suitable when the studied topic is very broad (Kitchenham and Charters, 2007). Systematic literature review (SLR) is another form of secondary study, which provides "*a means of identifying, evaluating, and interpreting all available research relevant to a particular research question*" (Kitchenham and Charters, 2007). We

* Corresponding author. Tel.: +86 27 68776137; fax: +86 27 68776027.
E-mail address: liangp@whu.edu.cn (P. Liang).
[1] For readability and clarity, we list all the abbreviations used in this paper in Appendix B for reference.

http://dx.doi.org/10.1016/j.jss.2015.09.028
0164-1212/

decided to conduct an SMS because the studied topics (software architecture and agile development) cover very broad areas. However, we did not just simply perform a mapping (classifying primary studies into categories), but also synthesized data from studies (by using grounded theory (Strauss and Corbin, 1998)).

The rest of this paper is structured as follows. Section 2 provides the context of this SMS, i.e., architecting activities and agile practices. Section 3 details the mapping study process with the research questions. Section 4 presents the results of the research questions. Section 5 further discusses the study results with their implications for researchers and practitioners. Section 6 presents the threats to validity. Section 7 concludes this work.

2. Context

The contextual elements of this SMS include architecture-centric development and agile development. We discuss the topic of architecture-centric development through architecting activities and agile development through agile practices. We summarize eleven architecting activities in Section 2.1, which are collected from existing literature, and briefly review the practices of agile development in Section 2.2.

2.1. Architecting activities

The architecting process is comprised of a number of specific architecting activities (covering the entire architectural lifecycle) and a number of general architecting activities (supporting the specific activities). The specific activities are listed and described below:

- **Architectural Analysis (AA)** is aimed at defining the problems an architecture must solve. The outcome of this activity is a set of architecturally significant requirements (ASRs) (Hofmeister et al., 2007).
- **Architectural Synthesis (AS)** proposes candidate architecture solutions to address the ASRs collected in AA, thus this activity moves from the problem to the solution space (Hofmeister et al., 2007).
- **Architectural Evaluation (AE)** ensures that the architectural design decisions made are the right ones, and the candidate architectural solutions proposed in AS are measured against the ASRs collected in AA (Hofmeister et al., 2007).
- **Architectural Implementation (AI)** realizes the architecture by creating a detailed design (Tang et al., 2010).
- **Architectural Maintenance and Evolution (AME):** Architectural maintenance is to change an architecture for the purpose of fixing faults (ISO, 2006, ISO, 2011) and architectural evolution is to respond to new requirements at architectural level (Postma et al., 2004). In this SMS, we simply considered architectural maintenance and architectural evolution as one activity, in which an architecture is changed either to fix faults or to implement new requirements.

An architecting process is composed of the five specific activities mentioned above (Hofmeister et al., 2007; Tang et al., 2010). There are also general architecting activities (e.g., Architectural Description) identified in (Li et al., 2013) that are meant to support the specific activities. For example, Architectural Description (ADp) can be used to specify and document the candidate architecture solutions during Architectural Synthesis (AS). The general activities are described as follows:

- **Architectural Recovery (AR)** is used to extract the current architecture of a system from the system's implementation (Medvidovic and Jakobac, 2006).
- **Architectural Description (ADp)** is used to describe the architecture with a set of architectural elements (e.g., architecture views). This activity can help stakeholders (e.g., architects) to understand the system, and improve the communication and cooperation among stakeholders (ISO, 2011).
- **Architectural Understanding (AU)** is used to comprehend the architectural elements (e.g., architectural decisions) and their relationships in an architecture design (Li et al., 2013).
- **Architectural Impact Analysis (AIA)** is used to identify the architectural elements, which are affected by a change scenario (Bengtsson et al., 2004). The analysis results include the components in architecture that are affected directly, as well as the indirect effects of changes to the architecture (Bengtsson et al., 2004).
- **Architectural Reuse (ARu)** aims at reusing existing architectural design elements, such as architecture frameworks, decisions, and patterns in the architecture of a new system (IEEE, 2010).
- **Architectural Refactoring (ARf)** aims at improving the architectural structure of a system without changing its external behavior (Babar et al., 2013; Fowler et al., 1999).

2.2. Agile practices

Agile development aims at stripping away, as much as possible, the effort-intensive activities in software development (Erickson et al., 2005), and focuses on quick response to various changes of a project (Erickson et al., 2005). Agile practices are in general practices used to support agile development. An example of such an agile practice is iterative and incremental development, which focuses on small releases and a planning strategy based on a release plan and an iteration plan (Augustine, 2005). Agile practices in principle adhere to the values proposed in the agile manifesto (16).

To the best of our knowledge, there is no work that systematically summarizes, analyzes, and classifies all existing agile practices. In literature we found more than a hundred such practices. We collected the top 20 of these agile practices (listed in Table 1) according to the number of articles that discuss them (as shown in the "Sources" column).

This list of agile practices as well as the related literature presented above is not comprehensive; there are potentially many other papers that discuss similar or other agile practices. Furthermore, the aforementioned agile practices may be partially overlapping as listed in the parentheses of Table 1. For example, "Direct Interaction with Customer" in (Begel and Nagappan, 2007) is the same as "On-Site Customer" in (Silva et al., 2014). Considering this, we include "agile practice" as a data item to be extracted in selected studies (see Section 3.2.3) as shown in Table 5. Note that we only focus on the agile practices related to software architecture in this SMS.

3. Mapping study process

3.1. Research questions

The goal of this SMS, formulated using Goal-Question-Metric approach (Basili et al., 1994), is to **analyze** the combination of software architecture and agile methods **for the purpose of** exploration and analysis **with respect to** architecting activities and approaches, agile methods and practices, costs, benefits, challenges, factors, tools, and lessons learned **from the point of view of** researchers and practitioners **in the context of** software development.

We decomposed the goal into nine research questions (RQs) shown in Table 2. The answers of these RQs can be readily linked to the objective of this mapping study: an understanding of how architecting can be used in agile development (RQ1, RQ2), which agile methods and practices can be used with architecture (RQ3, RQ4), the costs, benefits, challenges, and the available tools of the architecture-agility combination (RQ5, RQ6, RQ8), the factors which may have an impact on the combination (RQ7), and the lessons learned from the combination (RQ9).

Table 1
Top 20 agile practices.

Agile practice	Sources
Test-Driven Development (Early Testing, Test-First Development)	Begel and Nagappan (2007); Jalali and Wohlin (2012); Rodríguez et al. (2012); Sfetsos and Stamelos (2010); VersionOne (2015); Silva et al. (2014); Hossain et al. (2009); Larman (2003)
Pair Programming	Begel and Nagappan (2007); Jalali and Wohlin (2012); Rodríguez et al. (2012); Sfetsos and Stamelos (2010); VersionOne (2015); Silva et al. (2014); Larman (2003); Pikkarainen et al. (2008)
Continuous Integration (Frequent Integration)	Begel and Nagappan (2007); Jalali and Wohlin (2012); Rodríguez et al. (2012); VersionOne (2015); Silva et al. (2014); Petersen and Wohlin (2010); Larman (2003); Pikkarainen et al. (2008)
Daily Standup	Begel and Nagappan (2007); Jalali and Wohlin (2012); Rodríguez et al. (2012); VersionOne (2015); Silva et al. (2014); Larman (2003); Hossain et al. (2009); Pikkarainen et al. (2008)
Refactoring (Code Refactoring, Frequent Refactoring)	Jalali and Wohlin (2012); Rodríguez et al. (2012); Sfetsos and Stamelos (2010); VersionOne (2015); Petersen and Wohlin (2010); Larman (2003); Hossain et al. (2009)
On-Site Customer (Active Customer Participation, Direct Interaction with Customer, Whole Team, Constant Customer Feedback)	Begel and Nagappan (2007); Jalali and Wohlin (2012); Rodríguez et al. (2012); Sfetsos and Stamelos (2010); Silva et al. (2014); Petersen and Wohlin (2010); Larman (2003)
Backlog (Product Backlog, Flexible Product Backlog, Task List, Prioritized Backlog)	Jalali and Wohlin (2012); Rodríguez et al. (2012); VersionOne (2015); Silva et al. (2014); Petersen and Wohlin (2010); Larman (2003)
Collective Code Ownership (Team Code Ownership)	Begel and Nagappan (2007); Jalali and Wohlin (2012); Rodríguez et al. (2012); VersionOne (2015); Petersen and Wohlin (2010); Larman (2003)
Retrospective (Iteration Retrospective)	Jalali and Wohlin (2012); Rodríguez et al. (2012); VersionOne (2015); Silva et al. (2014); Hossain et al. (2009); Pikkarainen et al. (2008)
Coding Standard	Begel and Nagappan (2007); Jalali and Wohlin (2012); Rodríguez et al. (2012); VersionOne (2015); Hossain et al. (2009); Larman (2003)
User Story (Story Mapping)	Begel and Nagappan (2007); Jalali and Wohlin (2012); VersionOne (2015); Silva et al. (2014); Pikkarainen et al. (2008)
Planning Game (Release Planning Game, Iteration Planning Game)	Jalali and Wohlin (2012); Sfetsos and Stamelos (2010); VersionOne (2015); Larman (2003); Pikkarainen et al. (2008)
Iteration Planning (Sprint Planning)	Jalali and Wohlin (2012); Rodríguez et al. (2012); VersionOne (2015); Hossain et al. (2009); Pikkarainen et al. (2008)
Iterative Development (Iteration, Short Iteration, Iterations and Increments)	Jalali and Wohlin (2012); Rodríguez et al. (2012); VersionOne (2015); Silva et al. (2014); Petersen and Wohlin (2010)
Open Work Area (Open Office Space, Common Room, Sit Together)	Jalali and Wohlin (2012); Rodríguez et al. (2012); VersionOne (2015); Larman (2003); Pikkarainen et al. (2008)
Simple Design (Incremental Design)	Begel and Nagappan (2007); Jalali and Wohlin (2012); Petersen and Wohlin (2010); Larman (2003)
Unit Testing	Jalali and Wohlin (2012); Rodríguez et al. (2012); VersionOne (2015); Larman (2003)
Iteration Review (Sprint Review/Demo)	Jalali and Wohlin (2012); VersionOne (2015); Silva et al. (2014); Hossain et al. (2009)
System Metaphor	Begel and Nagappan (2007); Rodríguez et al. (2012); Larman (2003)
Short Release (Small Release, Frequent Release, Frequent and Incremental Delivery)	Begel and Nagappan (2007); Rodríguez et al. (2012); Larman (2003)

Table 2
Research questions and their description.

Research question	Description
RQ1: Which architecting activities can be used in agile development?	The architecting process is comprised of various architecting activities (e.g., Architectural Analysis, Architectural Description). We want to know which such activities can be used in agile development.
RQ2: How architecting can be practiced in agile development?	There are many architecting approaches (e.g., ATAM (Kazman et al., 1998)) proposed and practiced in software development. We want to know which specific approaches can be used in practicing architecture in agile development.
RQ3: Which agile development methods can be used with architecture?	There are many agile methods (e.g., XP and Scrum) which have been used in software development. We want to know which of the methods can be used together with architecture.
RQ4: Which practices of agile development can be used with architecture?	There are many practices (e.g., "Customer on Site") in agile development. We want to get a clear view of which such practices can be used in the architecture-agility combination.
RQ5: What are the costs and benefits of the architecture-agility combination?	Combining agile methods and architecture leads to certain benefits (e.g., reduced re-engineering costs) but also incurs costs (e.g., development effort). We want to help researchers and practitioners understand and consider the architecture-agility combination in a cost-benefit perspective.
RQ6: What are the challenges in the architecture-agility combination?	There are obstacles (conflicts) in combining agile methods and architecture, e.g., SA implicitly follows a top-down development process while agility represents a bottom-up development and delivery of new features. These challenges can provide research directions for improving the architecture-agility combination.
RQ7: What are the factors that impact the success of applying the architecture-agility combination?	Whether an architecture-agility combination is successful or not may be impacted by certain factors (e.g. contextual factors) in software development. Note that Chen and Babar identified the contextual factors that impact architecture emergence in agile development (Chen and Babar, 2014). Our SMS covers those contextual factors as well as other factors that impact the success of applying the architecture-agility combination. From this perspective, our study has a different focus and a larger scope compared with the study by Chen and Babar (Chen and Babar, 2014).
RQ8: What tools are available to support the architecture-agility combination?	There are many tools that can be used for both architecture-centric development and agile development. We want to make practitioners aware of the tools that they may employ to facilitate the architecture-agility combination.
RQ9: What are the lessons learned from the architecture-agility combination?	The lessons learned cover the experience of the authors and potential good practices, when adopting architecture-agility combination.

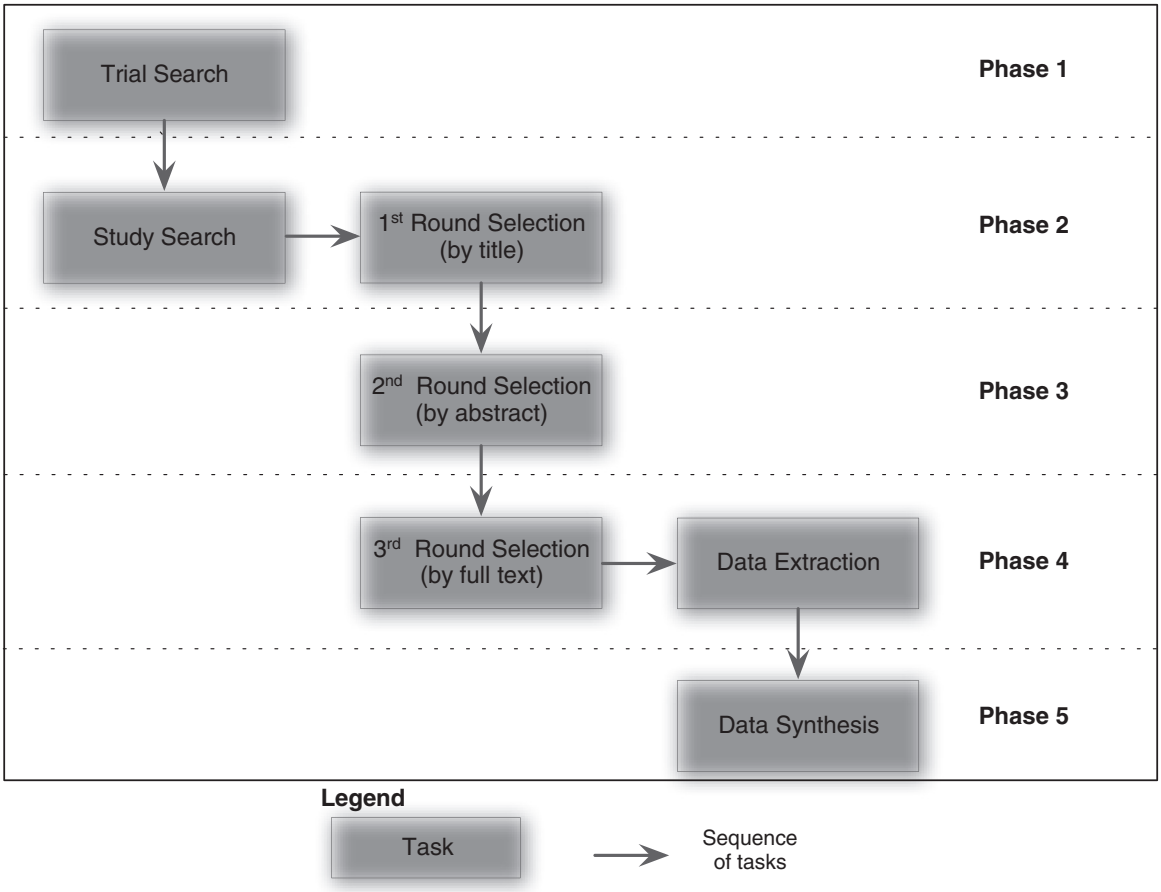

Fig. 1. Execution procedure of the SMS in phases and tasks.

Table 3
Electronic databases included in the mapping study.

#	Electronic database	Search terms used in
ED1	ACM Digital Library	Paper title, abstract
ED2	IEEE Explore	Paper title, keywords, abstract
ED3	Science Direct	Paper title, keywords, abstract
ED4	Springer Link	Paper title, abstract
ED5	Wiley InterScience	Paper title, abstract
ED6	EI Compendex	Paper title, abstract
ED7	ISI Web of Science	Paper title, keywords, abstract

3.2. Mapping study execution

This study follows the guidelines proposed in (Petersen et al., 2008). Fig. 1 shows the execution procedure of this SMS, which is comprised of seven tasks in five phases. Before the formal search, we executed a trial search (Phase 1) because the search engines provided by the databases (see Table 3) were different, and we could not employ the same search strategy in all databases. The trial search was also used to improve the search terms (see Section 3.2.1.2). We searched in seven databases as listed in Table 3 and used the search terms defined in Section 3.2.1.2. We conducted three rounds of study selection, including selection by title (1st round), selection by abstract (2nd round), and selection by full text (3rd round). To make the process of this SMS more efficient, some tasks were conducted simultaneously in Phase 2 and Phase 4 (see Fig. 1):

- Phase 2, study search and the first round study selection (by title): When searching in the electronic databases, we decided whether a paper should be included or not based on its title.
- Phase 4, the third round study selection (by full text) and study data extraction: When judging whether a paper should be included or not by reading its full text, we also extracted the data from the paper if it was selected.

In Phase 3, we conducted the 2nd round selection through reading the abstract of the papers, and in Phase 5, the extracted data was further synthesized to answer the specific RQs defined in Section 3.1. These tasks are detailed in the following subsections.

3.2.1. Study search

In this section, we describe the search scope and the search strategy.

3.2.1.1. Search scope. **Time period**: The time period is set between February 2001 when agile manifesto was proposed (Beck, 2014) and January 2014 when we started this SMS.

Electronic databases: The seven electronic databases listed in Table 2 were selected as the sources for the database search. Note that we excluded Google Scholar in this SMS, due to the reason stated in (Chen et al., 2010): it would have much overlap with e.g., ACM (ED1 in Table 3) and IEEE (ED2 in Table 3) on software engineering literature.

3.2.1.2. Search strategy.

(1) We initially identified the search terms based on the study topic (the architecture-agility combination) and the RQs. PICO (Population, Intervention, Comparison, Outcomes) (Kitchenham and Charters, 2007) was then used to construct the search terms presented below. Population includes the terms of agile development methods; we selected the agile

methods that are popularly used according to a recent literature review (Dybå and Dingsøyr, 2008) and surveys on agile development (Ali, 2012; Rodríguez et al., 2012; Stavru, 2014). Dynamic Systems Development Method (DSDM) (Stapleton, 1998) is not included in the population terms, because it was proposed in 1998 before the Agile Manifesto in 2001 (16). Intervention includes the terms about architecture. We initially included the word "design", which is related to architecture, as a search term in the *trial* searches. However, we found that it leads to a large proportion of irrelevant results since "design" is a very general word; hence we decided to exclude the word "design" in the formal search. Note that, regarding the intervention and population terms, we could have done the opposite, i.e., use terms about "architecture" for population, and terms about "agile methods" for intervention, because the topic of this SMS is about the architecture-agility combination; however this does not affect the combination results of search terms when using Boolean AND. Considering this, we did not include "architecture" terms in population and "agile method" terms in intervention for simplicity. The comparison and outcomes parts are not considered in the search terms, because we do not intent to compare agile development methods with other types of development methods and it is difficult to limit the scope of the outcomes in this SMS.

(2) We tried several combinations of search terms in the trial searches. This is because the search engines provided by each database are different, and we could not use the same combination of search terms in all the databases. An improper search may result in many irrelevant results. We tried to find a suitable way of search for each database with trial searches.

The search terms in population and intervention are listed below:

Population: agile, agility, extreme programming, XP, feature driven development, FDD, scrum, crystal, pair programming, test-driven development, TDD, leanness, lean software development, lean development, LSD

Intervention: architecture, architectural, architecting

Boolean OR is used to connect alternate terms, and Boolean AND is used to join major terms for population and intervention. For example, if the electronic search engine supports logic OR and AND, we can use the following query expression to search in the electronic databases (see Table 3):

(agile ***OR*** *agility* ***OR*** *extreme programming* ***OR*** *XP* ***OR*** *feature driven development* ***OR*** *FDD* ***OR*** *scrum* ***OR*** *crystal* ***OR*** *pair programming* ***OR*** *test-driven development* ***OR*** *TDD* ***OR*** *leanness* ***OR*** *lean software development* ***OR*** *lean development* ***OR*** *LSD)* ***AND*** *(architecture* ***OR*** *architectural* ***OR*** *architecting)*

3.2.2. Study selection

In this section, we describe the selection criteria and the process of the three rounds of study selection. One researcher selected studies in the first round, and two researchers selected the studies in parallel in the second and third rounds. The results of the study selection were discussed and any conflicts between the results were resolved, in order to mitigate personal bias in study selection.

3.2.2.1. Selection criteria. Before the study selection, the researchers discussed and reached a consistent understanding of the inclusion and exclusion criteria listed in Table 4. Two researchers also conducted a pilot study on study selection using these criteria and further resolved any disagreement and misunderstandings on these criteria.

3.2.2.2. First round study selection. We used the search strategies described in Section 3.2.1.2 to identify primary studies. At the same time, we checked the titles of the retrieved studies against the selection criteria. We retained the papers for the next round of study selection, if we could not decide through their titles. Note that the search terms described in Section 3.2.1.2 were matched with the titles, keywords, and abstracts in IEEE Explore, Science Direct, and ISI Web of Science. In the other electronic databases, i.e., ACM Digital Library, SpringerLink, Wiley InterScience, and EI Compendex (see Table 3), the search terms were only matched with the titles and abstracts due to the limitation of the databases.

3.2.2.3. Second round study selection. In this round, we read the abstracts of the retained papers and selected papers according to the selection criteria (see Table 4). Results were checked and any disagreements were discussed and resolved among all the researchers. If a resolution was difficult to make, the study was retained for the next round.

3.2.2.4. Third round study selection. In the last round, we read the full papers and used the selection criteria to judge whether the papers should be finally selected or not. The researchers also discussed and reached an agreement about whether a paper should be finally included or not.

3.2.3. Data extraction

We extracted the relevant data items shown in Table 5 from each selected study to answer the nine RQs of this SMS as defined in Section 3.1. The extracted data was recorded on a MS Excel spreadsheet for further synthesis.

3.2.4. Data synthesis

In this phase, the extracted data were synthesized to answer the nine RQs, and the main data were visualized in a map (see Fig. 3). We classified the SA activities into six general and five specific activities (see Section 2.1) as shown in the top of Fig. 1.

We performed a descriptive statistics analysis on the architecting activities, architecting approaches, agile methods, agile practices, factors, and tools, and used Grounded Theory (Strauss and Corbin, 1998) to analyze architecting approaches, agile practices, costs and benefits, challenges, factors, tools, and lessons learned in the

Table 4
Inclusion criteria and exclusion criteria in the mapping study.

Inclusion criteria
I1: A paper that is peer-reviewed.
I2: A paper that concerns the architecture-agility combination.
Exclusion criteria
E1: If a paper is an extended version of a previous paper, the previous version is excluded.
E2: If a paper introduces an approach that uses architecture, but this approach is not applied in agile development, the paper is excluded.
E3: If a paper introduces an approach that uses agile methods, but this approach is not applied in architecture-centric development, the paper is excluded.
E4: If a paper is grey literature (i.e., technical report, work in progress) (Kitchenham and Charters, 2007), the paper is excluded.
E5: If a paper is not written in English, the paper is excluded.

Table 5
Data items to be extracted from primary studies.

#	Data item	Description	Relevant RQ
D1	Index	The ID of the paper	Overview
D2	Title	The title of the paper	Overview
D3	Author list	The full name of all the authors of the paper	Overview
D4	Year	The year when the paper was published	Overview
D5	Venue	The name of the venue where the study was published	Overview
D6	Publication type	Journal, conference, workshop, or book chapter	Overview
D7	Authors' type	Researcher or practitioner or both	Overview
D8	Research questions	The research questions that the paper tries to answer	Overview
D9	Summary	The main content of the paper	Overview
D10	Limitations	Limitations (e.g., approaches proposed in the paper) discussed by the authors	Overview
D11	Architecting activities	The SA activities mentioned in the paper. Examples: architecture analysis, architecture synthesis, architecture evaluation, which are presented in Section 2.1.	RQ1
D12	Architecting approaches	The specific architecting approaches introduced or used in the paper. Example: DCAR (Decision Centric Architecture Reviews) (van Heesch et al., 2014), etc.	RQ2
D13	Agile development methods	The methods used in agile development. Examples: XP, lean software development, etc.	RQ3
D14	Agile practices	The agile practices used in the architecture-agility combination	RQ4
D15	Costs	The costs of the architecture-agility combination discovered or discussed by the authors	RQ5
D16	Benefits	The benefits of the architecture-agility combination discovered or discussed by the authors	RQ5
D17	Challenges	The challenges of the architecture-agility combination, which have not been solved	RQ6
D18	Factors	The factors which may impact the success of the architecture-agility combination discovered or discussed by the authors	RQ7
D19	Tool support	The tools used in the architecture-agility combination	RQ8
D20	Lessons learned	The lessons learned from the architecture-agility combination by the authors	RQ9

architecture-agility combination. In addition to the descriptive statistics analysis and Grounded Theory methods for data synthesis, we also provided some examples to clarify the data synthesis results.

We used descriptive statistics to answer RQ1, RQ2, RQ3, RQ4, RQ7, and RQ8 and Grounded Theory to answer RQ2, RQ4, RQ5, RQ6, RQ7, RQ8, and RQ9 (Adolph et al., 2011; Corbin and Strauss, 2007). Grounded Theory is a research method that generates theories from data (Glaser and Strauss, 2009). In this SMS, Grounded Theory was used to generate types and subtypes from extracted data items (e.g., D18: Factors in Table 5) to answer the specific RQs defined in Section 3.1. The three coding phases in classical Grounded Theory are open coding, selective coding, and theoretical coding (Adolph et al., 2011). Only open coding and selective coding were used in the data synthesis of this SMS, because we do not need to generate theories through this SMS using theoretical coding. Open coding generates codes for incidents that can be clustered into concepts and categories (Glaser and Strauss, 2009). This phase was used to generate code of certain data items (e.g., D18: Factors in Table 5). Selective coding identifies the core category, which explains the greatest variation in the data and around which the emerging theory is built (Glaser and Strauss, 2009). This phase was used to identify e.g., the six factors that impact the architecture-agility combination in this SMS. Grounded Theory was executed in an iterative process, and we refined and adapted the codes and their relationships in each iteration. During the process of using Grounded Theory, we also used Constant Comparison method (Glaser and Strauss, 2009) to compare incidents (i.e., the extracted data) with incidents, and incidents to concepts and categories (e.g., the six factors in Table 15). With this comparison, we (as researchers) needed to consider the similarities and differences of the meaning among the incidents, in order to achieve the best fit to the data (Strauss, 1987). For example, we identified over one hundred factors that may impact the successful architecture-agility combination from the selected studies initially. By using constant comparison, we compared each factor with other factors, and finally came up with the factors in Table 15.

4. Study results

In this section, the results of this SMS are presented to answer the nine RQs in Section 3.1.

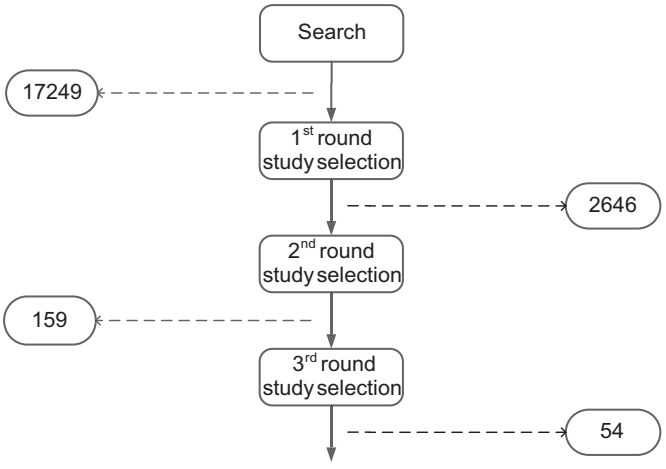

Fig. 2. Results of study search and selection.

4.1. Overview of results

The search and selection results, and the studies distribution are reported in this section.

4.1.1. Search and selection results

As shown in Fig. 2, 17,249 papers were initially retrieved by applying the search terms (see Section 3.2.1.2) before the first round. 2,646 studies were retained before the second round. 159 studies were kept before the third round. 54 studies were finally selected in this SMS.

4.1.2. Studies distribution

In this SMS, we used a bubble chart (see Fig. 3) to show the distribution of the selected studies over architecting activities, agile methods, and time period. Bubbles in the left part of the figure are used to represent the selected studies on certain architecting activities (used in agile development) published in a specific year. Bubbles in the right part of the figure are used to describe the selected studies combining certain agile methods and architecting activities. The numbers shown in a bubble represent the identification number of the selected studies (see Appendix A). In the rest of the paper, *Sn* presents the nth

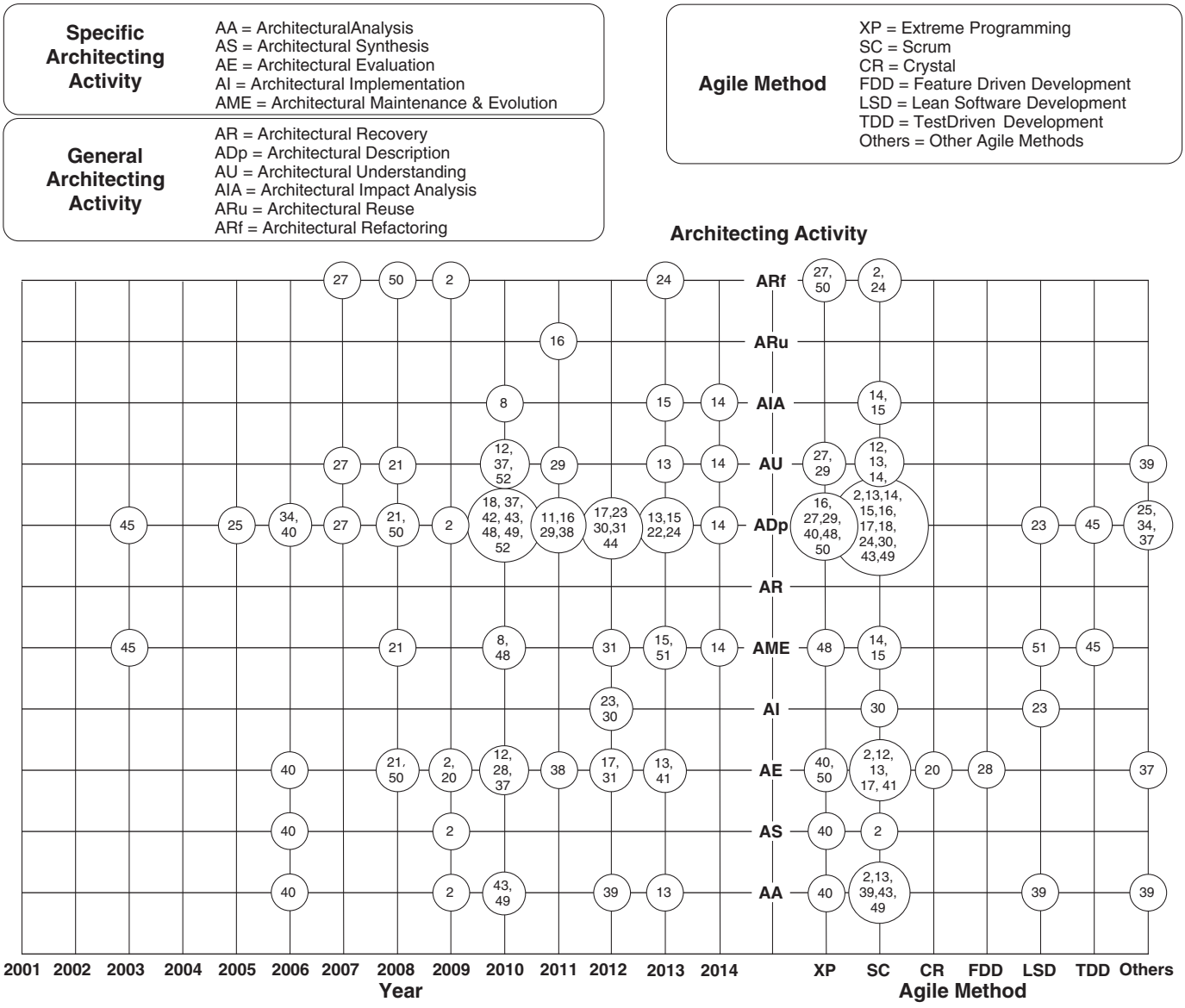

Fig. 3. Bubble chart over architecting activities, agile methods, and time period.

selected study. The detailed information of this map (Fig. 3) is described in Sections 4.2, 4.3, and 4.4.

Table 6 presents the number and proportion of the selected studies over the publication venues. The 54 included studies are distributed over 31 publication venues, suggesting the architecture-agility combination has been a widespread concern in the research community. As shown in Table 6, the leading venues in this study topic are one journal on software practices (IEEE Software, 18.5%) and one conference specialized in agile development (AGILE, 7.4%).

Fig. 4 shows the number of the selected studies published from 2001 to 2013. Note that there is one study published in 2014, which is not shown in Fig. 4, because only one month (January) of 2014 is covered in the search period (see Section 3.2.1.1). The number of studies on the architecture-agility combination has been increasing steadily in the last decade with an exceptional peak in 2010. Table 7 presents the distribution of selected studies over publication venues in 2010. We found that there are five studies from a special issue of IEEE Software and four book chapters from two books in the sixteen studies published in 2010, which explains the peak number of that year.

As shown in Fig. 5, authors of 48.1% of the selected studies (i.e., 26 studies) come from academia, and authors of 42.6% of the selected studies (i.e., 23 studies) work in industry. The rest (9.3%) of the studies (i.e., 5 studies) are co-authored by authors from both academia and industry.

As shown in Table 8, we classified the 54 selected studies into two groups according to their context (i.e., from industry or academia). 44 studies (81.5%) are conducted in an industrial context and 11 studies (20.4%) are based on an academic environment. Note that the survey in [S44] includes both practitioners and researchers as participants, and this study is classified in both "Industry" and "Academia". The result shows that most selected studies (81.5%) are evidenced by industrial projects, surveys, or experience.

4.2. RQ1: Which architecting activities can be used in agile development?

Fig. 3 presents the map of the selected studies over architecting activities, agile methods, and time period. Note that we only extract the data (i.e., data item D11: Architecting activities) if a study explicitly mentioned architecting activities and how they were used in the architecture-agility combination. The detailed information of the studies distribution over architecting activities is shown in Fig. 6. This

Table 6
Number and proportion of the selected studies over publication venues.

Publication venue	Type	No.	%
IEEE Software	Journal	10	18.5
Agile Development Conference (AGILE)	Conference	4	7.4
International Conference on Agile Software Development (XP)	Conference	3	5.6
CrossTalk	Journal	3	5.6
Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture (WICSA/ECSA)	Conference	3	5.6
Agile Software Development	Book	2	3.7
Agility Across Time and Space	Book	2	3.7
International Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)	Conference	2	3.7
European Conference on Software Architecture (ECSA)	Conference	2	3.7
International Conference on Software Engineering (ICSE)	Conference	2	3.7
Information and Software Technology	Journal	1	1.9
Journal of Software: Evolution and Process	Journal	1	1.9
Journal of the Brazilian Computer Society	Journal	1	1.9
Workshop on SHAring and Reusing Architectural Knowledge (SHARK)	Workshop	1	1.9
International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)	Workshop	1	1.9
International Conference on Software Technologies & International Conference on Evaluation of Novel Approaches to Software Engineering (ICSOFT/ENASE)	Conference	1	1.9
International Conference on Information Technology: New Generations (ITNG)	Conference	1	1.9
International Conference on Model Driven Engineering Languages and Systems (MODELS)	Conference	1	1.9
International Requirements Engineering Conference (RE)	Conference	1	1.9
Australian Conference on Software Engineering (ASWEC)	Conference	1	1.9
International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ)	Conference	1	1.9
EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)	Conference	1	1.9
Hawaii International Conference on System Sciences (HICSS)	Conference	1	1.9
AGILE India (AGILE INDIA)	Conference	1	1.9
International Conference on Services Computing (SCC)	Conference	1	1.9
International Conference on Agility (ICAM)	Conference	1	1.9
International Conference on Computational Intelligence and Software Engineering (CiSE)	Conference	1	1.9
International Conference on Information Reuse and Integration (IRI)	Conference	1	1.9
International Conference on Software and System Process (ICSSP)	Conference	1	1.9
International Conference on the Quality of Software Architectures & International Symposium on Architecting Critical Systems (QoSA/ISARCS)	Conference	1	1.9
WRI World Congress on Software Engineering (WCSE)	Conference	1	1.9

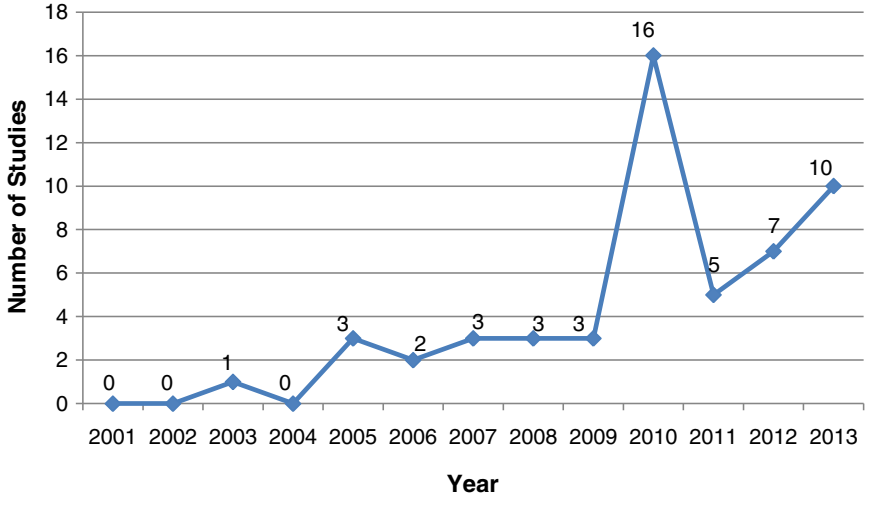

Fig. 4. Number of the selected studies over time period.

figure shows a significant difference in the numbers of the architecting activities used in agile development.

Architectural Description (ADp) is the most frequently discussed architecting activity in the architecture-agility combination (29 out of 54 studies, 53.7%). Existing literature on ADp is mostly related to heavy-weight development methodologies (Hadar et al., 2013). Architecture documents described using these ADp approaches (e.g., views and beyond (Clements et al., 2010)) are usually large and complex, which do not adapt well to the lean, flexible, and minimalistic approach of agile development (Hadar et al., 2013). The agile principle "Working software over comprehensive documentation" (16) also indicates that documentation is not an emphasis in agile development (Abrahamsson et al., 2010; Babar, 2009; Hadar et al., 2013). To this end, some studies aimed to develop suitable ADp approaches for agile development, such as Abstract Architecture Specification Document [S22], Agile Architecture Line Approach and Model [S25], and Developer Stories [S27]. These ADp approaches are further described in detail in Section 4.3 (answer to RQ2).

Architectural Evaluation (AE) is the second most frequently discussed architecting activity in the architecture-agility combination

Table 7
Number of the studies over publication venues in 2010.

Publication venue	Type	No.
IEEE Software	Journal	6
Agile Software Development	Book	2
Agility Across Time and Space	Book	2
European Conference on Software Architecture (ECSA)	Conference	2
CrossTalk	Journal	1
International Conference on Model Driven Engineering Languages and Systems (MODELS)	Conference	1
International Requirements Engineering Conference (RE)	Conference	1
International Conference on Computational Intelligence and Software Engineering (CiSE)	Conference	1

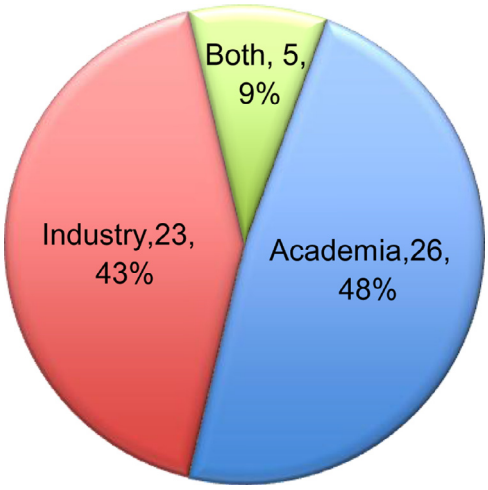

Fig. 5. Number and proportion of the selected studies over author types.

(13 out of 54 studies, 24.1%). AE ensures that the architectural design decisions made are the right ones (Hofmeister et al., 2007), while practitioners of agile methods usually see limited value from the customers' perspective in the up-front architecture design and evaluation (Abrahamsson et al., 2010). Traditional AE approaches, such as Architecture Tradeoff Analysis Method (ATAM) (Kazman et al., 1998), require to prepare extensive documents with long meeting sessions, which is rather heavy and costly for agile development (Christensen et al., 2010; Sharifloo et al., 2008). To address this issue, some studies aimed to develop AE approaches specifically for evaluating architecture in agile development, such as Decision Centric Architecture Reviews (DCAR) [S17], and Modified Architecture Tradeoff Analysis Method [S20]. Details of these approaches are provided in Section 4.3.

Architectural Understanding (AU) lies in the third position of the architecting activities discussed in agile development (8 out of 54 studies, 14.8%). Stakeholders are expected to clearly define the architecture, and should not assume a tacit, implicit understanding of architecture in agile development (Abrahamsson et al., 2010). A shared and consistent understanding on the existing architecture can also help the development team to identify the implementable architectural changes (Jensen et al., 2006). In [S29], the authors used a simple story about eating lunch as a metaphor, to help teammates understand architecture (e.g., how important quality attributes are promoted by architectural design decisions). For example, clean code as part of maintainability (one of the quality attributes) can be explained with the "Bento Box" metaphor, without which the dessert, noodles, and sushi might get all mixed up and won't taste very good.

Architectural Maintenance and Evolution (AME) occupies the fourth position (8 out of 54 studies, 14.8%). Architectures need

Table 8
Studies distribution over industry and academia.

Industry (44, 81.5%)	Academia (11, 20.4%)
[S2][S3][S4][S5][S6][S7][S9][S10][S12][S13][S14][S15][S17][S18][S19][S21][S22][S24][S25][S26] [S29][S30][S31][S32][S33][S34] [S35][S36][S37][S38][S39][S41][S42][S43][S44][S46][S47][S48] [S49][S50][S51][S52][S53][S54]	[S1][S8][S11][S16][S20][S23] [S27][S28][S40][S44][S45]

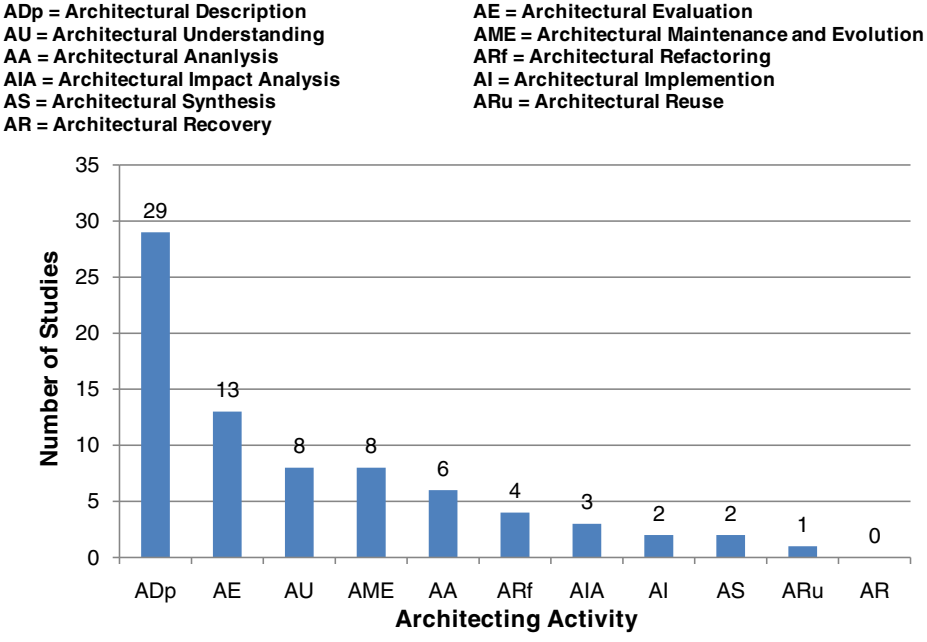

Fig. 6. Studies distribution over architecting activities.

to be maintained and evolved in agile development. For example, maintaining architectural integrity can preserve earlier architectural design decisions during iterations (Diaz et al., 2013). As an example, in [S15], the authors proposed a Change Impact Analysis approach, which allows stakeholders to iteratively and incrementally construct and evolve architectures based on two architecture properties: flexibility and adaptability in agile development. As a result, working architectures can be iteratively and incrementally designed and evolved in agile development through weaving/unweaving extensions of components, and/or by modifying the architecture configuration through optional components and connectors.

Architectural Analysis (AA) comes in at fifth position (6 out of 54 studies, 11.1%). In agile development, customers are actively involved in most of the tasks related to AA, such as examining architectural context and concerns (Babar, 2009). In [S13], the authors proposed Architecturally Savvy Personas approach to explore and analyze architecturally significant requirements (ASRs) in agile development, in which developers and architects work collaboratively to create a small and distinct set of personas and to write a series of architecturally significant user stories for each persona.

Architectural Refactoring (ARf) ranks in the sixth position (4 out of 54 studies, 7.4%). Thapparambil claimed that ARf is an important activity for constructing architecture in agile development (Thapparambil, 2005). This architecting activity allows the development team to optimize architecture design and provides required functionalities in a better way through refactoring past architecture solutions (Sharifloo et al., 2008). In [S2], the architects considered that both architectural and code refactoring could help in agile development to achieve system qualities (e.g., improving maintainability by fixing the structure).

Other architecting activities, Architectural Impact Analysis (AIA), Architectural Implementation (AI), Architectural Synthesis (AS), Architectural Reuse (ARu), Architectural Recovery (AR) have received much less attention in the selected studies. We will discuss the possible reasons in Section 5.

4.3. RQ2: How architecting can be practiced in agile development?

We collected and identified forty-three architecting approaches (AAps) from the selected studies which can be practiced in agile development, as listed in Table 9 with the supported architecting activities by these approaches. Architecting approaches in this SMS refer to various approaches which can facilitate architecting in the architecture-agility combination. These approaches include both approaches directly related to architecture (e.g., architectural evaluation methods like ATAM (Kazman et al., 1998) and DCAR (van Heesch et al., 2014)) and approaches borrowed from other areas that are applied within an architecting activity (e.g., Real Options from corporate finance (Blair et al., 2010; Brown et al., 2010)). Most of these architecting approaches (such as Architecturally-Savvy Personas in [S13] and Simplified Architecture Tradeoff Analysis Method in [S20]) are lightweight, and consequently they can be integrated in agile development with acceptable cost. We describe six of these architecting approaches in this section as examples to show how architecting is practiced in agile development to facilitate the architecture-agility combination.

The forty-three architecting approaches were used in most of the architecting activities (see Section 2.1) as shown in Fig. 7. Most of the architecting approaches were used to support ADp (23 out of 54 studies, 42.6%) and AE (12 out of 54 studies, 22.2%) in agile development. On the contrary, AU (5 out of 54 studies, 9.3%), AA (4 out of 54 studies, 7.4%), ARf (3 out of 54 studies, 5.6%), AME (3 out of 54 studies, 5.6%), AIA (2 out of 54 studies, 3.7%), AI (2 out of 54 studies, 3.7%), AS (1 out of 54 studies, 1.9%), ARu (1 out of 54 studies, 1.9%), and AR (0 out of 54, 0%) have received much less attention in the architecture-agility combination. There is no architecting approach that supports AR in agile development.

"AAp1: Iterative Architecture Approach" makes architecture design embedded into iterations of agile development (Prause and Durdik, 2012). This approach considers only necessary features for the current iteration or delivery of architecture design - it does not try to predict and address future requirements (Chivers et al., 2005). In [S10], the authors argued that the iterative architecture remains true to agile principles. They compared an iterative architecture with a top-down architecture, and found that the iterative architecture defers architecture design costs until features are needed. In [S17], an iterative architecture approach was used with Scrum (called "In-Sprints" architecture approach); there is no dedicated sprint for architecture design in this approach. In each sprint, the development team starts making architectural decisions, designs, and refactors the system part by part upon incoming feature requests.

"AAp2: Reference Architecture" has been frequently discussed in agile development. Bass et al. defined that "*a reference architecture is a reference model mapped onto software elements (that cooperatively implement the functionality defined in the reference model) and the data flows between them*" (Bass et al., 2003). In [S21], the authors proposed an agile architecture approach, which benefits from both the quick feedback in the delivery of short and incremental releases and the long-term vision of the architecture. The approach employs a reference architecture to describe both the function and structure of the major release in agile development. In [S53], the authors found that a predefined reference architecture can reduce the up-front effort needed to develop an architecture, while keeping the benefit of an explicit architecture design.

"AAp3: Change Impact Analysis (IA)" determines the potential effects upon a system resulted from a proposed change (Arnold, 1996). IA can be employed to predict the affected elements of a change before it is implemented, possibly giving an estimate of the effort and cost to implement the change (Ramil and Lehman, 2000), as well as the potential risks involved in making the change (Mens et al., 2008). This analysis can be then used to make better evolution decisions such as whether or not the change should be carried out based on economic viability of software evolution or other risks such as degradation of systems (Diaz et al., 2013). In [S14], IA was used to support both product-line variability and software variability. The output of the IA technique provides architectural knowledge which is useful for reasoning about a proposed change in features and guiding the change decision-making process in agile development. In [S15], the authors proposed a modified IA approach as the main driver for agile architecting. The approach can promote the communication between individuals and agile teams working on the system, and support (semi-)automatically reasoning over the space of architectural knowledge. In agile development, architects can take the advantages of this approach to support the change decision-making process and preserve architecture integrity.

"AAp4: Architecture Dependency Management" is composed of a set of activities, e.g., architecture dependency identification and analysis. In [S8], the authors argued that architectural agility can be achieved by identifying and analyzing architectural dependencies (e.g., between user stories and architectural elements), and integrating architecture dependency management into a responsive development model. [S51] presented an approach with two case studies for identifying implicit architectural dependencies in agile development using revision history of source code change waves. The approach provides architects an effective tool to evaluate the risks when implementing new features in the product and supporting the test planning.

"AAp5: Real Options" is an approach used in corporate finance to assist in investment decisions. Real Options represent the right, but not the obligation to commit to a particular business decision by a particular point in time known as the expiry, after which the right

Table 9
Selected studies over architecting approaches practiced in agile development.

ID	Architecting approach	Supported architecting activities	Studies
AAp1	Iterative Architecture Approach (considering only necessary features for the current iteration or delivery of architecture design)	ADp	[S10][S17] [S25][S44]
AAp2	Reference Architecture	AE, AME, ADp, AU	[S21][S34] [S53]
AAp3	Change Impact Analysis	AIA	[S14][S15]
AAp4	Architecture Dependency Management	AIA, AME	[S8][S51]
AAp5	Real Options (assisting architecture decision making process)	Not stated	[S4][S8]
AAp6	ATAM (Architecture Tradeoff Analysis Method)	AE	[S38][S40]
AAp7	Plastic Partial Component (malleable components that can improve the flexibility of architecture design)	AA, ADp	[S14][S43]
AAp8	Iterative/Incremental Architecture Evaluation	AE	[S31][S41]
AAp9	Connecting Architecture with Code (e.g., embedding architectural information into code)	ADp	[S11][S30]
AAp10	Iteration Zero Approach (introducing architecture design in the first iteration, e.g., in Scrum, it is called Sprint Zero Approach)	Not stated	[S17][S44]
AAp11	Agile Product-Line Architecting (including a Flexible-PLA (Product Line Architecture) modeling Framework)	AME, ADp, AU	[S14][S15]
AAp12	Extended Iteration Zero Approach (introducing architectural design in the initial iterations in agile development). This approach is an extension of AAp10.	ADp	[S44]
AAp13	Responsibility Driven Architecture (exploring the questions of when architectural decisions should be made, by what process, and by whom)	Not stated	[S4]
AAp14	Incremental Commitment Model (facilitating the architecture-agility combination by fitting the system situation)	Not stated	[S5]
AAp15	Architecture Technical Debt Management	Not stated	[S8]
AAp16	Architectural Quality Assurance Technique (a lightweight approach for continuous architectural quality assessment and prioritization)	AE, AU	[S12]
AAp17	Architecturally-Savvy Personas (eliciting ASRs in agile development, e.g., using Architectural Issues Template)	AA, AE, ADp, AU	[S13]
AAp18	Agile Architectural Modeling (adapting architectural modeling in an agile context)	ADp, ARu	[S16]
AAp19	Architectural Knowledge Management (e.g., using Architectural Knowledge Base)	ADp	[S17]
AAp20	Decision-Centric Architecture Review	AE	[S17]
AAp21	Separated Architecture Team (a team dedicated to make architectural decisions)	Not stated	[S17]
AAp22	Architecture as a Service and Architects as Service Providers	ADp	[S18]
AAp23	Simplified Architecture Tradeoff Analysis Method (eliminating several phases and steps in classical ATAM, e.g., preparation)	AE	[S20]
AAp24	One-page Component Contract (describing the capabilities of each component)	ADp	[S21]
AAp25	Abstract Architecture Specification Document (emphasizing minimal architectural documentation in agile development)	ADp	[S22]
AAp26	Lean Architecture (using Data-Context-Interaction paradigm with agile and lean practices)	ADp, AI	[S23]
AAp27	Release Development based on Architecture (considering architecture in release activities, e.g., release planning)	ADp, ARf	[S24]
AAp28	Architectural Design Pattern	ADp	[S25]
AAp29	Developer Story (similar to user story, but written in the developers' perspective)	ADp, AU, ARf	[S27]
AAp30	Architectural Game (explicating and visualizing architectural refactoring needs as stories)	ADp	[S27]
AAp31	Hybrid Architecture Evaluation Method (combining Active Review for Intermediate Designs, Quality Attribute Workshop, and ATAM)	AE	[S28]
AAp32	Lightweight Component Framework (generating glue code for each component to facilitate architectural implementation)	AI, ADp	[S30]
AAp33	Informed Technology-Insertion Decision Making (architecture information is periodically provided to stakeholders, which helps them to make and document architectural decisions)	AE, ADp	[S31]
AAp34	Enterprise-Service-Bus-Based Architecture	ADp	[S34]
AAp35	Mind Map (facilitating architectural description by connecting business and domain models)	ADp	[S34]
AAp36	Domain-Driven Architecture Design Technique (communicating the intent of code through domain-driven design and assessing architecture without necessitating large amounts of documentation)	Not stated	[S36]
AAp37	Proof-of-Concept (used for validating architecture)	Not stated	[S36]
AAp38	Active Reviews for Intermediate Designs (a combination of stakeholder-centric and scenario-based architecture evaluation methods, such as ATAM and active design review of design specifications)	AE	[S38]
AAp39	Architecture-related Tasks Visualization (improving development flow management in Lean Software Development)	AA	[S39]
AAp40	Attribute-Driven Design	AA, AS, ADp	[S40]
AAp41	CBAM (Cost-Benefit Architectural Analysis Method)	AE	[S40]
AAp42	Continuous Architectural Refactoring	ARf	[S50]
AAp43	Real Architecture Qualification (a brainstorming session to evaluate the system qualities of a working system that leads to a new iteration)	AE, ADp	[S50]

is no longer available (Blair et al., 2010; Brown et al., 2010). In [S4], Real Options approach provides agile teams with a way to frame their up-front work (e.g., architecture design), and allows users to re-frame the decision making process. By drawing on Real Options theory, the authors developed a process that allows development teams to collaborate on the decision making process, and in doing so to realize the benefits (e.g., actively exploring new options which could not have been foreseen up-front) of architectural design. In [S8], Real Options analysis provides models (e.g., option-based model) to help stakeholders make informed choices which balance between various aspects, such as speed of development and plan of future needs. Real Options analysis also offers a way in agile release planning to allocate architectural elements to releases in a value perspective.

"AAp6: ATAM" focused on identifying and surfacing architectural risks (Kazman et al., 1998). The goal of ATAM is to evaluate the design outcome of architectural decisions against system quality requirements, and help stakeholders to discover potentially problematic architectural decisions (Nord and Tomayko, 2006). In [S38], the

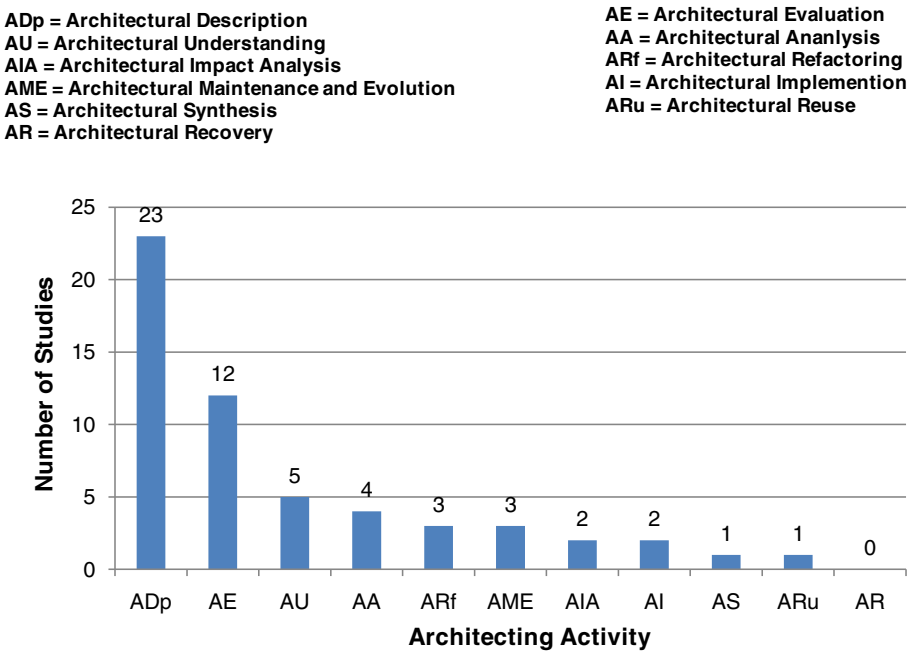

Fig. 7. Studies distribution over architecting approaches and architecting activities.

authors suggested that the development team can use ATAM to analyze the architecture with the quality scenarios in current iteration in agile development. In [S40], the authors emphasized that ATAM can provide guidance for analyzing architectural design and help stakeholders get early feedback on the risks caused by problematic architectural decisions in agile development. For example, early in the development, XP practitioners work on stories augmented by scenarios and the architectures they developed following the Attribute-Driven Design (ADD) approach. They can do ATAM right after this to track technical and business risks early in the process and to help prioritize stories for the next release.

4.4. RQ3: Which agile development methods can be used with architecture?

We collected ten agile methods which can be used in the architecture-agility combination from the selected studies, as shown in Fig. 8 with their numbers of studies. Some studies did not explicitly mention the agile methods used; those are classified as "Not Stated" in Fig. 8. Note that one study may introduce several agile methods used with architecture, so the number of the agile methods used in the selected studies (63) is greater than 54 (the number of the selected studies).

Scrum (22 out of 63 agile methods, 34.9%) is the most popular agile method used in the architecture-agility combination. This is consistent with the results from several recent surveys (Ali, 2012; Rodríguez et al., 2012; Stavru, 2014) that Scrum is the most popular method in agile development. Though XP has been dominant in agile development in the past (Cao et al., 2004; Dybå and Dingsøyr, 2008; Hamed and Abushama, 2013), we found that the focus of agile methods has shifted from almost exclusively XP to several other agile methods (e.g., Scrum) during recent years as well as in the architecture-agility combination.

XP (9 out of 63 agile methods, 14.3%) still reaches the second place in agile methods used in the architecture-agility combination. The rest agile methods such as LSD (4 out of 63 agile methods, 6.3%), and Crystal (1 out of 63 agile methods, 1.6%) received much less attention (less than 5 studies for each method) in the architecture-agility combination.

4.5. RQ4: Which practices of agile development can be used with architecture?

We collected and identified forty-one agile practices (APs) which can be used in the architecture-agility combination from the selected studies. Note that we only extract the data (i.e., data item D14: Agile practices) if a study explicitly mentioned the agile practices and how they were used in the architecture-agility combination. Table 10 presents a list of agile practices with the studies that use these practices. In this section we describe the six most frequently discussed agile practices used with architecture from the selected studies.

"AP1: Backlog" (22 out of 54 studies, 40.7%) is the most frequently discussed agile practice in the architecture-agility combination. This is because many selected studies in this SMS used Scrum as the agile development method in their work, and "Backlog" is a basis element of Scrum (Schwaber, 1995; Schwaber and Beedle, 2002). We further extracted several types of "Backlog", such as product backlog (e.g., [S5]), sprint backlog (e.g., [S24]), requirements backlog [S18], and architecture backlog [S37]. In [S17], the product backlog was used with Architectural Knowledge Base (AKB) to create an architectural document, which explicitly presents the architectural design to the stakeholders. AKB generates an architectural evaluation report ("Sprint architecture evaluation report") for each sprint, which can be reviewed during the sprint review meeting. The results of the review may have an impact on both the product backlog and sprint backlog. In [S46], the architects identified key areas in architecture design, which were described in UML diagrams and added to sprint backlog. This ensured key architectural documentation was completed, its cost was visible and that cost was not absorbed into the development activity.

"AP2: Iterative and Incremental Development" (22 out of 54 studies, 40.7%) also lies in the first position of the agile practices used with architecture. In [S13], the authors followed the ADD (attribute-driven design) approach to build an incremental scenario-driven approach (Architecturally-Savvy Personas) for architecture design, which first identifies quality attribute scenarios, and then proposes and evaluates candidate architectural solutions in agile development (e.g., in sprints). In [S40], the authors revealed that an architecture created and documented using the ADD approach provide developers greater

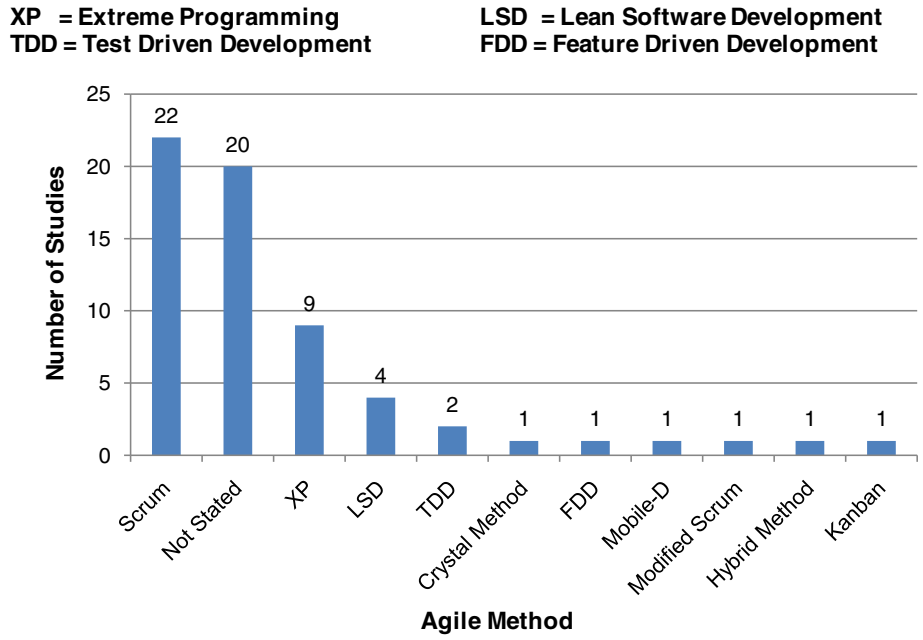

Fig. 8. Studies distribution over agile methods.

flexibility and opportunity to defer making more detailed decisions until the proper time. For example, the module view fits XP's iterative development quite well.

"AP3: Just Enough Work" (13 out of 54 studies, 24.1%) is in the third position. In [S8], the authors pointed out that "just enough anticipation" is required to achieve architectural agility, and architectural anticipation should be done in an "informed" way to not over-anticipate or under-anticipate architecture-related issues (e.g., future needs). In [S53], the authors proposed that "Just enough up-front design" is a solution to facilitate the architecture-agility combination, because architects only need to make necessary architectural design decisions up-front in agile development. Furthermore, "Just enough up-front design" can be practiced in various ways. For example, the "iteration zero" phase can include some up-front architectural design, with continuous architecture refinement in the subsequent iterations.

"AP4: Sprint" (12 out of 54 studies, 22.2%) used in Scrum comes in the fourth position. In [S12], the results of the study indicate that architectural problems or tasks can be included in the sprint backlog and processed in future sprints, through the use of the architectural Software Quality Assurance technique at the end of each sprint. In [S15], a Change Impact Analysis (IA) technique was applied to the working architecture of the previous sprint (except for the first sprint). It provides agile architects with the change-impact knowledge resulting from the changes planned for the sprint. In [S41], the authors argued that embedding incremental architecture evaluations into sprints can mitigate the risk of combining agile development and large-scale architecture-centric development.

"AP5: Agile Architecting (11 out of 54 studies, 20.4%) is in the fifth position. In [S15], the authors pointed out that agile architecting should be a lightweight architectural decision-making process in agile development, which can be achieved by providing knowledge to assist architects in reasoning about changes (feature increment or evolution). In [S16], agile architecting (through an agile architecture-driven modeling process) was used to combine agile methods and architectural modeling. Only a little up-front architectural planning is required and only the artifacts which directly contribute to the current iteration are created in architecting. The author of [S16] argued that the agile architecture-driven modeling process acts as a core element in the architecture-agility combination.

"AP6: Continuous Integration" (10 out of 54 studies, 18.5%) ranks in the sixth position. In [S24], the authors stated that continuous integration can be used with architecture through integration testing in each integration, which can validate the architecture iteratively. In [S45], the authors mentioned that new tests can be added to verify the new requirements if a change in requirements has an impact on the architecture. Continuous integration may facilitate this by informing the related stakeholders or teams when changes are made.

In addition to these six frequently-discussed agile practices, there are some other agile practices that can be applied in the architecture-agility combination, such as "Scrum Meeting" (9 out of 54 studies, 16.7%), "Incremental Delivery" (9 out of 54 studies, 16.7%), and "User Story" (8 out of 54 studies, 14.8%). As shown in Table 10, the top two agile practices "Backlog" and "Iterative and Incremental Development" have been much more frequently discussed than the others in the architecture-agility combination.

4.6. RQ5: What are the costs and benefits of the architecture-agility combination?

We found that most studies tend to discuss the benefits of the architecture-agility combination instead of the costs. The detailed answers are provided in Section 4.6.1 (costs) and Section 4.6.2 (benefits).

4.6.1. Costs

The cost refers to the effort (e.g., time, money, and labor) of using architecture in agile development or employing agile methods and practices in architecture-centric development. All the costs discussed in the selected studies are related to architecture (as shown in Table 11). It shows that when combining architecture and agile development, the effort of architecting activities as well as the use of associated architecting approaches (see the answer of RQ2 in Section 4.3) are the major reasons that lead to the costs. For example, some proponents of agile development perceive architecture as a bad investment, which may lead to BDUF, massive documentation, and redundancy (Abrahamsson et al., 2010). This means that combining architecture and agile development would cause some extra effort and resources consumption (e.g., time) of the development team.

Table 10
Distribution of selected studies over agile practices.

ID	Agile practice	Studies	No.
AP1	Backlog	[S1][S2][S3][S5][S12][S13][S14][S15][S17][S18][S21][S23][S24] [S32][S37][S39][S41][S43][S45][S46][S49][S52]	22
AP2	Iterative and Incremental Development	[S1][S2][S3][S4][S8][S10][S13][S14][S15][S16][S17][S20][S24][S34] [S38][S39][S40][S43][S48][S49][S50][S53]	22
AP3	Just Enough Work	[S1][S2][S8][S17][S21][S22][S25][S32] [S43][S47][S48][S53][S54]	13
AP4	Sprint	[S5][S12][S13][S14][S15][S16][S17][S24] [S37][S39][S41][S43]	12
AP5	Agile Architecting	[S6][S14][S15][S16][S17][S20][S25][S28] [S41][S43][S45]	11
AP6	Continuous Integration	[S3][S14][S24][S26][S35][S45][S48][S49] [S50][S54]	10
AP7	Scrum Meeting / Daily Stand-up	[S13][S14][S15][S20][S24][S30][S32][S41][S43]	9
AP8	Incremental Delivery	[S1][S8][S10][S21][S30][S32][S39] [S43][S54]	9
AP9	User Story	[S1][S2][S13][S14][S39][S40][S45][S49]	8
AP10	Rapid and Flexible Development	[S14][S22][S28][S34][S40][S45][S48]	7
AP11	Refactoring	[S8][S10][S14][S27][S43][S50]	6
AP12	Retrospective	[S3][S14][S17][S20][S25][S46]	6
AP13	Face-to-Face Communication	[S2][S7][S40][S43]	4
AP14	Collaborative Software Development Process (co-located working)	[S7][S22][S31][S50]	4
AP15	Test-Driven Development	[S3][S30][S36][S45]	4
AP16	Quick and Early Feedback	[S21][S30][S38][S41]	4
AP17	Use Case	[S21][S30][S46][S52]	4
AP18	Metaphor	[S1][S29][S50]	3
AP19	Release Planning	[S3][S8][S24]	3
AP20	Effective Communication	[S7][S20][S30]	3
AP21	On-Site Customer	[S16][S27][S40]	3
AP22	Pair Programming	[S10][S30][S50]	3
AP23	Trust in Teams	[S25][S37]	2
AP24	Prototyping	[S3][S20]	2
AP25	Self-Organized Team (the architectural modeling process can be executed by a self-organized team)	[S16][S51]	2
AP26	Developer Story	[S27]	1
AP27	Informal Communication	[S2]	1
AP28	Roadmap / Vision (reducing the risk of being blind-sided by unanticipated external changes)	[S3]	1
AP29	Periodic Design Review	[S3]	1
AP30	Small Team (20–25)	[S5]	1
AP31	Simple Architecture	[S10]	1
AP32	Goal-Oriented Method (every increment adds value to the final product)	[S16]	1
AP33	Incremental Re-architecting	[S20]	1
AP34	End-user-focused and Value-centric System Design	[S23]	1
AP35	Poka-Yoke (a mechanism in a lean manufacturing process to reduce defects through careful up-front deliberation)	[S23]	1
AP36	Small Increments and Short Iteration	[S38]	1
AP37	Feature-based and High-Priority Task	[S39]	1
AP38	Eliminating Waste (i.e., overproduction, delay, and defect waste)	[S39]	1
AP39	Just-In-Time Delivery	[S39]	1
AP40	Spike Solution (a simple program to explore potential solutions)	[S50]	1
AP41	Tacit Knowledge Increasing Agility (agility in agile development is highly related to the tacit knowledge of the team members)	[S53]	1

Table 11
Classification of costs of the architecture-agility combination.

ID	Cost	Description	Studies
C1	Architectural design cost	Architectural design can be costly in agile development due to several reasons (e.g., no time for careful design during iterations, inappropriate use of metaphors in architecture design results in additional time and effort in e.g., communicating architecture).	[S2][S10] [S25][S39] [S44][S50]
C2	Architecture documentation cost	The development team needs to spend extra effort for architecture documentation in agile development.	[S48]
C3	Architectural knowledge learning cost	Learning architectural knowledge would cost time and effort, and slow down the development process.	[S14]
C4	Quality fixing cost caused by unsatisfied architecture design	The architecture design that cannot meet quality requirements in agile development, may lead to a cost overrun for fixing the quality requirements during software maintenance.	[S2]
C5	Architecture delay cost	Large increments in lean development may lead to the delay cost due to waiting for the architecture increments to be ready.	[S39]
C6	Architecture-caused rework cost	Architecting in many smaller increments may add to the cost of rework because it might involve rearchitecting.	[S39]
C7	Architectural maintenance cost	Maintaining architecture within agile development methods can be costly because architecture adds extra formalism and documentation.	[S48]

Table 12
Classification of general benefits of the architecture-agility combination.

ID	Benefit type	Subtype	Description	Studies
GB1	Architecture	Architectural Evolution	The architecture-agility combination can facilitate a continuous evolution of architecture.	[S8][S31]
		Architectural design	The architecture-agility combination can set architectural design right at the early phase of development, and help to construct systems with good architecture.	[S23][S37]
		Architectural quality	Incremental architecture can provide flexibility of architecture, and agile architecting can assist architects to maintain integrity of architecture.	[S10][S15]
		Architecture decision-making process	Agile architecting can assist and guide architects in the decision-making process of implementing changes in each agile iteration.	[S15]
		Architectural Implementation	The architecture-agility combination follows a "just-in-time" model to develop architecture, and delivers features without having to complete the whole architecture design in advance.	[S8]
		Architectural Analysis	The architecture-agility combination can save effort in Architectural Analysis by considering only essential features needed for the current iteration.	[S10]
		Architectural documentation	Agile development can save resources on architectural documentation activity.	[S2]
		Architectural Refactoring	Architectural refactoring provides a better means to help architects achieve system qualities incrementally than an up-front design.	[S2]
		Architectural knowledge sharing	Agile development can easily and quickly share architectural design decisions by using e.g., Wiki.	[S2]
GB2	People	Stakeholder	Evaluation of architecture in agile development can provide the stakeholders with confidence about the architecture.	[S38]
		Developer	Agile development methods can deliver early information to developers and help them make design decisions.	[S2]
GB3	Reduced Cost	Up-front design cost	Incremental architecture can minimize up-front architecture design cost.	[S10]
		Architecture delay cost	Architecting in small increments reduces the delay cost, because the development team does not need to wait for the whole architecture to be completed.	[S39]
		Rearchitecting cost	Architecting in large increments reduces the cost of rearchitecting in agile development, because rework costs are incurred due to architecture-related defects and overproduction waste from increments.	[S39]
GB4	Project & System	Project status	The architecture-agility combination provides improved visibility of project status.	[S41]
		Feedback to system	Incremental architecture design leads to early feedbacks to the system.	[S38]
GB5	Others	System quality	The architecture-agility combination can improve and address quality attributes of systems in a systematic way by e.g., focusing early on architectural decisions.	[S40][S50]
		Communication	Physically decentralizing architects and having them incorporate architecture concerns can facilitate the communication between agile teams and product owners.	[S37]
		Risk mitigation	The architecture-agility combination can mitigate risks and improve risk management through reviewing architecture risk factors or conducting incremental architecture evaluations.	[S41]

4.6.2. Benefits

We collected and classified the benefits of the architecture-agility combination into two categories: general benefits (GBs) and specific benefits (SBs). General benefits refer to the benefits, which are not restricted to specific architecting approaches (see Section 4.3) and agile methods (see Section 4.4) as well as supporting tools employed (see Section 4.9), while specific benefits are the effect when employing specific approaches, methods, and tools. A detailed classification of the architecture-agility combination benefits is presented in Table 12 (general benefits) and Table 13 (specific benefits) respectively, which are derived by using Grounded Theory from the extracted raw data as detailed in data synthesis (in Section 3.2.4). Note that one study may mention or discuss several benefits of the architecture-agility combination.

4.6.2.1. General benefits. We classified general benefits into five benefit types with nineteen subtypes in Table 12.

"GB1: Architecture" is the type of benefits of the architecture-agility combination that is most frequently discussed (by twelve studies). The architecture-agility combination can benefit various aspects of architecture. In [S31], the architecture-agility combination can facilitate a continuous evolution of architecture that supports evolving system capabilities. In [S37], agile development allows architects to work closely with both the business and technical teams, which helps to construct systems with good architecture.

"GB2: People" refers to the roles in software development who may get benefits through the architecture-agility combination. Note that some benefits only apply to specific roles (e.g., developer) and some other benefits hold for a whole group (e.g., development team). In [S2], the author found that agile development can bring the development team early in the picture of architecture. In [S38], the authors argued that a comprehensive evaluation of the architecture at the end of an iteration can make the stakeholders confident that their concerns are well-addressed by the architecture in agile development.

"GB3: Reduced Development Cost" denotes that the architecture-agility combination can reduce certain cost in development (e.g., up-front design cost). For example, in [S39], one of the benefits of architecting in small increments in lean development is the reduced delay cost without the necessity of waiting for a full-fledged architecture to be ready. In [S10], the authors argued that an incremental architecture was clearly effective, which can minimize up-front design cost.

"GB4: Project & System" refers to the benefits of the architecture-agility combination to project development or the system itself. In [S41], the authors argued that combining the agile principles and architecture may improve the visibility of project status. In [S38], the authors revealed that architecture work helped to build a "skeleton system" that supports incremental addition of features, and led to releases for early feedback from stakeholders.

"GB5: Others" contains those benefits which cannot be classified into any type above. In [S40], the authors stated that including architecture-centric design and analysis methods in XP helped address quality attributes in an explicit, methodical, and engineering-principled way.

4.6.2.2. Specific benefits. We collected and classified specific benefits into four benefit types with twenty-one subtypes in Table 13. "SB1: Architecture" is still the type of benefits of the combination that is most frequently discussed when using specific approaches, methods, and tools.

"SB1: Architecture". In [S14], Flexible-PLA Modeling Primitives are effective in providing architectures with flexibility and adaptability

Table 13
Classification of specific benefits of the architecture-agility combination.

ID	Benefit type	Subtype	Description	Studies
SB1	Architecture	Architectural design	The various approaches facilitate architectural design in several ways in agile development, e.g., providing architectural guidance, making it easier to make architectural changes.	[S13][S17][S18] [S21][S23][S27] [S29][S34][S38] [S43][S50][S53]
		Architectural Description	The various approaches help to make architectural description explicit, efficient, and simple with just enough information, which fits the agile context.	[S11][S13][S17] [S18][S22][S23] [S25][S29][S30] [S38][S45]
		Architecture quality	The various approaches help to assess, quantify, prioritize, understand, and improve architecture qualities in agile development (e.g., integrity, consistency, flexibility, and adaptability).	[S12][S14][S15] [S23][S25][S27] [S34][S36][S37]
		Architecture decision-making process	The various approaches provide agile architects knowledge and help them in the decision-making process, e.g., deferring detailed decision making until the proper time.	[S4][S14][S15] [S27][S29][S30] [S40]
		Architectural Evaluation	The various approaches make Architectural Evaluation suitable in agile development with early feedback, e.g., lightweight and incremental methods.	[S4][S11][S17] [S20][S38][S41]
		Architectural knowledge management	The various approaches can improve architectural knowledge management in agile development through e.g., addressing the knowledge evaporation problem.	[S17][S21][S22] [S27]
		Architectural Analysis	The various approaches are effective for analyzing architectural requirements (e.g., through piecewise analysis), aligned with agile practices.	[S13][S17][S27] [S40]
		Architectural Understanding	The various approaches help development teams to better understand the architecture (e.g., how the design satisfies specific quality attributes) in agile development.	[S14][S29][S37]
		Architecture communication	The various approaches can improve team communication about architecture in agile development.	[S18][S29][S30]
		Architectural Refactoring	The various approaches use design knowledge (e.g., architectural patterns) for refactoring the architecture or increase the visibility of architectural refactorings (e.g., using developer stories) in agile development.	[S25][S27]
		Architectural Implementation	CA[a] Agile Architecture helps the development teams to effectively implement an architecture without drifting towards continuously changing requirements.	[S21]
		Architectural effort optimization	Technical debt management and real options analysis provide the ability of optimizing architectural investment decisions in agile development.	[S8]
		Architectural Reuse	Agile Architecture-Driven Modeling approach supports reuse of components, patterns, and artifacts in agile development.	[S16]
		Architectural Recovery	A lightweight XML-based architectural description approach was used to effectively capture and communicate architecture without a heavy modeling frontend.	[S30]
SB2	People	Development team	The various approaches can bridge agile teams with a non-agile enterprise, reduce time, effort, and improve confidence of architecture in agile development.	[S4][S42][S53]
		Product owner	Architecture backlog provides a measurable mechanism that motivates the product owner to move stories from the architecture backlog to the project backlog, and handle the backlog in a consistent way.	[S37]
SB3	Agile development	Agile principle	The various approaches can support certain agile principles, e.g., the development team can get closer to customer needs during iterations through the use of PPCs. This facilitates the agile principle "*The highest priority is to satisfy the customer through early and continuous delivery of valuable software*" (16).	[S14][S25][S43]
		Agile practice	The various approaches can support certain agile practices, e.g., architectural patterns can help and encourage the project team to document knowledge about architecture in a brief and salient form, which improves agile practice "Just enough work".	[S25][S40][S43]
SB4	Others	System quality	The various approaches can assure or improve system qualities (e.g., simplicity, consistency, and testability) in agile development.	[S13][S25][S31] [S34][S40][S50]
		Risk	System architectural description can reduce risks in agile development process.	[S14][S42]
		Release planning	Real options analysis provides a way to allocate architectural elements to releases in a value perspective, which facilitates agile release planning.	[S8]

[a] CA represents the name of the company CA Technologies, Inc. who developed and applied this method.

while architecting in agile development. In [S29], the authors argued that using metaphors to create a representation of the architecture in self-documenting code were helpful. In [S17], the approach Incorporating Systematic Architectural Knowledge Management can alleviate the pain of architecture work (e.g., the vaporization of architectural knowledge) in Scrum. In [S30], a lightweight XML-based architectural description approach was used to effectively capture and communicate architecture without any heavy modeling frontend, which is suitable for agile development.

"SB2: People". In [S4], Responsibility Driven Architecture is useful when there is a need to bridge agile development teams with a decidedly non-agile enterprise and relatively strict, waterfall-inspired governance processes. In [S37], architecture backlog provides a measurable mechanism that motivates the product owner to move stories from the architecture backlog to the product backlog, and handle the backlog in a consistent way.

"SB3: Agile Development". In [S43], the proposed approach PPC (Plastic Partial Component) can be employed to support six agile principles. For example, PPCs are ready to be modified during development, and can easily accommodate changes of features. Therefore, PPC supports the agile principle "Welcome changing requirements".

"SB4: Others". In [S31], time-phased iterative and incremental development can enhance system testability through incremental architecture verification and just-in-time architectural decision making.

4.7. RQ6: What are the challenges in the architecture-agility combination?

We collected and classified the challenges in the architecture-agility combination into two challenge types (Architecture and Others) with twenty subtypes. Table 14 presents a detailed classification and description of the identified challenges.

Table 14
Classification of challenges in the architecture-agility combination.

ID	Challenge type	Subtype	Description	Studies
CH1	Architecture	Tension: Architecture design and agile development	Architecture design represents a plan for the system development, while agile development embraces change, and pays less attention to plans. How much architecture design is required is a challenge in the architecture-agility combination.	[S9][S16][S17] [S27][S30][S47] [S52]
		Tension: Architectural description and agile development	Deciding how much effort for architectural description is needed in agile development is a challenge.	[S11][S16][S22] [S41][S45]
		Role of architecture in agile development	The role of architecture in agile development is not clear, and often overlooked.	[S1][S4][S8] [S15]
		Tension: Architectural evaluation and agile development	There is a lack of proper architectural evaluation approaches suitable for agile development.	[S12][S20][S28] [S41]
		Tension: Architectural knowledge management and agile development	Agile teams often depend on tacit architectural knowledge, and how to deal with architectural knowledge vaporization in agile development is a challenge.	[S11][S14][S15] [S21]
		Tension: Architectural quality and agile development	There is a need to balance the effort for architectural quality (e.g., flexibility) and agile development. For example, an excessively flexible architecture may result in redundant work.	[S3][S14][S39]
		Tension: Architectural refactoring and agile development	In agile development, architectural refactoring is often costly and may need significant rework.	[S8][S15][S39]
		Architect	The role of architects in agile development is not clear, and the agile architects have far fewer standard practices to follow.	[S1][S4][S47]
		Tension: Architectural work and agile teams	As agile development teams became self-organized, the architectural work became more distributed and harder to control centrally.	[S51]
		Tension: Architectural analysis and agile development	Existing techniques for early discovery of ASRs are typically rejected by agile development teams as being somewhat heavy-weight.	[S13]
		Tension: architectural evolution and agile development	Evolving architecture while complying with the agile principles is a challenge.	[S14]
		Tension: architectural patterns and agile development	In agile development, there is a lack of guidance on how to use architectural patterns.	[S25]
		Tension: architectural delivery and agile development	Delivering the architecture in smaller increments and shorter iterations is a challenge.	[S38]
		Understanding of architecture work's value	The architecture work will get low priority in the product backlog if the product owner is not aware of the value of architecture, and consequently may lead to an inferior architecture.	[S37]
CH2	Others	Champion of the architecture-agility combination	There is a lack of a champion of the architecture-agility combination for many reasons (e.g., the time, motivation and effort, the merging of agile elements and architectural elements, the different cultures).	[S1][S2][S4] [S13][S14][S21] [S24] [S25][S39] [S40][S42][S43] [S44] [S45][S53] [S54]
		Large project	An inaccurate architectural design in agile development is one of the major reasons that lead to the failure of large projects. Geographically dispersed teams working on large projects may have difficulties in using agile methods (e.g., metaphors) for understanding, communicating, and documenting architecture.	[S5][S24][S29] [S42][S43]
		Techniques of the architecture-agility combination	There is a lack of techniques (e.g., approaches, tools) to support the architecture-agility combination.	[S12][S13][S20] [S28]
		Evidence of the architecture-agility combination	There is a lack of evidence to support that the architecture-agility combination works.	[S2][S24][S54]
		User story prioritization	User stories in agile development are usually prioritized without considering the technical aspects (e.g., interdependencies between them), which potentially has a negative effect on the architectural decisions made to implement the user stories.	[S2]
		System quality	There is a lack of attention on system qualities in the architecture-agility combination.	[S2]

"CH1: Architecture" refers to the challenges about architecture itself. In [S1], one of the challenges is that agile methods perceive architecture as planning too much in advance. In [S2], the authors emphasized that agile methods underestimate architecture design as an important activity. In [S11], the authors recognized that the chances are high that architecture only exists in the quickly fading memory of developers and forgotten architectural spikes, given that the agile methods highly focus on creating code over documentation.

"CH1: Others" covers the challenges which are indirectly involved with architecture. In [S24], the author found through interviewing developers in a large company that agile methods do not generally guide architecting large-scale systems. Agile development in large projects and the co-existence of agile development and architecture in different problem classes are among the most burning issues from practitioners. In [S20], the authors argued that one of the challenges in agile development is the lack of proper architecture evaluation approaches suitable for agile development process. In [S4], the authors argued that the role of architects as well as their responsibilities is not clearly defined or simply ignored in agile development.

4.8. RQ7: What are the factors that impact the success of applying the architecture-agility combination?

We collected and classified six types of factors with twenty-nine subtypes which may impact the architecture-agility combination from the selected studies. Note that if we could not find any detailed information about a factor (e.g., how it can impact the architecture-agility combination), we excluded it without any further discussion. Table 15 presents the classification and description of these factors with the studies and Fig. 9 shows the top ten factors that may impact the architecture-agility combination.

"F1: Project" covers the factors of the project level (e.g., project size). "F2: People" refers to the stakeholders (e.g., architects) with their properties (e.g., experience) which can impact the architecture-agility combination. "F3: Architecture" includes the

Table 15
Classification of factors of the architecture-agility combination.

ID	Factor type	Subtype	Description	Studies
F1	Project	Project size	Different project size can impact the architecture-agility combination (e.g., the amount of architectural rework is an exponential function of project size).	[S1][S5][S15] [S18][S19][S33] [S53][S54]
		Time	There is not much time for designing architecture in agile development. Stakeholders should consider when to do architecture work and how much time can be spent on it (e.g., the time of architectural evaluations may be restricted to several hours per week).	[S21][S25][S38] [S39][S41][S50] [S51]
		Governance	It refers to the management of agile projects, which includes the standard of architectural work (e.g., which standards should be followed in architecture design).	[S1][S26][S33] [S44][S54]
		Criticality	Criticality (e.g., lives may be lost if the system fails) can be a factor that impacts the architecture in agile development. For example, a project with critical ASRs hardly progresses in an expectation that a satisfactory architecture will emerge through refactoring.	[S1][S5][S33] [S53][S54]
		Rate of change	Software change will impact the adoption of the architecture-agility combination. For example, in agile development, if the rate of requirements change is high, it is quite unlikely that a satisfactory architecture will emerge through small continuous refactoring.	[S1][S5][S33] [S44][S54]
		Development cost	This factor covers various costs in the software development lifecycle, e.g., understanding how the architectural communication costs will be paid during the releases in agile development is a key to success.	[S7][S8][S50]
		Goals confliction	Different stakeholders may have different goals (e.g., business goals, project goals). Goals confliction is a reason for poor documentation and neglected architecture design in agile development.	[S44]
F2	People	Team	This factor covers team distribution, team dimension of scale, and team experience. For example, team distribution calls for more explicit communication and coordination of architectural decisions, and more stable technical and business interfaces between teams.	[S1][S7][S16] [S18][S19][S20] [S33][S38][S41] [S54]
		Architect	This factor covers architect's role, responsibilities, background, and experience. For example, when developing a large and novel system, an appropriate combination of experience from the maker of important decisions (e.g., architectural design decisions), as well as the mentor, prototyper, and troubleshooter of the system is needed.	[S1][S2][S4] [S15][S18][S37] [S46][S50][S53] [S54]
		Customer	This factor contains customer requirements, participation, and collaboration. For example, customers' participation may help them to choose what user stories or developer stories to include or exclude in each iteration in a well-informed way, which consequently improves the decision-making on architecture.	[S23][S27][S33] [S41][S50][S54]
		Developer	This factor covers developer's attitude, behavior, etc. For example, developers may not care about architecture, have very few experience on architecture design, etc. This would cause problems in the architecture-agility combination.	[S27][S44][S53]
		Domain expert	In agile development, domain expert engagement is valuable for designing architecture.	[S23]
F3	Architecture	Architecture documentation	This factor includes several aspects of architecture documentation, such as how much effort should be assigned, awareness of the importance, and the length and complexity of architecture documentation, etc.	[S1][S2][S7][S11][S10] [S20][S21][S22][S23] [S38][S44][S50]
		Architectural quality	In agile development, architecture should have certain qualities (e.g., simplicity and stability). For example, XP uses "simplicity" as the main guiding principle for architectural design.	[S1][S10][S12] [S15][S33][S50] [S53][S54]
		Architecture scope	Clearly defining the architecture scope is important in agile development. A tacit and implicit scope would cause some problems, such as architecture misunderstanding and erosion.	[S1][S38][S41]
		Architecting value and cost	The agile team should consider both the value and cost of architecting to decide the amount of architectural effort they should spend.	[S1][S54]
		Architectural defects	In agile development, architectural defects (such as lack of desirable system qualities) may lead to a large amount of rework.	[S39]
		Architectural increment	The batch size of architectural work in increments can impact the architecture-agility combination.	[S38]
F4	People-related	Communication	Agile methods highlight an incremental development paradigm, in which architecture design is under a communicative development process. Communication (e.g., communicating architecture) can impact the whole agile development process.	[S2][S7][S17][S18][S20] [S23] [S25][S26][S27] [S29][S31][S37] [S42][S46][S50]
		Coordination/ collaboration	One key aspect of agile development is the coordination and collaboration within the agile team (e.g., between architects and product owners).	[S7][S37][S48]
		Learning	Learning new architectural concepts could slow down the agile development process.	[S14]
		Support from stakeholders	Lack of support from surrounding stakeholders is a contextual factor in the architecture-agility combination.	[S33]
		Expertise	Expertise can be used to combine architecture and agile practice to avoid disruptions in velocity, e.g., release planning with architectural considerations.	[S3]
F5	Organization	Business	This factor covers business domain and business model, etc. For example, web-based systems, embedded systems, and mobile APPs can be very different when applying the architecture-agility combination in their development.	[S1][S33][S41] [S47][S53][S54]
		Organization culture	The architecture-agility combination would be very difficult to perform if an organization is not experienced with agile development.	[S1][S17][S33] [S41][S47]
		Technical environment	Having a development infrastructure (e.g., automated tests, nightly builds) to support the development teams, is an important factor for the combination (e.g., meeting architectural requirements in agile development).	[S33][S41][S47] [S54]
		Organization maturity	This factor covers the development history of the organization, the maturity of the development processes employed, and the scale of the organization, etc.	[S33]
F6	System	System quality	This factor contains system complexity, platform portability, etc. A safety-critical system with high security and real-time constraints may impact the way that architecture and agility are combined.	[S3][S13][S33] [S40][S41][S50] [S54]
		Legacy system	Legacy systems can influence an architectural effort's impact in agile development (e.g., intractable legacy systems can destroy an architecture program).	[S47]

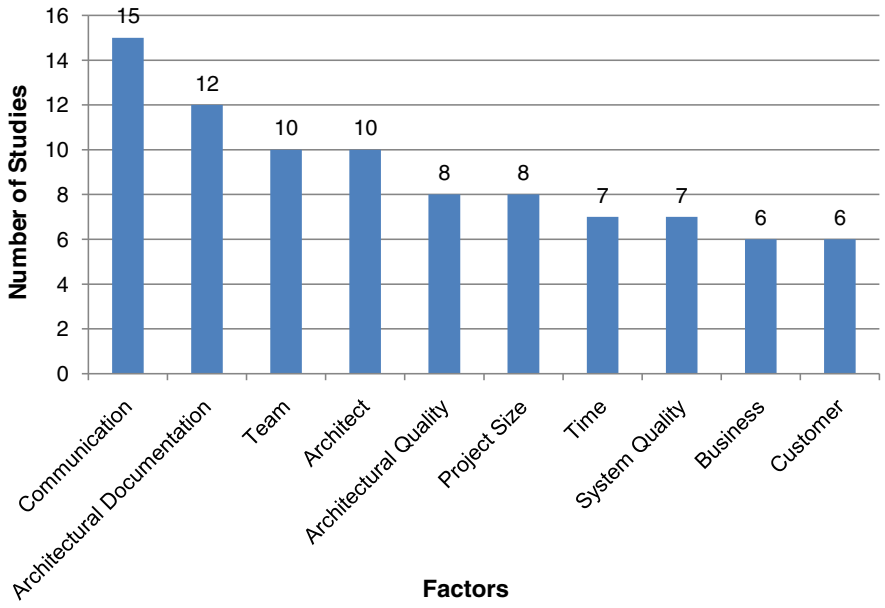

Fig. 9. Studies distribution over the subtype factors that impact the architecture-agility combination (top 10).

factors about architecture. "F4: People-related" denotes the factors that are related to people (e.g., "Communication" is an activity between stakeholders). Compared with the factors of "F1: Project" and "F6: System", the factors of "F5: Organization" are considered at an organization level (e.g., the culture characteristics of a company). "F6: System" includes the system properties (e.g., the size of a system). "F1: Project" (34 out of 136 studies, 25.0%) and "F2: People" (30 out of 136 studies, 22.1%) are the two most important factors that impact the architecture-agility combination. "F3: Architecture" lies in the third position (27 out of 136 studies, 19.9%). The rest factors ("F4: Project-related", "F5: Organization", and "F6: System") have a moderate impact on the combination. Note that the number of studies that discuss various factors (136) exceeds the number of selected studies (54), because one study may discuss several factors.

4.9. RQ8: What tools are available to support the architecture-agility combination?

We collected seven tools that can be used to support the architecture-agility combination in the selected studies, as listed in Table 16. Only 24.1% studies (13 out of 54 studies, and one study may use several tools) use tools, which shows that tools are not widely used to support the combination. These tools are all generic tools that can be used for a variety of purposes. We have not found any dedicated tools that are specifically built for the purposes of supporting the architecture-agility combination.

"T1: Board" and "T2: Wiki" are each reported in 4 out of 54 studies (7.4%). The concept of board is independent of the actual formats, e.g., electronic or traditional blackboard. In this SMS, we define all of them as "Board". In [S2], whiteboards were used to share and explain architectural design decisions during agile development, while the architecture design with design decisions was kept on the board until it was implemented. In [S2], agile teams also used Wiki to record and share architectural design decisions.

"T3: Spreadsheet" (2 out of 54 studies, 3.7%) and "T4: XML File" (2 out of 54 studies, 3.7%) lie in the third position. In [S4], spreadsheet was used to map architectural concerns to development phases, in order to adjust the generic, waterfall-inspired milestones to fit the agile process. The mapping acts as a roadmap to make clear to stakeholders when they should expect to engage with the development team. In [S30], XML files were used to describe architecture in components and interfaces, and the architecture descriptions in XML were used for "on-demand" generation (automatic extraction) of architectural views in agile development.

Note that we did not list all the tools discussed or used in the selected studies as supporting tools for the architecture-agility combination, because some tools provided no information on how exactly they were used to support the combination. Such tools include

Table 16
Classification of supporting tools for the architecture-agility combination.

ID	Tool	Description	Studies
T1	Board	It was used to record and communicate architectural information (e.g., design and design decisions) in agile development.	[S2][S27][S39][S40]
T2	Wiki	It was used to document and share architectural information (e.g., architecture design decisions) in agile development.	[S2][S46][S48][S52]
T3	Spreadsheet	It was used to record architectural information in agile development, for example, mapping architectural concerns to development phases in order to fit the agile process.	[S4][S12]
T4	XML file	It was used to record architectural information in agile development.	[S30][S45]
T5	Abstract specification tool	It was used to help architects in organizing relevant information regarding architecture while creating design and architecture blueprints in agile development.	[S22]
T6	Flip chart	It was used to record certain architectural information (e.g., design problems) in agile development.	[S27]
T7	Reputation system	It was used to improve architecture documentation in agile development, through continuously analyzing the contributions (i.e., writing good documentation) and publishing the reputation scores of developers.	[S44]

SoMox [S16], AEvol [S16], MagicDraw [S21], Office Word [S21], Software Configuration Management System [S30], Mind Mapping Tool [S34], Flexible-PLA [S43], CruiseControl [S45], and Apache ANT [S45].

4.10. RQ9: What are the lessons learned from the architecture-agility combination?

We classified the lessons learned from the architecture-agility combination into three types with twenty-five subtypes, as presented in Table 17. "LL1: Architecture" denotes the lessons learned related to architecture in the architecture-agility combination. "LL2: People" refers to the lessons learned concerning various roles in agile development (e.g., architects). "LL3: Others" covers the lessons learned which cannot be classified into the aforementioned types.

5. Discussion

The selected studies were published in thirty-one venues, which indicates that the architecture-agility combination has been a widespread concern in the software engineering community covering a large part of the lifecycle. The number of studies has been increasing over the last thirteen years with an exceptional peak in 2010. IEEE Software (10 out of 54 studies, 18.5%), a popular magazine for software practitioners, tops the list of the publication venues of selected studies (see Table 6), which shows that this topic is interesting and a major concern for practitioners. The data shown in Fig. 5 also supports this in that 52% (28 out of 54) of the selected studies have at least one author from industry.

The results of this SMS demonstrate that a significant difference exists in the proportion of architecting activities (Fig. 6), agile methods (Fig. 8), and agile practices (Table 10) that can be employed in the architecture-agility combination: some have been sufficiently investigated while others not at all. For example, "Test-Driven Development" and "Pair Programming" are frequently employed agile practices in agile development (Jalali and Wohlin, 2012; Sfetsos and Stamelos, 2010), however, these agile practices did not receive much attention in the combination.

In this section, we further discuss the study results for the various research questions, except for the challenges (RQ6) and lessons learned (RQ9) of the combination, which are discussed afterwards in the context of implications for researchers and practitioners.

5.1. Analysis and synthesis of results

Architecting activities: Most research effort has been put on Architectural Description (29 out of 54 studies, 53.7%) in the architecture-agility combination. This is not a surprising result. Architecture has to be described to a certain extent if stakeholders want to use it in agile development; as Kruchten suggested "*if an architecture is not written, it does not exist*" (Kruchten, 2009). Other architecting activities received quite limited attention in the architecture-agility combination, e.g., Architectural Analysis was only used in six (11.1%) selected studies (see Fig. 6). One reason may be that some architecting activities (e.g., Architectural Impact Analysis) require significant effort and time, which is not suitable for the context of agile development.

Architecting approaches: We collected and identified forty-three architecting approaches used in agile development (see Table 9) from the selected studies. These architecting approaches can be used with agile practices such as "Iterative and Incremental Development", "Just Enough Work", and "Continuous Integration". However, none of the approaches has been widely used in the combination, for example "Iterative Architecture Approach", which tops the list, has only been used in four selected studies. The reasons are the following: (1) the forty-three architecting approaches are dedicated to support ten architecting activities in the combination (see Fig. 7 and Table 9). For example, Abstract Architecture Specification Document [S22] is used to support ADp; (2) different approaches used to support the same architecting activity have different focuses. For example, Plastic Partial Component [S14][S43] and Architectural Design Pattern [S25] are used to specify different elements (i.e., component vs. pattern) in ADp; (3) different contexts of the architecture-agility combination require approaches that specifically match the context. For example, Agile Product-Line Architecting [S14][S15] is proposed and used for product-line development.

Agile methods: Scrum and XP are the most popular agile methods used in the architecture-agility combination (see Fig. 8). This is not surprising, because several recent surveys on agile development have confirmed the same result (e.g., (Ali, 2012; Rodríguez et al., 2012; Stavru, 2014)), which partially contributes to the popularity of agile methods employed in the combination. Other agile methods like LSD, FDD, and TDD received much less attention in the combination.

Agile practices: We collected and identified forty-one agile practices (in Table 10) from the selected studies. However, only a few of them have been widely discussed (such as "Backlog", "Iterative and Incremental Development", "Sprint", "Continuous Integration", and "Just Enough Architectural Work"). One reason is that these agile practices (e.g., "Iterative and Incremental Development") are the common characteristics employed by various agile development methods (Abrahamsson et al., 2010; Boehm et al., 2010; Diaz et al., 2014). Another reason is that some agile practices are highly linked to specific agile methods (e.g., "AP35: Poka-Yoke" is used in the lean manufacturing process with only one study). How to apply other popular agile practices (e.g., "User Story" and "Refactoring") in the architecture-agility combination can be an interesting research direction. Another issue is that the employment of agile practices in the architecture-agility combination is heavily based on personal experience and knowledge; there is a lack of guidance on how to use these practices in the combination. In this SMS, some examples are provided that use the most frequently discussed agile practices from the selected studies. However, this area is far from mature.

Costs: We identified seven types of costs (in Table 11) in the architecture-agility combination, all of them being related to architecture, which indicates that architecture-related costs (e.g., caused by architecting activities and approaches) are the major source of the extra effort in the architecture-agility combination. Since "pure" agile development does not involve such costs, this is a major concern for using architecture in an agile development team.

Benefits: In this SMS, nineteen general benefits and twenty-one specific benefits, which are attached to specific approaches, methods, and tools, were collected and identified. Benefits may not come for free, and can lead to certain costs in the architecture-agility combination. For example, in [S2], the author found that agile development could save effort on architectural documentation activity (the benefit), however, it may result in "Architectural Knowledge Learning Cost" and "Architectural Maintenance Cost" (see Table 11).

Factors: Six types of factors with twenty-nine subtypes (see Table 15), which can impact the architecture-agility combination were collected and classified. However, these factors are not of same importance or priority for consideration in development. This leads to the question about which factors should be considered first and paid more attention. The top ten factors shown in Fig. 9 are the most frequently-mentioned factors to the success of applying the combination, which can be considered with a high priority. Other questions, such as how these factors impact the combination and what solutions can be considered when dealing with a specific factor (e.g., project size) in the combination, are also important.

Tools are important to facilitate the architecture-agility combination. However, there are only seven general tools (e.g., Wiki) that can be used to support the combination (see Table 16). There are no existing tools built on purpose to support the architecture-agility combination.

Table 17
Classification of lessons learned of the architecture-agility combination.

ID	Type of lessons learned	Subtype	Description	Studies
LL1	Architecture	Architecture design	Two representative lessons learned: (1) it is better to focus on architecture design early enough (in early iterations); (2) stakeholders should decide when to freeze the architecture in order to provide architecture stability for developers to finish a product release, and be aware of the amount of technical debt accumulated.	[S1][S2][S6][S8][S10][S15] [S21][S24][S32][S34][S35] [S37][S39][S41][S42][S43] [S44][S46][S48][S52] [S53][S54]
		Architectural Description	Two representative lessons learned: (1) healthy agile projects welcome the iterative delivery of documentation (e.g., architecture documentation with the working software); (2) agile architectural modeling is better conducted in an incremental, customer-involved process, which may lower the modeling overhead.	[S1][S16][S18] [S29][S37][S42] [S43][S45][S46] [S48][S52]
		Architecture quality	Stakeholders should keep architecture simple, vital, transparent, and flexible.	[S5][S10][S21] [S48][S52]
		Architectural design decision	Architecturally significant decisions should be made early and mainly before implementation, and these decisions should be explicitly documented during agile development.	[S1][S6][S24] [S41][S49]
		Architectural Evaluation	In agile development, it is better to (1) review architecture documentation as it emerges from each sprint, and (2) evaluate architecture by an external team.	[S37][S38][S41]
		Architectural Refactoring	Architectural refactoring is frequently used in agile development to construct architecture incrementally.	[S10][S24][S39]
		Architectural Analysis	Architecture-agility combination enables agile teams to focus on both functional and non-functional requirements, and capturing ASRs as acceptance test cases can improve the visibility of ASRs.	[S1][S39]
		Architectural knowledge management	In agile development, it is better to (1) keep code as the first-class citizen and then get as much architecture information as possible under this premise, and (2) always reflect up-to-date architectural information.	[S11][S17]
		Architectural Maintenance	In agile development, a uniform understanding of the existing architecture is required for making and implementing consistent architectural changes.	[S27]
		Architectural Understanding	In agile development, stakeholders should not assume a tacit, implicit architectural understanding, and they should understand the interdependencies between architectural issues and user functionality to connect them over iterations.	[S1]
		Architectural Implementation	In agile development, keeping the architecture documents consistent with the implementation is fundamental to dealing with complex systems with changing requirements.	[S34]
		Architectural issues tracking	In agile development, it is important to track unresolved architectural issues in both the backlog and the risks.	[S1]
		Architectural pattern	In XP, architectural patterns are most useful when developers are familiar with them and use them in a disciplined way.	[S25]
		Architecture-centric development	Guidelines for combining architecture-centric development and agile development: (1) develop architecture in a lightweight way; (2) keep the architecture flexible to change; (3) use test-driven architectural design at the interface level; (4) module users instead of module providers write tests.	[S45]
		Semantics of architecture concept	The concept of architecture often has fuzzy boundaries for various stakeholders. An agreement on the definition of architecture is a prerequisite for using architecture in agile development.	[S1]
LL2	People	Architect	Three representative lessons learned: (1) clearly define architect's role and responsibility in agile development; (2) do not let architects lock themselves in an ivory tower, and architects should be part of the development group; (3) agile architects must be educated so that even the most intractable techie will at least be understood, even if the techie do not agree with the architectural decisions.	[S1][S4][S9] [S18][S24][S37] [S46][S47][S48] [S53]
		Development team	Two representative lessons learned: (1) the development team needs to be involved in the communication and decision process, and constant merging through Continuous Integration reduces the complexity and risk considerably; (2) the development team should keep stakeholders' concerns and any other rationale information they capture in a format they can include in the architecture documentation – usually on a public whiteboard.	[S18][S26][S38] [S40][S47][S53]
		Developer	In agile development, developers are equally important as other stakeholders, and they should consider architecture relevant to their day-to-day activities. They should collaborate with architects early and often, and continuously share ideas.	[S1][S4]
		Stakeholder	Stakeholders should consider the potential impact on architecture when introducing agile methods in their organizations. It is also important to have an "architecture representative" counterpart who is responsible for the system qualities to keep a balance between the functional and non-functional requirements.	[S2][S8]
LL3	Others	Approach of the architecture-agility combination	Two representative lessons learned: (1) it is necessary to consider the potential value clash the development teams may perceive, when combining architecture and agile approaches; (2) the approaches that support the combination should be lightweight.	[S1][S4][S8] [S21][S23][S24] [S35][S37][S39] [S46][S49]
		Large project	Two representative lessons learned: (1) agile methods should fit the specific context of the development of large and complicated projects, which require significant architectural effort; (2) architecture-centric methods can add value to agile development by adapting agile methods using a hybrid approach to handle large and complex systems.	[S1][S24][S40]
		System quality	Three representative lessons learned: (1) stakeholders may better manage the cost (e.g., rework cost) of the project by monitoring the system qualities through the architecture (e.g., represented in technical debt or rework-related defects); (2) a shared architectural vision helps to ensure integrity and improves the software's inner quality; (3) architecture-centric methods can facilitate agile development by emphasizing system qualities.	[S39][S40][S48]
		Coordination	For every problem not handled by the architecture, a process coordination mechanism needs to be put in place to allow teams to release the system. Stakeholders should use architecture as a mechanism for coordination, and through close coordination, architects achieve the confidence of the developers that they can effectively support them.	[S7][S18]
		Communication	Communication in the architecture-agility combination should be direct and stimulated all the time during the project. It requires management attention to convince everybody that communication of architectural issues is important and intended.	[S18]
		Metaphor	A good metaphor may: (1) represent a single perspective; (2) deal with only one type of structure; (3) provide clear guidance concerning design decisions; (4) shed light on system properties.	[S29]

5.2. Comparison of architecting in agile and non-agile development

In this section, we compare and discuss architecting in agile and non-agile development (in Table 18). By doing so we can better understand what are the differences when architecting in agile development compared to non-agile, based on the results of this SMS. We analyze and discuss three major architecting activities (Architectural Analysis, Architectural Evaluation, and Architectural Description) as well as their related approaches. The other eight architecting activities (in Section 2.1) are not compared and discussed in this section because these architecting activities are rarely discussed in the architecture-agility combination, so we have very sparse data for the comparison. Each of the three analyzed architecting activities is discussed along a number of dimensions.

5.3. Implications for researchers

(1) Not all the architecting activities (see Section 2.1) are suitable for the architecture-agility combination. Researchers need to explore the benefits, costs, and other factors when proposing the introduction of architecting activities into the architecture-agility combination.
(2) As shown in Figs. 6 and 7, no selected studies employed Architectural Recovery (AR) in agile development, while there are no specific architecting approaches identified in this SMS that support AR in agile development. One possible reason is that most AR approaches are performed manually, and thus may be too expensive to use in an agile context. We suggest that researchers explore the feasibility of applying AR with (semi-)automated and lightweight approaches in agile development.
(3) There is a lack of guidance of using agile practices with architecture. There is a need for researchers to further conduct more research to provide such guidelines.
(4) Architecture-related costs are a major concern for using architecture in agile development. How to alleviate such costs (see Table 11) needs further investigation. We also encourage researchers to identify more costs (including the architecture-related costs as well as other types of costs) in the architecture-agility combination.
(5) The benefits of the architecture-agility combination are heavily based on the context of projects. We suggest that researchers may clarify the context by further exploring the factors that have impact on the benefits of the combination.
(6) The challenges of the architecture-agility combination (in Table 14) collected and identified in this SMS are potential future directions for researchers. For example, how to use the architecture-agility combination to support the development of large and distributed projects.
(7) There is a lack of guidance of using the factors (in Table 15) to facilitate the architecture-agility combination. Researchers may further classify these factors into sub-factors which are more precise and easier to consider in development, and develop the guidance of using these factors in the combination. For example, considering the "Communication" factor, which stakeholders should be involved in what kind of communication in development that may facilitate the combination?
(8) There is a lack of dedicated tools that support the architecture-agility combination. We encourage researchers to come up with such prototype tools, especially in collaboration with industry partners who apply the combination in their projects.
(9) The lessons learned collected in this SMS are mostly based on the personal experience and knowledge from the authors of the selected studies; we cannot claim that they are generalizable to other contexts. This calls for consolidated evaluation of the lessons learned, in order to convert "lessons learned" into "best practices".
(10) We found that most studies either emphasize that the architecture-agility combination is successfully applied or do not mention the result at all. We invite researchers to report on failures in addition to success stories: failures could help identify the problems and weaknesses of existing work, and consequently improve the processes, approaches, and tools that support the architecture-agility combination.

5.4. Implications for practitioners

(1) According to the results of this SMS, only Architectural Description (ADp) has been discussed in more than 50% selected studies. Therefore, in a statistical sense, it is more reliable for practitioners to use ADp and associated approaches in their agile projects than other architecting activities and approaches (e.g., AS, ARu). However, how much architectural description is enough and appropriate in agile development is largely dependent on the factors identified in this SMS (see Section 4.8).
(2) There are forty-three architecting approaches identified in this SMS (see Table 9), which are used to support most of the architecting activities (see Fig. 7). We suggest that practitioners choose suitable architecting approaches according to the architecting activities used in their agile projects. For example, architectural evaluation can be introduced in an agile project by selecting an appropriate architecting approach listed in Table 9.
(3) We suggest that practitioners report equally on both the benefits and costs of the architecture-agility combination. The cost-benefit ratio should be explicit in a project in order to decide how much of the architecture-agility combination can be used. We also advise that practitioners reflect critically on the benefits of the architecture-agility combination, especially with respect to the specific context of their project.
(4) The challenges of the architecture-agility combination collected and identified in this SMS can be considered as a checklist for practitioners. For example, if practitioners want to introduce ADp into agile development, they need to check against the related challenges, e.g., how much architecture and what architectural information should be described and documented in the projects.
(5) The factors identified in this SMS (see Table 15) play a significant role in the architecture-agility combination. We hope to make practitioners aware of the importance of the various factors as they may determine the success of their projects.
(6) As aforementioned in the implications for researchers (Section 5.2), there is a lack of dedicated tools that support the architecture-agility combination. Practitioners can use general tools (e.g., Excel, Wiki) in their projects, which provide certain level of support to the combination. However, practitioners can also collaborate with researchers to make explicit their needs and requirements for dedicated tools.

6. Threats to validity

Study search: There may be relevant studies that were not retrieved, which may affect the results of this SMS. To mitigate this threat, we searched the most popular electronic databases on software engineering and finally acquired 54 studies. In addition, before the formal search, we performed a trial search to improve the correctness and completeness of the search results. With these measures, the probability of missing relevant studies has been reduced.

Study selection largely depends on personal knowledge and experience of the researchers who conducted this study, which might

Table 18
Comparison of architecting in agile and non-agile development.

Architecting activity	Dimension	Agile development	Non-agile development
Architectural Analysis (AA)	Basis	AA in agile development is user story based (e.g., Architecture-related Tasks Visualization [S39]), because user stories are short and simple with one specific need, which is suitable for conducting AA in an agile context. One template of user story is: "*As a <user role> I want <goal> so that <business value>*".	AA in non-agile development is scenario-based (e.g. ADD) (Bass et al., 2012, Hofmeister et al., 2007, Barbacci et al., 2003). For example, a quality attribute scenario contains source of stimulus, stimulus, environment, artifact, response, and response measure (Bass et al., 2012). This requires collecting and recording more elaborate information than user stories.
	Customer participation	AA in agile development encourages customer participation, thus this activity is about keeping customers close-by (e.g., [S2]).	There is no emphasis on participation of the customer but a general involvement of different types of stakeholders.
	Process	AA in agile development is iteratively and incrementally conducted (e.g., ADD [S40]).	The emphasis is on the intertwinement between requirements engineering and architecting (i.e., Architectural Analysis and Synthesis), also known as the Twin Peaks (Nuseibeh, 2001).
	Focus	AA in agile development pays more attention to functional requirements than non-functional requirements (e.g., [S43]).	AA in non-agile development also pays attention to functional requirements but is largely driven by non-functional requirements (Hofmeister et al., 2007).
Architectural Evaluation (AE)	Process	AE in agile development is iteratively and incrementally performed similarly to AA (e.g., Iterative/Incremental Architecture Evaluation [S31]).	AE in non-agile development can be performed in different phases such as with the final version of the SA (e.g., Software Architecture Analysis Method (Dobrica and Niemelä, 2002)), or through an iterative process (e.g., ATAM (Dobrica and Niemelä, 2002)), or with initial design (e.g., Active Reviews for Intermediate Design (Babar et al., 2004)).
	Goal	The goal of AE in agile development concerns assessing architectural decisions (e.g., [S40]), identifying architectural risks, and finding solutions to these risks (e.g., [S41]).	The goal of AE in non-agile development contains system qualities assessment through quality scenarios, maintenance cost prediction, identifying architectural risks, tradeoffs, and the boundaries of the system (Babar et al., 2004).
	Approach	Most AE approaches used in agile development are based on adapting resource-intensive methods in a simple and lightweight way (e.g., Hybrid Architecture Evaluation Method [S28] and Simplified ATAM [S20]).	The AE approaches used in non-agile development are mostly scenario-based (e.g., Architecture Tradeoff Analysis Method (Babar et al., 2004, Dobrica and Niemelä, 2002), Scenario-based Architecture Analysis Method (Babar et al., 2004, Dobrica and Niemelä, 2002), and Architecture Level Prediction of Software Maintenance (Babar et al., 2004, Dobrica and Niemelä, 2002)).
Architectural Description (ADp)	Role	There are several specific roles related to ADp in agile development. For example, "Architecture Representative", who is responsible for architectural information (e.g., elaborating architectural tasks from system quality requirements in a backlog) [S18].	There are different types of architects, depending on the domain and context, such as system architect, enterprise architect, solution architect, data architect etc. All these types can potentially perform ADp.
	Architecture viewpoints and process	Architecture viewpoints are rarely used to describe architecture in agile development (5 out of 54, 9.3%). ADp in agile development is iteratively and incrementally conducted (e.g., Release Development based on Architecture [S24]). ADp in agile development emphasizes lightweight, quickly producing, machine-readable, simple, and flexible (e.g., [S21]).	Architecture viewpoints have been introduced as a standard practice for ADp in general (including non-agile) software development (ISO, 2011, Clements et al., 2010). In non-agile approaches ADp is primarily performed based on architecture frameworks (e.g., TOGAF (Josey et al., 2011)), which are composed of a set of architecture viewpoints.
	Backlog	Backlog is an important element of ADp in agile development. In [S17], the product backlog was used with an Architectural Knowledge Base to create an architectural document, which explicitly presents the architectural design to the stakeholders.	Backlog is used in non-agile development to record the outputs of various architecting activities for incremental architectural design (Hofmeister et al., 2007). Recently it has been suggested that backlog should also be used to record technical debt items (Kruchten et al., 2012).
	ADp and system implementation	In agile development, it is recommended to make architecture tangible for architects and developers by linking ADp to the source code through e.g., embedding architectural information into source code comments [S11].	There are approaches that link architecture artifacts to the implementation through an implementation viewpoint (Clements et al., 2010, Kruchten, 1995). However, lack of traceability between architecture documents and code is reported often as a problem (Manteuffel et al., 2014).
	Effort	The effort of ADp in agile development is kept at a minimum through several ways: (1) avoiding redundant information (e.g., [S2]) and focusing on the required information (e.g., [S22]), (2) producing individual short documents (e.g., [S49]), (3) using architecture document templates (e.g., [S25]), and (4) minimizing up-front ADp (e.g., [S52]).	ADp in non-agile development is often reported to produce redundant SA documentation (Su, 2010), which is often not up-to-date and lacks utility (Rost et al., 2013).
	Architectural description language	ADp in agile development uses mostly natural language and XML (e.g., [S30]).	The predominant architectural description language used for ADp in non-agile development is UML (Malavolta et al., 2012).
	Tool	The most frequently used tools for ADp in agile development are board (e.g., [S40]) and wiki (e.g., [S48]).	There are various tools that support ADp in non-agile development (see (Shahin et al., 2014, Ding et al., 2014) for an overview).

introduce bias to the selection results. We used the following strategies to mitigate this bias.

(1) A set of inclusion and exclusion criteria for study selection were provided as a basis of an objective selection process.
(2) Considering the possible different interpretation and understanding of selection criteria by the researchers, a pilot selection was conducted before the formal selection to guarantee that the researchers reached a clear and consistent understanding of the selection criteria.
(3) Two researchers conducted the study selection independently in the second and third round of selection, and discussed and resolved any conflicts between their results, in order to mitigate personal bias in study selection.

Data extraction results might be negatively affected due to the bias of researchers who extract the data. Bias on data extraction may result in inaccuracy of the extracted data items, and further affect the results of data synthesis (the next step). This bias was mitigated with four measures:

(1) A list of extracted data items (in Table 5) was specified in detail to reduce possible misunderstandings on the data items to be extracted.
(2) Discussions on extracted data items among the researchers throughout the whole data extraction process were used to improve the consistency and correctness of data extraction results.
(3) A pilot data extraction was performed before the formal data extraction by two researchers to mitigate personal bias on data extraction.
(4) The results of the data extraction were also checked by two researchers and confirmed by the third researcher to reduce personal bias.

Data synthesis might be negatively affected in two aspects. One is bias on data synthesis: researchers may have different understanding on the extracted data, for example, a benefit of the architecture-agility combination was classified into the type "People" by one researcher, but was classified into the type "Architecture" by another researcher. This bias was mitigated through continuous discussions among the researchers. The second aspect is lack of information on the extracted data. For example, the author of [S16] mentioned that SoMox was used as a tool in the architecture-agility combination, without any further information about the tool (e.g., how this tool was used in the combination). If we could not find any detailed information on the extracted data, we excluded them in data synthesis.

7. Conclusions

In this SMS, we searched the papers in seven electronic databases published between Feb. 2001 and Jan. 2014. Based on the data extracted from fifty-four selected studies, this SMS provides a comprehensive overview of the state of research on the architecture-agility combination. The main points of the study are as follows:

(1) The research on the architecture-agility combination received significant attention according to the number of the selected studies over time period (see Fig. 4). This tendency shows that an increasing effort has been made in this area, especially in the last three years. Most selected studies (44 out of 54, 81.5%) are evidenced by industrial projects, surveys, or experience (see Table 8).
(2) Eleven architecting activities (see Section 2.1) were collected in this SMS. However, these architecting activities received significantly different levels of attention (see Fig. 6). Architectural Description is the most frequently discussed activity in the architecture-agility combination. In contrast, the activities Architectural Refactoring, Architectural Impact Analysis, Architectural Implementation, Architectural Synthesis, and Architectural Reuse received much less attention in the combination. Architectural Recovery has never been used in the combination in the selected studies.
(3) Forty-three architecting approaches employed in the architecture-agility combination were identified from the selected studies. However, none of them has been widely used (see Table 9). For example, the most frequently used architecting approach "Iterative Architecture Approach" was only discussed in four studies. Most of the approaches (32 out of 43, 74.4%) were only used in one study (see Table 9).
(4) Ten agile methods were collected in this SMS (see Fig. 8). Scrum and XP have received much more attention than other agile methods (e.g., Crystal and Test-Driven Development) in the architecture-agility combination.
(5) Forty-one agile practices used in the architecture-agility combination were collected and identified in this SMS (see Table 10). However, only several agile practices have been widely discussed (e.g., "Backlog", "Iterative and Incremental Development", "Sprint", "Continuous Integration", and "Just Enough Architectural Work").
(6) Seven types of costs in the architecture-agility combination were collected and classified from the selected studies (see Table 11). We did not acquire much data (description or discussion) about the costs of the architecture-agility combination, because most studies tend to present the benefits of the combination instead of the costs.
(7) The benefits of the architecture-agility combination are classified into two categories - general and specific benefits, and each of the categories has been further classified into types and subtypes. Five types (Architecture, People, Reduced Cost, People & System, and Others) with nineteen subtypes were identified for the general benefits (see Table 12), and four types (Architecture, People, Agile Development, and Others) with twenty-one subtypes were classified for the specific benefits.
(8) Challenges of the architecture-agility combination are grouped into two types (Architecture and Others) with twenty subtypes (see Table 14). Most of the challenges are directly related to architecture (e.g., "Tension between Architecture Design and Agile Development"). Several other challenges are also important, for example, "Champion of the Architecture-Agility Combination" is the most important challenge in the combination, which was mentioned in almost one third of the selected studies (16 out of 54, 29.6%).
(9) Six types (Project, People, Architecture, People-related, Organization, and System) with twenty-nine subtype factors that may impact the success of applying architecture-agility combination were collected and classified in this SMS (see Table 15). "Communication" between all the stakeholders (e.g., architects, the development team, and customers) is the most important factor (see Fig. 9) to the success of the architecture-agility combination.
(10) Most selected studies either emphasized that the architecture-agility combination was successful or did not mention the combination result. Therefore, we did not collect much data (description or discussion) about failure stories of the combination. This is in line with the result of point (7) about the lack of cost data.
(11) Seven tools that support the architecture-agility combination were collected from the selected studies (see Table 16). However, none of them is dedicated to supporting the architecture-agility combination, which means all the seven tools are borrowed from other fields of software development (e.g., Wiki is from agile development).

(12) The lessons learned from the architecture-agility combination have been classified into three types (Architecture, People, and Others) with twenty-five subtypes (see Table 17). "Architecture Design" (one subtype of lessons learned) received much more attention (22 out of 54 studies, 40.7%) than the other types of lessons learned (e.g., "Architectural Maintenance", "Developer", and "System Quality").

To conclude, this SMS can act as a foundation for future research directions on combining architecture and agile development, and help practitioners to select appropriate architecting activities, approaches, and agile practices.

Acknowledgments

This work is partially sponsored by the NSFC under Grant No. 61472286 and the Ubbo Emmius scholarship program by the University of Groningen.

Appendix A. Selected studies

[S1] P. Abrahamsson, M.A. Babar, and P. Kruchten. Agility and architecture: Can they coexist?. IEEE Software, 27(2):16-22, 2010.

[S2] M.A. Babar. An exploratory study of architectural practices and challenges in using agile software development approaches. In: Proceedings of the Joint 8th Working IEEE/IFIP Conference on Software Architecture & 3rd European Conference on Software Architecture (WICSA/ECSA), Cambridge, UK, pp. 81-90, 2009.

[S3] S. Bellomo, R.L. Nord, and I. Ozkaya. A study of enabling factors for rapid fielding combined practices to balance speed and stability. In: Proceedings of the 35th International Conference on Software Engineering (ICSE), San Francisco, CA, USA, pp. 982-991, 2013.

[S4] S. Blair, T. Cull, and R. Watt. Responsibility driven architecture. IEEE Software, 27(2):26-32, 2010.

[S5] B. Boehm, J.A. Lane, S. Koolmanojwong, and R. Turner. Architected Agile Solutions for Software-Reliant Systems, in Agile Software Development. Springer Berlin Heidelberg, pp. 165-184, 2010.

[S6] G. Booch. An architectural oxymoron. IEEE Software, 27(5):95-96, 2010.

[S7] J. Bosch and P. Bosch-Sijtsema. Coordination Between Global Agile Teams: From Process to Architecture, in Agility Across Time and Space. Springer Berlin Heidelberg, pp. 217-233, 2010.

[S8] N. Brown, R. Nord, and I. Ozkaya. Enabling agility through architecture. CrossTalk, 13(6):12-17, 2010.

[S9] F. Buschmann and K. Henney. Architecture and agility: Married, divorced, or just good friends?. IEEE Software, 30(2):80-82, 2013.

[S10] H. Chivers, R.F. Paige, and X. Ge. Agile security using an incremental security architecture. In: Proceedings of the 6th International Conference on Agile Processes in Software Engineering and Extreme Programming (XP), Sheffield, UK, pp. 57-65, 2005.

[S11] H.B. Christensen and K.M. Hansen. Towards architectural information in implementation. In: Proceedings of the 33rd International Conference on Software Engineering (ICSE), Waikiki, Honolulu, Hawaii, pp. 928-931, 2011.

[S12] H.B. Christensen, K.M. Hansen, and B. Lindstrom. Lightweight and continuous architectural software quality assurance using the aSQA technique. In: Proceedings of the 4th European Conference on Software Architecture (ECSA), Copenhagen, Denmark, pp. 118-132, 2010.

[S13] J. Cleland-Huang, A. Czauderna, and E. Keenan. A persona-based approach for exploring architecturally significant requirements in agile projects. In: Proceedings of the 19th International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ), Essen, Germany, pp. 18-33, 2013.

[S14] J. Diaz, J. Perez, and J. Garbajosa. Agile product-line architecting in practice: A case study in smart grids. Information and Software Technology, 56(7):727-748, 2014.

[S15] J. Diaz, J. Perez, J. Garbajosa, and A. Yague. Change-impact driven agile architecting. In: Proceedings of the 46th Hawaii International Conference on System Sciences (HICSS), Wailea, HI, USA, pp. 4780-4789, 2013.

[S16] Z. Durdik. Towards a process for architectural modelling in agile software development. In: Proceedings of the Joint 7th International Conference on the Quality of Software Architectures & 2nd International Symposium on Architecting Critical Systems (QoSA/ISARCS), Boulder, Colorado, USA, pp. 183-192, 2011.

[S17] V.P. Eloranta and K. Koskimies. Aligning architecture knowledge management with Scrum. In: Proceedings of the Joint 10th Working IEEE/IFIP Conference on Software Architecture & 6th European Conference on Software Architecture (WICSA/ECSA), Helsinki, Finland, pp. 112-115, 2012.

[S18] R. Faber. Architects as service providers. IEEE Software, 27(2):33-40, 2010.

[S19] D. Falessi, G. Cantone, S.A. Sarcia, G. Calavaro, P. Subiaco, and C. D'Amore. Peaceful coexistence: Agile developer perspectives on software architecture. IEEE Software, 27(2):23-25, 2010.

[S20] S. Farhan, H. Tauseef, and M.A. Fahiem. Adding agility to architecture tradeoff analysis method for mapping on crystal. In: Proceedings of the WRI World Congress on Software Engineering (WCSE), Xiamen, China, pp. 121-125, 2009.

[S21] E. Hadar and G.M. Silberman. Agile architecture methodology: Long term strategy interleaved with short term tactics. In: Proceedings of the 23rd ACM SIGPLAN Conference on Object-Oriented Programming Systems Languages and Applications (OOPSLA), Nashville, Tennessee, USA, pp. 641-652, 2008.

[S22] I. Hadar, S. Sherman, E. Hadar, and J.J. Harrison. Less is more: Architecture documentation for agile development. In: Proceedings of the 6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE), San Francisco, CA, USA, pp. 121-124, 2013.

[S23] T. Hayata, J. Han, and M. Beheshti. Facilitating agile software development with lean architecture in the DCI paradigm. In: Proceedings of the 9th International Conference on Information Technology: New Generations (ITNG), Las Vegas, NV, USA, pp. 343-348, 2012.

[S24] T. Ihme. Scrum adoption and architectural extensions in developing new service applications of large financial IT systems. Journal of the Brazilian Computer Society, 19(3):257-274, 2013.

[S25] T. Ihme and P. Abrahamsson. The use of architectural patterns in the agile software development of mobile applications. In: Proceedings of the International Conference on Agility (ICAM), Helsinki, Finland, pp. 155-162, 2005.

[S26] M. Isham. Agile architecture IS possible - You first have to believe!. In: Proceedings of Agile Conference (AGILE), Toronto, ON, Canada, pp. 484-489, 2008.

[S27] R.N. Jensen, N. Platz, and G. Tjornehoj. Developer stories: Improving architecture in agile practice. In: Proceedings of the 2nd International Conference on Software & Data Technologies (ICSOFT/ENASE), Barcelona, Spain, pp. 172-184, 2007.

[S28] F. Kanwal, K. Junaid, and M.A. Fahiem. A hybrid software architecture evaluation method for FDD - An agile process model. In: Proceedings of the International Conference on Computational Intelligence and Software Engineering (CiSE), Wuhan, China, pp. 1-5, 2010.

[S29] M. Keeling and M. Velichansky. Making metaphors that matter. In: Proceedings of Agile Conference (AGILE), Salt Lake City, UT, USA, pp. 256-262, 2011.

[S30] T. Keuler, S. Wagner, and B. Winkler. Architecture-aware programming in agile environments. In: Proceedings of the Joint 10th Working IEEE/IFIP Conference on Software Architecture & 6th European Conference on Software Architecture (WICSA/ECSA), Helsinki, Finland, pp. 229-233, 2012.

[S31] R.J. Koontz and R.L. Nord. Architecting for sustainable software delivery. CrossTalk, 15(3): 14-19, 2012.

[S32] P. Krogdahl, G.Luef, and C. Steindl. Service-oriented agility: An initial analysis for the use of agile methods for SOA development. In: Proceedings of the 2nd IEEE International Conference on Services Computing (SCC), Orlando, FL, USA, pp. 93-100, 2005.

[S33] P. Kruchten. Contextualizing agile software development. Journal of Software: Evolution and Process, 25(4):351-361, 2013.

[S34] I.H. Krueger, M. Meisinger, M. Menarini, and S. Pasco. Rapid systems of systems integration - Combining an architecture-centric approach with enterprise service bus infrastructure. In: Proceedings of the 7th International Conference on Information Reuse and Integration (IRI), Waikoloa, Hawaii, USA, pp. 51-56, 2006.

[S35] V. Krunic. Agile architecture - Changing application servers. In: Proceedings of the Conference on Agile (AGILE), Washington, DC, USA, pp. 162-168, 2007.

[S36] E. Landre, H. Wesenberg, and J. Olmheim. Agile enterprise software development using domain-driven design and test first. In: Proceedings of the 22nd Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems Languages and Applications (OOPSLA), Montréal, Québec, Canada, pp. 983-993, 2007.

[S37] J. Madison. Agile architecture interactions. IEEE Software, 27(2):41-48, 2010.

[S38] R.L. Nord, N. Brown, and I. Ozkaya. Architecting with just enough information. In: Proceedings of the 6th International Workshop on SHAring and Reusing Architectural Knowledge (SHARK), Waikiki, Honolulu, Hawaii, pp. 9-12, 2011.

[S39] R.L. Nord, I. Ozkaya, and R.S. Sangwan. Making architecture visible to improve flow management in lean software development. IEEE Software, 29(5):33-39, 2012.

[S40] R.L. Nord and J.E. Tomayko. Software architecture-centric methods and agile development. IEEE Software, 23(2):47-53, 2006.

[S41] I. Ozkaya, M. Gagliardi, and R.L. Nord. Architecting for large scale agile software development: A risk-driven approach. CrossTalk, 16(3):17-22, 2013.

[S42] L. Pareto, P.Eriksson, and S. Ehnebom. Architectural descriptions as boundary objects in system and design work. In: Proceedings of the 13th International Conference on Model Driven Engineering Languages and Systems (MoDELS), Oslo, Norway, pp. 406-419, 2010.

[S43] J. Perez, J. Diaz, J. Garbajosa, and P.P. Alarcon. Flexible working architectures: Agile architecting using PPCs. In: Proceedings of the 4th European Conference on Software Architecture (ECSA), Copenhagen, Denmark, pp. 102-117, 2010.

[S44] C.R. Prause and Z. Durdik. Architectural design and documentation: Waste in agile development?. In: Proceedings of the International Conference on Software and System Process (ICSSP), Zurich, Switzerland, pp. 130-134, 2012.

[S45] K. Read and F. Maurer. Issues in scaling agile using an architecture-centric approach: A tool-based solution. In: Proceedings of the 3rd Conference on Agile Processes in Software Engineering and Extreme Programming (XP), New Orleans, LA, USA, pp. 142-150, 2003.

[S46] A. Rendell. Descending from the architect's ivory tower. In: Proceedings of the Conference on Agile (AGILE), Chicago, IL, USA, pp. 180-185, 2009.

[S47] E. Richardson. What an agile architect can learn from a hurricane meteorologist. IEEE Software, 28(6):9-12, 2011.

[S48] J. Sauer. Architecture-Centric Development in Globally Distributed Projects, in Agility Across Time and Space. Springer Berlin Heidelberg, pp. 321-329, 2010.

[S49] J. Savolainen, J. Kuusela, and A. Vilavaara. Transition to agile development – rediscovery of important requirements engineering practices. In: Proceedings of the 18th IEEE International Conference on Requirements Engineering (RE), Sydney, NSW, Australia, pp. 289-294, 2010.

[S50] A.A. Sharifloo, A.S. Saffarian, and F. Shams. Embedding architectural practices into extreme programming. In: Proceedings of the 19th Australian Conference on Software Engineering (ASWEC), Perth, WA, Australia, pp. 310-319, 2008.

[S51] M. Staron, W. Meding, C. Hoglund, P. Eriksson, J. Nilsson, and J. Hansson. Identifying implicit architectural dependencies using measures of source code change waves. In: Proceedings of the 39th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA), Santander, Spain, pp. 325-332, 2013.

[S52] T. Stober and U.Hansmann. Considerations on Planning and Architecture, in Agile Software Development. Springer Berlin Heidelberg, pp. 93-114, 2010.

[S53] M. Waterman, J. Noble, and G. Allan. How much architecture? Reducing the up-front effort. In: Proceedings of the International Conference on Agile and Lean Software Methods (AGILE INDIA), Bengaluru, India, pp. 56-69, 2012.

[S54] M. Waterman, J. Noble, and G. Allan. The effect of complexity and value on architecture planning in agile software development. In: Proceedings of the 14th International Conference on Agile Processes in Software Engineering and Extreme Programming (XP), Vienna, Austria, pp. 238-252, 2013.

Appendix B. Abbreviations used in the study

AA	Architectural Analysis
AAP	Architecting Approach
ADD	Attribute-Driven Design
ADp	Architectural Description
AE	Architectural Evaluation
AI	Architectural Implementation
AIA	Architectural Impact Analysis
AME	Architectural Maintenance and Evolution
AP	Agile Practice
AR	Architectural Recovery
ARf	Architectural Refactoring
ARu	Architectural Reuse
AS	Architectural Synthesis
ASR	Architecturally Significant Requirement
ATAM	Architecture Tradeoff Analysis Method
AU	Architectural Understanding
DCAR	Decision Centric Architecture Review
FDD	Feature Driven Development
GB	General Benefit
IA	Change Impact Analysis

LL	Lesson Learned
LSD	Lean Software Development
PPC	Plastic Partial Component
RQ	Research Question
SA	Software Architecture
SB	Specific Benefit
SLR	Systematic Literature Review
SMS	Systematic Mapping Study
TDD	Test Driven Development
XP	Extreme Programming

References

Abrahamsson, P., Babar, M., Kruchten, P., 2010. Agility and architecture: Can they coexist? IEEE Softw. 27 (2), 16–22.

Adolph, S., Hall, W., Kruchten, P., 2011. Using grounded theory to study the experience of software development. Empir. Softw. Eng. 16 (4), 487–513.

Ali, M.A., 2012. Survey on the state of agile practices implementation in Pakistan. Int. J. Inf. Commun. Technol. Res. 2 (5), 459–464.

Arnold, R.S., 1996. Software Change Impact Analysis. IEEE Computer Society Press.

Augustine, S., 2005. Managing Agile Projects. Prentice Hall, Upper Saddle River, NJ.

Babar, M.A., 2009. An exploratory sudy of architectural practices and challenges in using agile software development approaches. In: Proceedings of the Joint 8th Working IEEE/IFIP Conference on Software Architecture & 3rd European Conference on Software Architecture (WICSA/ECSA). UK. Cambridge, pp. 81–90.

Babar, M.A., Brown, A.W., Mistrik, I., 2013. Agile Software Architecture: Aligning Agile Processes and Software Architectures.

Babar, M.A., Zhu, L., Jeffery, R., 2004. A framework for classifying and comparing software architecture evaluation methods. In: Proceedings of the 15th Australian Software Engineering Conference (ASWEC). Melbourne, Australia, pp. 309–318.

Barbacci, M.R., Ellison, R., Lattanze, A.J., Stafford, J.A., Weinstock, C.B., Wood, W.G., 2003. Quality Attribute Workshops (QAWs), third edition Technical Report CMU/SEI-2003-TR-016, SEI.

Basili, V., Caldiera, G., Rombach, D., 1994. In: Marciniak, J.J. (Ed.). The Goal Question Metric Approach, in Encyclopedia of Software Engineering. John Wiley & Sons, New York, NY.

Bass, L., Clements, P., Kazman, R., 2003. Software Architecture in Practice, 2nd edition Addison-Wesley Professional.

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

K. Beck et al. Principles behind the agile manifesto. http://www.agilemanifesto.org/principles.html, accessed on 2014-01-06.

Begel, A., Nagappan, N., 2004. Usage and perceptions of agile software development in an industrial context: an exploratory study. In: Proceedings of the 1st International Symposium on Empirical Software Engineering and Measurement (ESEM). Madrid, Spain, pp. 255–264.

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

Blair, S., Cull, T., Watt, R., 2010. Responsibility driven architecture. IEEE Softw. 27 (2), 26–32.

Boehm, B., Lane, J.A., Koolmanojwong, S., Turner, R., 2010. Architected agile solutions for software-reliant systems. Agile Software Development. Springer,, Berli n H eidelberg, pp. 165–184.

Brown, N., Nord, R., Ozkaya, I., 2010. Enabling agility through architecture. CrossTalk 13 (6), 12–17.

Cao, L., Mohan, K., Xu,, P., 2004. How extreme does extreme programming have to be? Adapting XP practices to large-scale projects. In: Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS). Big Island, Hawaii, USA, pp. 1–10.

Chen, L., Babar, M.A., 2014. Towards an evidence-based understanding of emergence of architecture through continuous refactoring in agile software development. In: Proceedings of 11th Working IEEE/IFIP Conference on Software Architecture (WICSA). Sydney, Australia, pp. 195–204.

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

Chivers, H., Paige, R.F., Ge, X., 2005. Agile security using an incremental security architecture. In: Proceedings of the 6th International Conference on Agile Processes in Software Engineering and Extreme Programming (XP). Sheffield, UK, pp. 57–65.

Christensen, H.B., Hansen, K.M., Lindstrom, B., 2010. Lightweight and continuous architectural software quality assurance using the aSQA technique. In: Proceedings of the 4th European Conference on Software Architecture (ECSA). Copenhagen, Denmark, pp. 118–132.

Clements, P., Bachmann, F., Bass, L., 2010. Documenting Software Architectures: Views and Beyond, 2nd edition Addison-Wesley Professional.

Corbin, J., Strauss, A., 2007. Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory, 3rd edition Sage Publications.

Diaz, J., Perez, J., Garbajosa, J., 2014. Agile product-line architecting in practice: a case study in smart grids. Inf. Softw. Technol. 56 (7), 727–748.

Diaz, J., Perez, J., Garbajosa, J., Yague, A., 2013. Change-impact driven agile architecting. In: Proceedings of the 46th Hawaii International Conference on System Sciences (HICSS). Wailea, HI, USA, pp. 4780–4789.

Ding, W., Liang, P., Tang, A., van Vliet, H., 2014. Knowledge-based approaches in software documentation: A systematic literature review. Inf. Softw. Technol. 56 (6), 545–567.

Dingsøyr, T., Dybå, T., Moe, N.B., 2010. Agile Software Development: Current Research and Future Directions. Springer.

Dobrica, L., Niemelä, E., 2002. A survey on software architecture analysis methods. IEEE Trans. Softw. Eng. 28 (7), 638–653.

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

Erickson, J., Lyytinen, K., Siau, K., 2005. Agile modeling, agile software development, and extreme programming: the state of research. J. Database Manag. 16 (4), 88–100.

Fowler, M., Beck, K., Brant, J., Opdyke, W., Roberts, D., 1999. Refactoring: Improving the Design of Existing Code. Addison-Wesley Professional.

Freudenberg, S., Sharp, H., 2010. The top 10 burning research questions from practitioners. IEEE Softw. 27 (5), 8–9.

Glaser, B.G., Strauss, A.L., 2009. The Discovery of Grounded Theory: Strategies for Qualitative Research. Transaction Publishers.

Hadar, I., Sherman, S., Hadar, E., Harrison, J.J., 2013. Less is more: architecture documentation for agile development. In: Proceedings of the 6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE). San Francisco, CA, USA, pp. 121–124.

Hamed, A.M.M., Abushama, H., 2013. Popular agile approaches in software development: review and analysis. In: Proceedings of the International Conference on Computing, Electrical and Electronics Engineering (ICCEEE). Khartoum, Sudan, pp. 160–166.

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

Hossain, E., Babar, M.A., Verner, J., 2009. How can agile practices minimize global software development co-ordination risks? In: Proceedings of the 16th European Conference on Software Process Improvement (EuroSPI). Alcala, Spain, pp. 81–92.

IEEE, 2010. IEEE Std 1517-2010, IEEE Standard for Information Technology – System and Software Life Cycle Processes – Reuse Processes.

ISO, 2006. ISO/IEC/IEEE Std 14764-2006, Software Engineering – Software Life Cycle Processes – Maintenance.

ISO, 2011. ISO/IEC/IEEE Std 42010-2011, Systems and software engineering – Architecture description.

Jalali, S., Wohlin, C., 2012. Global software engineering and agile practices: a systematic review. J. Softw.: Evol. Process 24 (6), 643–659.

Jensen, R.N., Moller, T., Sonder, P., Tjornehoj, G., 2006. Architecture and design in extreme programming; introducing "developer stories". In: Proceedings of the 7th International Conference on Agile Processes in Software Engineering and Extreme Programming (XP). Oulu, Finland, pp. 133–142.

Josey, A., Harrison, R., Homan, P., Rouse, M.F., van Sante, T., Turner, M., Merwe, P.vander, 2011. TOGAF Version 9.1 – A Pocket Guide. Van Haren Publishing.

Kazman, R., Klein, M., Barbacci, M., Longstaff, T., Lipson, H., Carriere, J., 1998. The architecture tradeoff analysis method. In: Proceedings of the 4th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS). Monterey, CA, USA, pp. 68–78.

B. Kitchenham, S. Charters, 2007. Guidelines for Performing Systematic Literature Reviews in Software Engineering, Version 2.3. EBSE Technical Report EBSE-2007-01, Keele University and Durham University.

Kruchten, P., 1995. The 4 + 1 view model of architecture. IEEE Softw. 12 (6), 42–50.

Kruchten, P., 2009. Documentation of software architecture from a knowledge management perspective – design representation. Software Architecture Knowledge Management. Springer, Ber lin H eidelberg, pp. 39–57.

Kruchten, P., Nord, R.L., Ozkaya, I., 2012. Technical debt: from metaphor to theory and practice. IEEE Softw. 29 (6), 18–21.

Larman, C., 2003. Agile and Iterative Development: A Manager's Guide. Pearson Education.

Li, Z., Liang, P., Avgeriou, P., 2013. Application of knowledge-based approaches in software architecture: a systematic mapping study. Inf. Softw. Technol. 55 (5), 777–794.

Malavolta, I., Lago, P., Muccini, H., Pelliccione, P., Tang, A., 2012. What industry needs from architectural languages: a survey. IEEE Trans. Softw. Eng. 39 (6), 869–891.

Manteuffel, C., Tofan, D., Koziolek, H., Goldschmidt, T., Avgeriou, P., 2014. Industrial implementation of a documentation framework for architectural decisions. In: Proceedings of the 11th Working IEEE/IFIP Conference on Software Architecture (WICSA). Sydney, Australia, pp. 225–234.

Medvidovic, N., Jakobac, V., 2006. Using software evolution to focus architectural recovery. Autom. Softw. Eng. 13 (2), 225–256.

Mens, T., Demeyer, S., Mens, T., 2008. Introduction and roadmap: history and challenges of software evolution. Software Evolution. Springer, Be rlin Hei delberg, pp. 1–11.

Nord, R.L., Tomayko, J.E., 2006. Software architecture-centric methods and agile development. IEEE Softw. 23 (2), 47–53.

Nuseibeh, B., 2001. Weaving together requirements and architecture. IEEE Comput. 34 (3), 115–119.

Petersen, K., Feldt, R., Mujtaba, S., Mattsson, M., 2008. Systematic mapping studies in software engineering. In: Proceedings of 12th International Conference on Evaluation and Assessment in Software Engineering (EASE). Bari, Italy, pp. 68–77.

Petersen, K., Wohlin, C., 2010. The effect of moving from a plan-driven to an incremental software development approach with agile practices: an industrial case study. Empir. Softw. Eng. 15 (6), 654–693.

Pikkarainen, M., Haikara, J., Salo, O., Abrahamsson, P., Still, J., 2008. The impact of agile practices on communication in software development. Empir. Softw. Eng. 13 (3), 303–337.
Postma, A., America, P., Wijnstra, J.G., 2004. Component replacement in a long-living architecture: the 3RDBA approach. In: Proceedings of the 4th Working IEEE/IFIP Conference on Software Architecture (WICSA). Oslo, Norway, pp. 89–98.
Prause, C.R., Durdik, Z., 2012. Architectural design and documentation: waste in agile development? In: Proceedings of the International Conference on Software and System Process (ICSSP). Zurich, Switzerland, pp. 130–134.
Ramil, J., Lehman, M., 2000. Metrics of software evolution as effort predictors - a case study. In: Proceedings of the 16th International Conference on Software Maintenance (ICSM). San Jose, CA, USA, pp. 163–172.
Rodríguez, P., Markkula, J., Oivo, M., Turula, K., 2012. Survey on agile and lean usage in finnish software industry. In: Proceedings of the 6th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM). Lund, Sweden, pp. 139–148.
Rost, D., Naab, M., Lima, C., Chavez, C.F.G., 2013. Software architecture documentation for developers: a survey. In: Proceedings of the 7th European Conference (ECSA). Montpellier, France, pp. 72–88.
Schwaber, K., 1995. Scrum development process. In: Proceedings of the 10th Annual Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA) Workshop on Business Object Design and Implementation. Austin, Texas, USA, pp. 117–134.
Schwaber, K., Beedle, M., 2002. Agile Software Development with Scrum. Prentice Hall PTR Upper Saddle River.
Sfetsos, P., Stamelos, I., 2010. Empirical studies on quality in agile practices: a systematic literature review. In: Proceedings of the 7th International Conference on the Quality of Information and Communications Technology (QUATIC). Porto, Portugal, pp. 44–53.
Shahin, M., Liang, P., Babar, M.A, 2014. A systematic review of software architecture visualization techniques. J. Syst. Softw. 94 (8), 161—185.
Sharifloo, A.A., Saffarian, A.S., Shams, F., 2008. Embedding architectural practices into extreme programming. In: Proceedings of the 19th Australian Conference on Software Engineering (ASWEC). Perth, WA, Australia, pp. 310–319.
Silva, F.S., Soares, F.S.F., Peres, A.L., de Azevedo, I.M., Vasconcelos, A.P.L., Kamei, F.K., de Lemos Meira, S.R., 2014. Using CMMI together with agile software development: a systematic review. Inf. Softw. Technol. 58 (2), 20–43.
Stapleton, J., 1998. Dynamic Systems Development Method: The Method in Practice. Addison-Wesley: Reading.
Stavru, S., 2014. A critical examination of recent industrial surveys on agile method usage. J. Syst. Softw. 94 (8), 87–97.
Strauss, A., Corbin, J., 1998. Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory, 2nd edition Sage Publications.
Strauss, A.L., 1987. Qualitative Analysis for Social Scientists. Cambridge University Press.
Su, M.T., 2010. Capturing exploration to improve software architecture documentation. In: Proceedings of the 4th European Conference on SoftwareArchitecture (ECSA). Copenhagen, Denmark, pp. 17–21.
Tang, A., Avgeriou, P., Jansen, A., Capilla, R., Babar, M.A., 2010. A comparative study of architecture knowledge management tools. J. Syst. Softw. 83 (3), 352–370.
Thapparambil, P., 2005. Agile architecture: pattern or oxymoron? Agile Times 6 (1), 43–48.
van Heesch, U., Eloranta, V.P., Avgeriou, P., Koskimies, K., Harrison, N., 2014. DCAR – Decision-centric architecture reviews. IEEE Softw. 31 (1), 69–76.
VersionOne, 2015. The 9th annual state of agile survey. http://info.versionone.com/state-of-agile-development-survey-ninth.html, (accessed 26.03.15).

Chen Yang is a PhD candidate in the State Key Lab of Software Engineering (SKLSE), School of Computer, Wuhan University, China and the Software Engineering and Architecture (SEARCH) research group at the University of Groningen, the Netherlands. He is working in the research field of architecture and agility. Before becoming a PhD student, he had worked as a senior software engineer and architect in industry for four years.

Peng Liang is a professor of software engineering in the State Key Lab of Software Engineering (SKLSE), School of Computer, Wuhan University, China. His research interests concern the areas of software architecture and requirements engineering. He has published more than 70 articles in peer-reviewed international journals, conference and workshop proceedings, and books.

Paris Avgeriou is a professor of software engineering in the Department of Mathematics and Computing Science, University of Groningen, the Netherlands where he has led the software engineering research group since 2006. He has co-organized several international conferences and workshops and sits on the editorial board of Springer TPLOP. He has edited special issues in IEEE Software, Elsevier JSS and Springer TPLOP. He has published more than 120 peer-reviewed articles in international journals, conference proceedings and books. His research interests lie in the area of software architecture, with strong emphasis on architecture modeling, knowledge, evolution and patterns.