IET Software

Special Issue: Advances in Knowledge and Information Software Management

IET Journals
The Institution of Engineering and Technology

Review of approaches to manage architectural knowledge in Agile Global Software Development

ISSN 1751-8806
Received on 9th September 2016
Revised 6th February 2017
Accepted on 21st March 2017
E-First on 25th May 2017
doi: 10.1049/iet-sen.2016.0197
www.ietdl.org

Gilberto Borrego[1] ✉, Alberto L. Morán[1], Ramón René Palacio Cinco[2], Oscar Mario Rodríguez-Elias[3], Eloísa García-Canseco[1]

[1]*Facultad de Ciencias, Universidad Autónoma de Baja California, Carretera Tijuana-Ensenada 3917, Ensenada, Mexico*
[2]*Unidad Navojoa, Instituto Tecnológico de Sonora, Ramón Corona S/N, Navojoa, Mexico*
[3]*División de Estudios de Posgrado e Investigación, Instituto Tecnológico de Hermosillo, Ave. Tecnológico S/N, Hermosillo, Mexico*
✉ *E-mail: gilberto.borrego@uabc.edu.mx*

Abstract: Nowadays, Agile and Global Software Development (AGSD) has brought benefits and new challenges to the software industry. Among the main challenges is Architecture Knowledge Management (AKM), due to the following reasons: (i) in Agile Software Development team members prefer to convey knowledge in a face-to-face manner, over transmitting it through formal documents; and (ii) an efficient AKM in Global Software Development involves managing explicit knowledge. These opposite paradigms turn AKM into an unsolved issue in AGSD. In this study, the authors present a systematic mapping review about AKM in AGSD. From this review, they identified nine approaches that AGSD companies use to overcome the AKM challenge, which are grouped in three areas: (i) documentation artefact-based, (ii) communication-based, and (iii) methodological-based. Also, they found that the selected papers evenly support the three phases of the integrated knowledge management cycle (creation/capture, sharing/dissemination and acquisition/application), although only 7% of them support the capture of architectural knowledge in a formalised way. Finally, they conclude proposing critical points to consider in the implementation of AKM solutions in AGSD, and presenting their directions of future work.

1 Introduction

The globalisation phenomenon has reached many industry sectors, including software development, thus, there is an increasing number of software engineers that are expected to work in Global Software Development (GSD) [1]; even GSD is increasingly becoming the normal practice in the software industry [2]. GSD attempts to have worldwide top quality professionals, reduce recruitment costs, produce software for remote clients without moving the development team, and increase innovation through cultural diversity, among others benefits [3]. At the beginning of GSD adoption, plan driven software development (PDSD) was the preferred methodological approach to perform the work [4]. PDSD is characterised by intense analysis and design phases before coding and testing software [5], which mainly affects software time delivery. Moreover, it is frequent that customers should wait a considerable amount of time to see working software, and the final product could not satisfy all the users' needs, since these needs could have changed during the development [6].

For these reasons, in the last decades Agile Software Development (ASD) has been adopted by GSD companies. ASD is an iterative approach to software development, with regular deliveries of working software increments from project inception to completion, ending with a complete product [7]. This has originated a new concept: Agile GSD (AGSD) [8].

There is an intrinsic antagonism between GSD and ASD in AGSD that is fostering research in this area [9]. Both paradigms have different statements that we briefly describe next. In GSD the teams are geographically distributed, and in ASD they are co-located, so usually developers' culture and language are the same. Also, in GSD most of the knowledge is explicit (with formal documents and strict process tracking) to reduce the impact of the four distances of this approach (namely temporal, physical, cultural and linguistic); while in ASD there are few documents, which are generally *ad-hoc* or informal, and face-to-face interactions are preferred to share knowledge [3, 7]. Most of the knowledge in ASD is tacit. In terms of knowledge management (KM) strategies [10], ASD tends to personalisation, and GSD tends to codification.

This antagonism in AGSD turns every aspect of KM into a challenge [11, 12], and hence so is architectural KM (AKM). To define AKM first we define the concepts of software architecture (SA) and architectural knowledge (AK). SA is 'the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them' [13]. These software elements and their relationships are products of a design phase, which consists of two sub-phases: architectural global design (high-level design), and detailed design (low-level design) [14]. In practical terms, we can say that SA comprise from a very abstract software structure, up to a detailed technique or pattern applied at code level. Regarding the AK concept, it is all the knowledge that the SA implies (structures, software elements, relationships either high or low level), plus the decisions and rationale that led to the current software organisation [15]. Then, AKM implies creating/coding, sharing/disseminating and acquiring/applying AK among the different team members, regardless of their geographic location [16].

We should recognise that a software development team must have access to AK, since it is the base to build and understand the technical part of any software development cycle. In AGSD, the predominance of tacit knowledge and the lack of explicit documentation affect AKM, since there are fewer opportunities to share knowledge face-to-face. It derives in problems such as loss of time answering the same questions repeatedly by experts on certain issues, waste of time finding solutions to problems presented before, lack of knowledge transfer between different teams, loss of knowledge when a project member retires, defects in software evolution and maintenance, lack of visibility for project monitoring and technical solutions and poorly understood requirements, only to name some significant ones [17, 18]. These problems may seem trivial, but it has been proven that a good KM is an important factor for any software development project; KM absence is a failure factor [19]. However, Agile developers consider the additional time and effort required by AKM activities as an overhead, due to the following values of the Agile Manifesto [20]: 'Working software over comprehensive documentation', and

'Working software is the primary measure of progress'. The previous arguments support the need for research about AKM in AGSD.

In this paper, we present a literature mapping review about AKM in AGSD. We should remark that the development process of any software company is considered an asset, then all the studies that we analysed are subject to it. The remaining of this paper is organised as follows: in Section 2, we present different reviews related to KM, AKM, GSD, and AGSD; in Section 3, we describe the methodology used to perform the mapping review; in Section 4, we present the results of this review; in Section 5, we discuss these results; in Section 6, we discuss on the validity of this study; and finally in Section 7, our conclusions and future work are presented.

2 Related work

To the best of our knowledge, nowadays there is no literature review or survey that tackles the topic of AKM in AGSD. Here we present five papers (reviews and surveys) related to this topic: two about KM in software engineering, two about AKM in GSD, and another one about KM in ASD.

2.1 KM in software engineering

Dingsøyr and Conradi [21] presented a survey where they found that software development companies store experience in some way (codification strategy), and most of them facilitate knowledge flow in the organisation (personalisation strategy). Also, most of them emphasise that developers should actively participate in collecting and distributing knowledge, like experiences descriptions, i.e. qualitative knowledge. The authors concluded that there is a great interest in developing Intranet-based technology to support KM in software engineering, but there is a lack of understanding about how experience sharing works.

In the same vein, Bjørnson and Dingsøyr [22] conducted a systematic literature review to identify the major KM concepts that have been investigated in software engineering, in terms of Earl's schools of KM [23]: Technocratic – information systems to manage knowledge, economic/commercial – how knowledge assets relates to organisations' incomes, and Behavioural – organisational and spatial issues to promote knowledge sharing. They found that, in software engineering, knowledge is managed mostly through repositories that support knowledge flows. They also report that in Agile methods, cartographic schools (knowing who knows what), and part of technocratic schools, have an important place given the predominance of tacit knowledge in the Agile paradigm.

It is interesting to see that the survey of Dingsøyr and Conradi, published in 2002, points out that companies showed great interest in developing technology to support KM, and the review of Bjørnson and Dingsøyr, published in 2008, showed that KM technology was prominent in software development companies. This shows that KM has become a need for the software engineering community.

2.2 AKM in global software engineering

Ali *et al.* [24] conducted a literature review to find AKM concepts, practices, tools, and challenges in GSD. They found that GSD companies require coordination strategies to disseminate design decisions among distributed teams, particularly when these teams develop interfaces to communicate software components. Also, they found that there is no agreement about coordination strategies for AKM. The authors proposed strategies to manage AK, such as including lead architects to improve communication among architecture stakeholders; and face-to-face meetings to discuss AK issues. Unresolved challenges were also presented, such as the need of tools to manage AK in GSD, and the need of communicating design decisions across teams geographically dispersed and culturally different.

In another review, Beecham *et al.* [25] suggested AK practices in three main areas: (i) alignment of architecture and organisation, (ii) KM practices for creating and disseminating AK, and (iii) infrastructure for managing AK. Concerning architecture-organisation, they suggested to have a team of architects to facilitate the communication among the locations of the GSD company. Regarding KM practices for creating and disseminating AK, they identified seven practices focused on: meetings to create/disseminate AK, define clear lines of communication or a group to ask SA questions, and the creation of repository artefacts with decisions, rationale, and architectural designs. Finally, an infrastructure for managing AK, simply refers to have only one place to store the AK generated per software project.

In these two reviews, coordination strategies were considered an important aspect to achieve AKM, as well as, the architect role and the face-to-face meetings to achieve agreements at the architectural level.

2.3 KM in ASD

Yanzer Cabral *et al.* [26] conducted a systematic review to identify the most important findings about KM in ASD. They found problems in projects documentation, where the lack of documentation among developers is highlighted, since they prefer face-to-face communication; it leads developers to get AK directly from the code, which does not warrant knowledge acquisition. Another finding was the existence of techniques and frameworks to manage tacit knowledge; however, only a few of them have been tested in-situ. They also identified a set of KM tools, such as wikis, semantic wikis and different groupware tools. Most of these KM tools are for developers; however, only a few were tested in real settings. Further, the adoption of KM methodologies and KM sharing practices (e.g. communities of practice) was found as a common way to manage knowledge in ASD.

Yang *et al.* [27] presented a systematic mapping review about using SA practices/activities in ASD. Although this review is not directly related to KM topics, they found that architectural description (i.e. AK codification) is the most frequently discussed activity when ASD and SA are combined. Particularly, the researchers are focused on developing lightweight notations to express AK in ASD. Further, they found that most of the reported challenges are also related to architecture description and the tacit way to manage AK in ASD. Finally, they coincide with Yanzer Cabral *et al.* [26] on the used tools to support architectural activities in ASD (e.g. Wiki); however, Yang *et al.* argue that those tools are borrowed from other fields of software development, thus none of them are dedicated to support an SA in ASD.

The findings, approaches and solutions proposals of the papers presented above are important to achieve either KM or AKM in GSD, ASD or software engineering in general. However, the intrinsic antagonism within in AGSD (presented in Section 1) forces either to re-evaluate the applicability of those findings, or to conduct new literature reviews to identify how AGSD companies manage AK.

3 Methodology

The present mapping review conforms to the guidelines proposed by Petersen *et al.* [28]. Particularly, we used the PICO strategy (specified in Section 3.2) to build the search string, and the test-retest technique to search and select papers to decrease bias during the review process. We describe the steps of this process below.

3.1 Definition of research questions

We defined two research questions: (**RQ1**) How do AGSD companies manage AK in their projects?, and considering the integrated KM cycle phases [29]: (a) capture/creation, (b) sharing/dissemination, and (c) acquisition/application, and (**RQ2**) Which KM phases are prominent in the AKM of AGSD companies?

3.2 Conduct search for primary studies

This step consists of two parts: (i) build a search string, and (ii) databases definition. We defined a search string that conforms to the guidelines proposed by Kitchenham and Charters [30]. These guidelines suggest the PICO criteria to build a search string. PICO stands for: Population (targeted group), Intervention (e.g. methodology, tool, technology, procedure), Comparison

Table 1 Search string building

Part	String
population	'GSD' OR 'distributed software development' OR 'global software engineering' OR 'distributed development' OR 'virtual team' OR 'virtual teams'
intervention	Agile AND ('AK' OR documentation OR 'software design' OR 'architecture design')
whole search string	Agile AND ('GSD' OR 'distributed software development' OR 'global software engineering' OR 'distributed development' OR 'virtual team' OR 'virtual teams') AND ('AK' OR documentation OR 'software design' OR 'architecture design')

(comparison between different interventions) and Outcome (e.g. improved reliability, reduced production costs, reduced time to market). These are the parts that must be included in a search string for a systematic literature review. According to Petersen *et al.* [28], outcome and comparison could be omitted for a systematic mapping review, since it tries to cover broadly the literature. Thus, we define the population as GSD companies, and the intervention as the Agile methods and the AK. As the final step, we searched iteratively the common way to refer to each concept (e.g. GSD, AK) in Google Scholar, to identify synonyms and alternative spellings for each of the question elements. We linked the synonyms and alternative spellings using OR as a Boolean operator to obtain the search string as shown in Table 1.

Regarding database definition, we selected relevant databases for the research in software engineering: IEEE, Springer, Emerald, Science Direct, ACM, and Wiley. It is worth mentioning that we executed the search process at different moments with the same search string, and on the same databases: the first one to obtain an initial set of papers, and the second one to update the set a few months later.

3.3 Screening of papers for inclusion and exclusion

We defined the inclusion and exclusion criteria to select the relevant literature for the research questions. The inclusion criteria were defined as follows:

(a) Primary studies published on journals, conferences/workshops proceedings and books, which passed through a peer review process.
(b) Studies (empirical or theoretical) which include report/proposal of any kind of: technical/architectural documentation or technical/architectural sharing or publishing.
(c) The main topic of the publications could be other than AK, i.e. it is not required that the main topic of the papers should be AK.

With these inclusion criteria we want to obtain evidence of any approach to document/share/publish AK reported on any work related to AGSD, which has been peer-reviewed. It is worth describing how we applied the second inclusion criteria. We first read the title and abstract of the paper; if the criteria were not fulfilled, we conducted a rapid reading of paper's introduction and conclusion sections; if the criteria were not fulfilled yet, we conducted a rapid reading of the full paper; and finally, if the criteria were not fulfilled yet, we discarded the paper. Now, the defined exclusion criteria were the following:

(a) Literature that was only available in the form of abstracts or slides presentations.
(b) Documents that were not peer reviewed (grey literature).

We did not include grey literature because papers of this kind are prone to be biased by the business objectives of the companies that published the reports. Also, we applied the criteria at three different moments (test-retest) to decrease bias and to have a better selected set of papers: one on the first set of papers, and two on the second set of papers.

Table 2 Distribution of papers per database

Database	Papers count		
	Initial search	Chosen[a]	Percentage[a], %
ACM	40	6	14
IEEE	602	14	29
Emerald	67	0	0
Science Direct	241	9	21
Springer	455	15	36
Wiley	116	0	0
Total	1521	42	100

[a]After inclusion and exclusion criteria.

3.4 Keywording

The objective of this part is to build a classification scheme based on the obtained results of the literature search (once the application of the inclusion and exclusion criteria). The process that we applied to define the classification scheme was the following:

(a) Rapid reading of each paper.
(b) Build a conceptual map of each paper.
(c) Identification of similar concepts on all the conceptual maps.
(d) Name the groups of similar concepts.

3.5 Data extraction and mapping of studies

In this last step, the selected papers were sorted into the classification scheme. Also, we defined which phase(s) of the integrated KM cycle support(s) each paper. We conduct these two classifications in a spreadsheet; and then we calculated the frequencies of publications in each category aiming at answering the research questions. Further, we obtained some general data such papers per database, per year and per publication name. Finally, we fully read all the selected papers in order to write a proper description of each group of the classification scheme.

4 Results

In this section, we present the results of the systematic mapping organised in the following subsections: (i) overall findings, (ii) approaches to manage AK in AGSD, and (iii) KM phases supported by the AKM approaches in AGSD.

4.1 Overall findings

After querying the selected databases using the defined search string, we obtained 1521 papers in total. These papers were filtrated using the defined inclusion and exclusion criteria. We selected 42 papers, distributed as shown in Table 2.

Then, we obtained the classification scheme that was represented as an ontology. The ontology depicts the approaches used to overcome AKM in AGSD (see Fig. 1). We identified three main areas: methodological-based approaches, documentation artefact-based approaches and communication-based approaches. Methodological-based approaches refer to the inclusion of phases or roles to address architectural issues. Documentation artefact-based approaches refer to software development support (SwDev support) (e.g. IDE, Wikis, repositories etc.) to register architectural design or architectural decisions; also it refers to documentation generated by an automatic process or based on a lightweight notation. Finally, communication-based approaches refer to share or publish AK, particularly: videoconferencing, electronic textual media (e.g. email, instant messaging, forums etc.), and information radiators. With this classification scheme we start to answer the first research question of this review (**RQ1**); we present more details about these approaches in the next subsection.

As the next step, we related the papers to this classification scheme. We only assigned papers to leaf entities in the aim to be clearer on the classification (see Table 3). We obtained that methodological-based approaches include 20% of the papers, documentation artefact-based approaches include 45% and

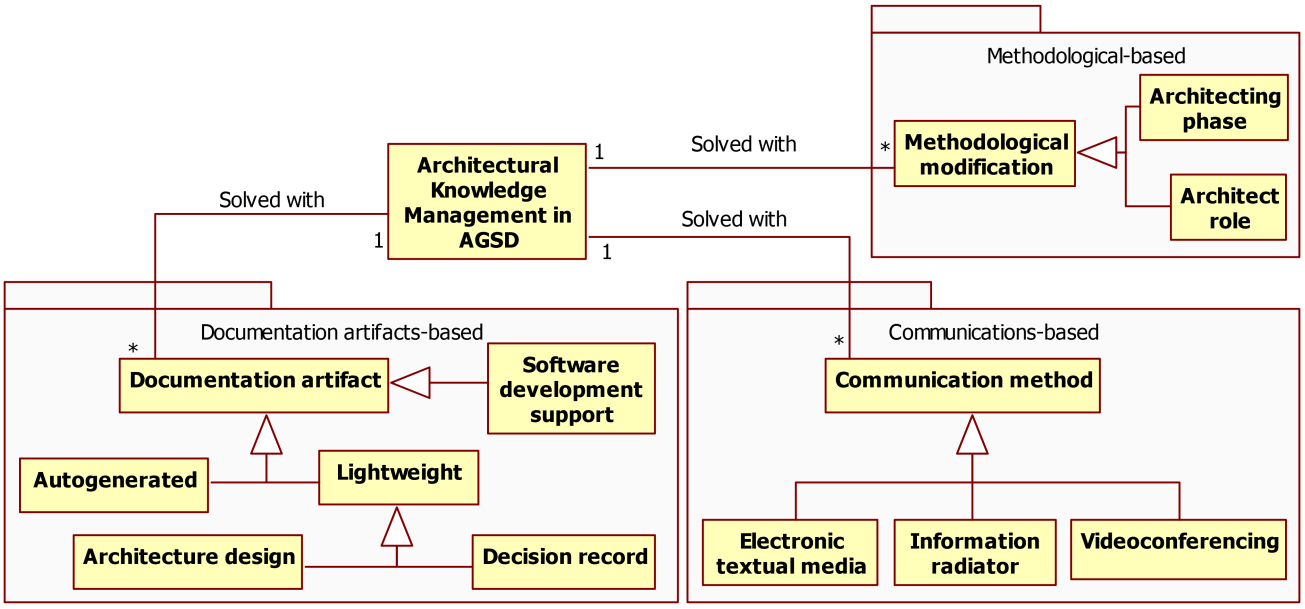

Fig. 1 *Ontological representation of the approaches used to overcome AKM challenges in AGSD*

Table 3 Distribution of papers per ontology entity

Ontology entities		Related papers	Papers count
Documentation Artefacts	software development support	[31–43]	13
	auto-generated	[44–47]	4
	Lightweight architecture design	[48, 49]	2
	decision record	[36, 50, 51]	3
Communication methods	videoconferencing	[17, 39, 52–58]	10
	electronic textual media	[33, 39, 52, 53, 59]	5
	information radiator	[52, 55, 60]	3
Methodological modifications	architect role	[33, 37–39, 61–66]	10
	architecting phase	[50, 62, 67–71]	7

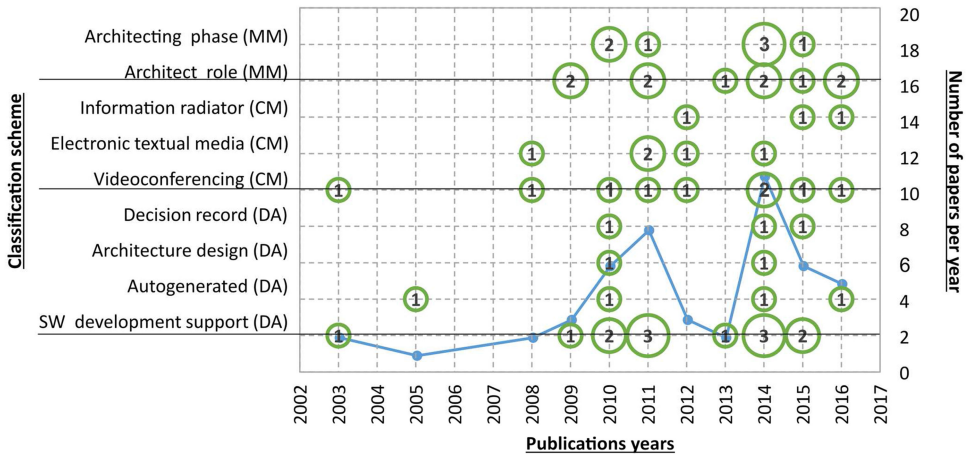

Fig. 2 *Distribution of papers per year and per approach. The line graph represents the publication behaviour throughout the years (MM = methodological modification, CM = communication methods, DA = documentation artefacts)*

communication-based approaches include 35%. The papers classification was refined at the endings of 2015 and 2016, and in the middle of 2016.

We observed that the first papers reporting ways to manage AK in AGSD were published in 2003 (see Fig. 2). This is interesting because the Agile manifesto [20] was published in 2001, so these papers can be considered the early adopters of AKM in AGSD. Then, since 2008 the number of publications started to increase; coincidentally, the first International Conference of Global Software Engineering (ICGSE) was held on 2006, so it seems that the GSD community paid more attention to AKM aspects in Agile methods, since the first ICGSE.

In Fig. 2, we also noted that the documentation approach is almost constantly present. This may be influenced by the traditional way to manage knowledge (through documents). Communications methods have an important presence, since ASD heavily relies on personalisation strategies. However, in AGSD these strategies are conducted over video, electronic textual media and information radiators. Finally, methodological modification approaches are constantly present since 2009, but never before. This behaviour could be explained as follows: ASD practitioners state that SA emerges from development process, since the Agile manifesto states that '*the best architectures, requirements, and designs emerge from self-organising teams*' [20]. This mindset could had been blocking the inclusion of architectural phases or

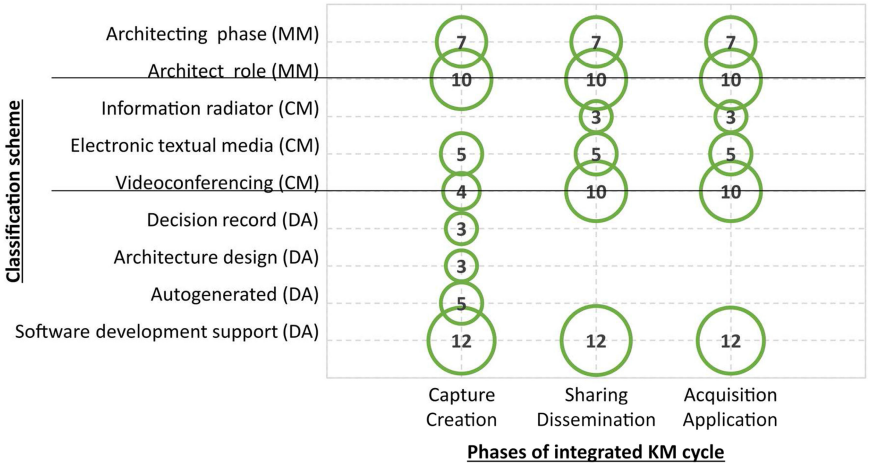

Fig. 3 *Distribution of approaches of papers per KM phase and per category (MM = methodological modification, CM = communication methods, DA = documentation artefacts)*

roles in the Agile methods, until the size of systems, and the problems related to the GSD's four distances forced to open the mind about methodological modifications.

Concerning the second research question (**RQ2**), we related each approach or each selected paper to a KM phase of the integrated KM cycle. In Table 2, it can be seen that a single paper could be related to more than one approach, thus the numbers in Fig. 3 do not match the number of selected papers.

At first sight, we observed that the number of approaches supporting each KM phase is evenly distributed: capture/creation – 35%, sharing/dissemination – 33%, and acquisition/application – 32%. It would seem that AKM support in AGSD is complete; however, there are hidden details that we will clarify once we describe all the identified approaches.

We also observed that in documentation artefacts predominates capture/creation support (see Fig. 3), since most of its approaches are only focused on registering AK. In the case of SwDev support, sharing/dissemination and acquisition/application are present since tools such as wikis, repositories and groupware tools also support those KM phases (the same case as for electronic textual media). The videoconferencing approach has four papers supporting Capture/Creation phase, since these papers specify that architectural designs were created on video calls. Finally, methodological modification approaches support all the phases, since an architect role and an architecting phase have the objective of creating, capturing, sharing and applying AK.

4.2 Approaches to manage AK in AGSD

As we stated previously, the approaches to overcome AKM in AGSD are distributed in three main areas: methodological-based approaches, documentation artefact-based approaches and communication-based approaches. In this subsection, we describe all the approaches in order to fully answer the first research question (**RQ1**).

4.2.1 Documentation artefact-based: Lightweight and auto-generated artefacts, as well as, SwDev support are part of this main area. Concerning the former, we found that repositories, wikis and groupware tools are prominent for AKM.

Repositories are mainly used to store and share source code, software libraries, and project documentation [61]. However, repositories are not frequently used to store AK in practice, mainly because they need a great maintenance effort [72]. In AGSD, developers rarely use a single repository to share AK. They rather send emails from site A to site B to ask for information, instead of sharing it in a repository [33]. This preference causes a wide distribution of AK among different repositories, many representations of the same element, and inconsistent definitions [34]. Further, AGSD developers think that repositories are not the best solution for sharing AK. They would prefer tools for navigating or tracing software artefacts, and tools for the interactive search of AK [35]. Moreover, the integration of knowledge repositories with modelling and developing tools is an emerging trend, to bring an AK closer to the developers [36].

Repositories have another way to store AK, when a developer commits a new piece of code, s/he explains in a comment the objective of the new code (it is either an error fix, an improvement, a new functionality etc.) [31]. All these comments could be viewed as a source of design decisions, since each developer exposes the reasons of the new code in the project. Repositories can be mined to extract design decisions based on machine learning and natural language processing, and then reuse this AK in different projects [32].

Wikis represent another way to manage AK. A wiki is a collaborative and informative tool used in software development to share knowledge [73]. There are ASD companies with policies that state that AK artefacts must be published in wikis [37–39, 53]. However, since these artefacts are unplanned and *ad hoc*, developers struggle to find AK [37], since these artefacts loose part of the meaning once they are consulted out of the original context [74]. Further, AGSD developers report issues with knowledge retrieval, such as: an ineffective structure for knowledge querying [32], and an inefficient mechanism to operate a wiki [31]. Thus, wikis are not preferred by AGSD developers to manage AK. They prefer simpler ways, such as web pages, electronic documents, and even natural language [35]. Nowadays, wikis are integrated in repository portals, e.g. in: GitHub, GoogleCode, and BitBucket [40]. However, frequently there is an insufficient AK in wikis of open source projects [31]. A similar situation occurs in AGSD projects, so people are forced to capture AK to keep the project documented [44]. Despite the reported problems, the adoption of wikis might reduce the time to find AK [31, 61]. Moreover, wikis are considered as a lightweight and a prominent solution to manage AK, but more research about its adoption is required [41].

The last SwDev support tool used to manage AK in AGSD is groupware tools. A groupware is a computer-based system that supports a group of people committed in a common task, and it provides an interface to a shared environment [75]. In AGSD, developers successfully collaborate in technical issues tracking and requirements managing using commercial groupware tools, such as JIRA and Team Foundation Server [38, 41]. However, this collaboration generally only covers the software specification. There are groupware tools only focused on sharing/representing AK, such as COACH-IT [42]. It uses the Architecture Definition Language (ADL) as a standard language to represent an architecture. Nonetheless, ADL is also the main disadvantage of COACH-IT, since this language is not broadly adopted in software development. In addition, Archinotes [43] has an informal way to represent architectures, and it is effective to collaborate among AGSD teams to define them. This tool is promising, but more industrial case studies are required to determine its reliability. There are other groupware tools focused on AKM; however, it is unclear whether they are used in AGSD. These groupware tools

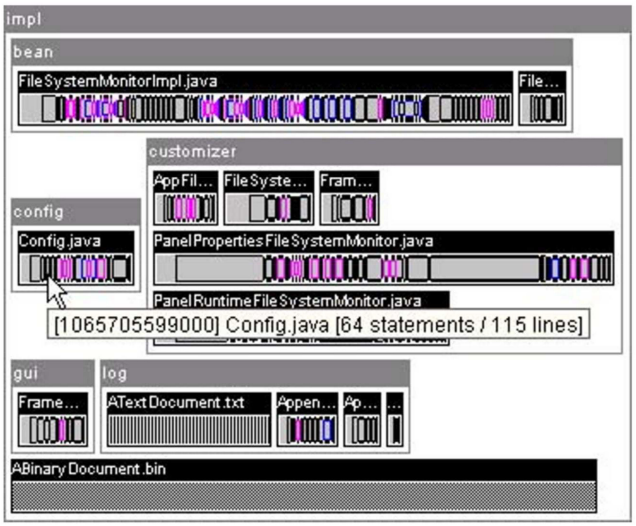

Fig. 4 *Screenshot of WRCC showing hierarchical stacked layout of a software component (taken from [47])*

have been classified in three generations: the first one focuses on knowledge representation, the second one focuses on knowledge sharing, and the last one focuses on collaborative aspects, reusing AK and decision taking [32].

Regarding auto-generated artefacts to support AKM in AGSD, we found the case of Traceman [45, 46], where a team member must establish the following relations in Enterprise Architect (a UML modeller): user stories – requirements, user stories – test cases, and architecture design – user stories. These relations are used to generate documents such as requirements specification and unit tests (in code). Traceman only works with the following assumptions: (i) the existence of requirements documents, user stories, architecture design, and test cases; (ii) the existence of the following roles: requirements engineer, software architect, and tester. Also, there are tools that generate new knowledge or a new view of knowledge, based on code repositories analysis. For example, in [36] it is proposed that repository harvesting could be useful to find and to generate knowledge. Also, the War Room Command Console (WRCC) application [47] analyses code repositories to obtain artefacts that represent the code in a graphical way (see Fig. 4). The graphical views of WRCC act as a common reference to help during decision-making, conflict resolution, and issue addressing at the code level. Also, these views are useful to understand the current architecture and the complexity of the software.

We also found auto-generation of artefacts based on communications analysis. This technique stands from the preference of AGSD developers to use media, such as phone, video, email, instant messenger [33], or even through repository commit message. In [76], email communications and code repository histories are analysed to recommend the most qualified person to help with a specified part of the code. In terms of Earl's schools of KM [23], this approach is aligned to the cartographic school (know who knows what). The preliminary results show that it needs adjustments to improve the accuracy of the recommendations.

Another source of knowledge are email messages, but since they are unformatted (free text), they must be classified or categorised. In [44], a machine learning technique is applied to automatically create relations between emails and user stories. This technique assigns a tag (representing a user story) to each message, thus when a developer finds a message, s/he knows the related user story. At the moment, this technique reports 70% of assignment accuracy, so it requires improvement for an industrial implementation.

The last approach referring to documentation is lightweight artefacts. Artefacts of this kind intend to be an alternative to the informal way to represent architectures in AGSD, but without causing loss of meaning, once they are consulted out of the original context [74] (e.g. when an architectural sketch is made in a design meeting, and a team member consults it later, usually it is not equally interpreted than during the meeting). Plastic partial component (PPC) [48] is a lightweight notation for architectural artefacts. It rests on components, aspects or variations, and features, and each has a graphical representation (see Fig. 5). An aspect is the possible variation that a component could have in its implementation. For example, in Fig. 5 the variations are: distribution type, protocol and the client system. Each aspect has different ways to implement it. For instance, in Fig. 5 the aspect named as protocol has two implementation options: IP and UPnP. With only three graphical elements an SA is registered in a lightweight manner.

One of the main concerns about text-based artefacts is their limited support for structuring and indexing content, and their small capability to describe AK unambiguously and comprehensively for all. Consequently, developers spend valuable time to find and understand AK in text-based artefacts. From this concern, in [49] an ontology-based SA documentation is presented. To implement this documentation, the development team must create an ontology to represent its AK requirements (see Fig. 6), and then they can instantiate (or 'populate') the ontology with AK of different projects, preferably using a semantic wiki. AK would be structured and easy to find. However, the authors only tested the ontology creation process in a big AGSD company. More case studies are needed to determine the success of the ontology-based documentation approach.

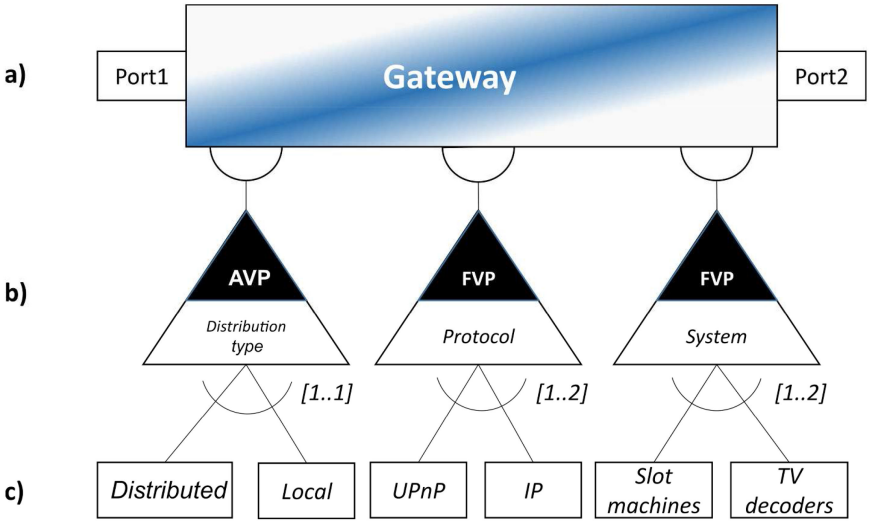

Fig. 5 *Example of a PPC (based on [48])*
(a) Component, ***(b)*** Aspects, ***(c)*** Features

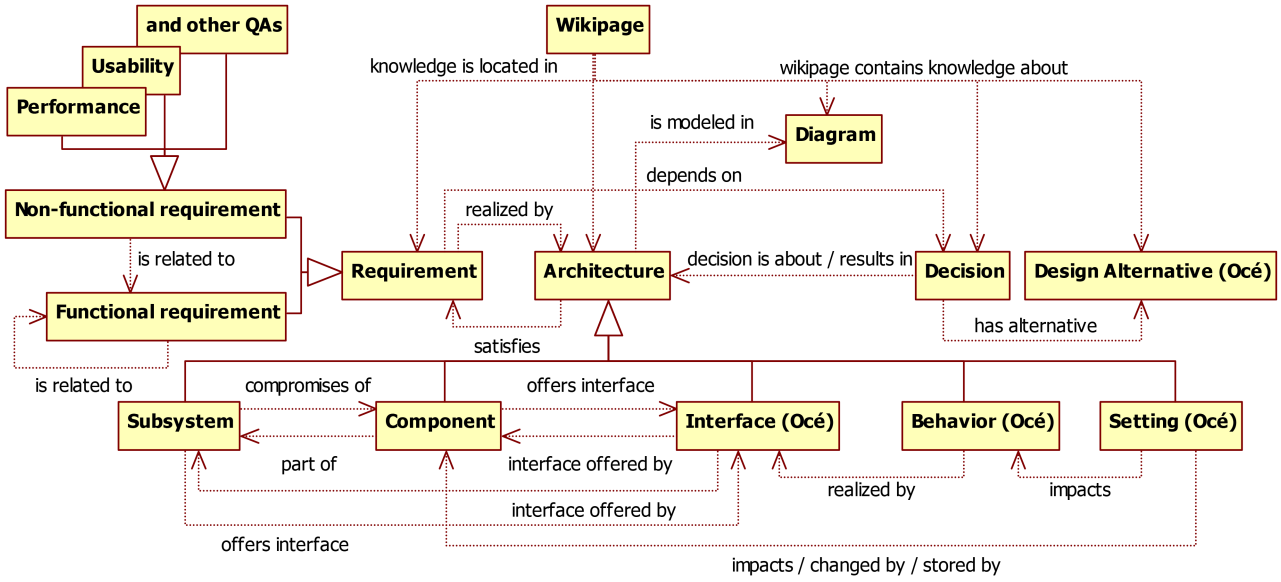

Fig. 6 *Example of ontology-based SA documentation (reproduced from [49])*

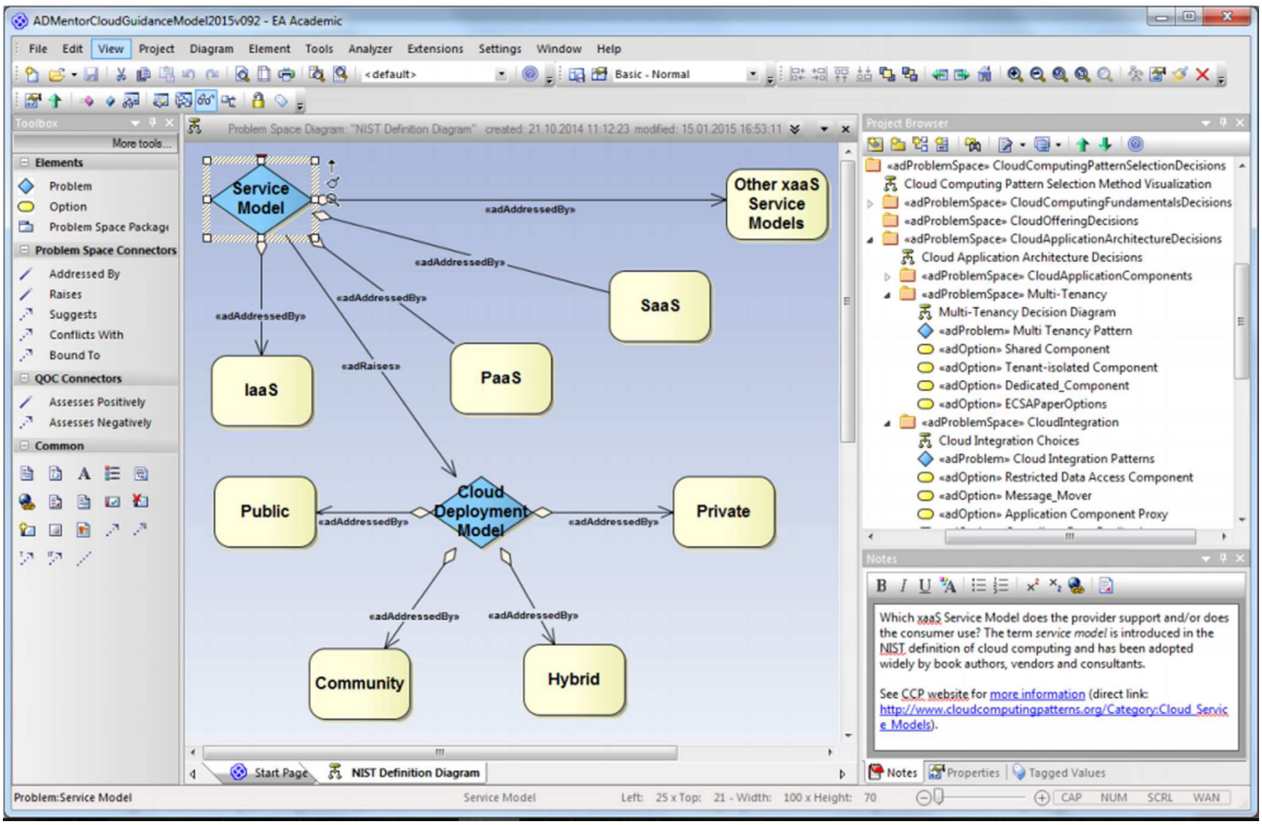

Fig. 7 *ADMentor screenshot (taken from [51])*

Regarding the documentation of design decisions, the adoption of lean models in AGSD has reduced the amount of items to capture them [32]. To ease architectural decisions registering, in [51] ADMentor is presented (see Fig. 7). ADMentor is an extension of Sparx Enterprise Architect that uses rapid problem space modelling, UML model linkage, question-option-criteria diagram support, meta-information for model tailoring, and decision backlog management. ADMentor could be used in meeting and design workshop situations (for decision modelling on the fly). This tool is projected to become a virtual mentor making formerly tacit knowledge explicit in an easy-to-consume way. However, the amount of work that ADMentor requires in practice to model decisions could be inconvenient for some AGSD projects.

The support on design decisions has also been focused on choosing one alternative. Fussy decision modelling [36] helps to discard alternatives of the design space. For every architectural concern (in a rectangle in Fig. 8), all the possible alternatives are listed (in an oval in Fig. 8), and then each alternative is marked as acceptable or unacceptable. Next, the acceptable alternatives are related to the implementation implications. When all the fuzzy models are saved, the team has a history of the design decisions of the current architecture.

Product Line Architecture Knowledge (PLAK) [50] is another way to capture design decisions. It rests on PPC and extends the concept of AK considering the architecture and design decisions in the same model (see Fig. 9). Developers have the ability to trace the decisions' story and the architecture. Each element of PPC (component, aspect and feature) has associated the following descriptions: (i) why that element was chosen, (ii) what are the cost and the risk of using that element, (iii) the description of the trade off, (iv) assumptions, and (v) constraints. To implement PLAK,

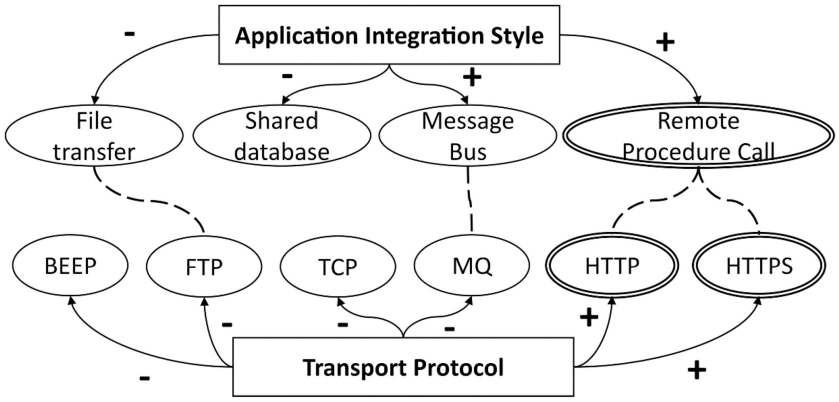

Fig. 8 *Example of fuzzy decision modelling with acceptable (+) and unacceptable (−) decisions (reproduced from [36])*

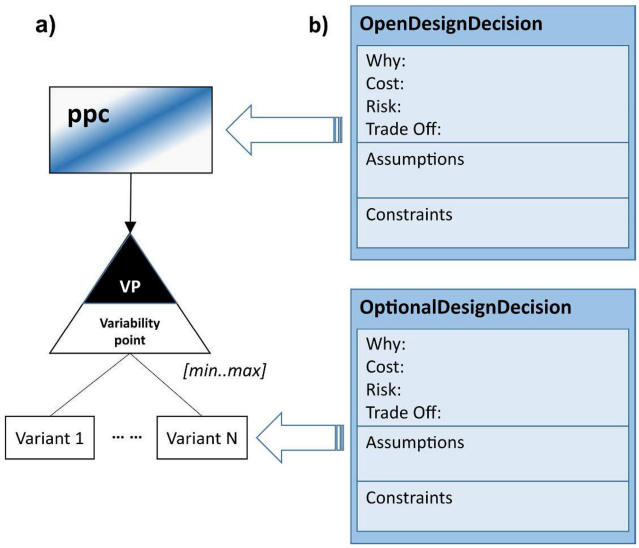

Fig. 9 *Example of PLAK (taken from [50])*
(a) PPC definition, ***(b)*** Design decisions

any Agile methodology must be added with a phase (between sprints) to refine the SA, i.e. a methodological modification is needed.

4.2.2 Communications methods-based: Face-to-face communication and collaboration between team members is the natural way to share AK in ASD; however, in AGSD, these communication methods are difficult, due to the temporal and physical distances. Electronic textual media (email, forums, and instant messaging etc.) emerges as an alternative to share AK. These media are some of the preferred by developers in software companies [33, 39, 52, 59]. Particularly, email is used to share and ask for AK, because developers can verify their writing before sending the email [33]. Also, code discussions are held through email, but it quickly becomes a laborious and time consuming task [53]. Generally, social media (email, social networks, blogs etc.) are popular in ASD (co-located or distributed), especially to clarify architectural requirements [39, 59], especially during the beginning and planning sprints (concerning Scrum-based projects). Agile developers take advantage of chats, since they keep logs of them and they use those logs to clarify issues with clients [59].

We identified videoconferencing as a popular practice to share AK in AGSD [39, 52–58]. This practice is used to hold different kinds of meetings, including kick-off, daily, design/architectural, teaching, communities of practice, and clarification meetings. Videoconferencing is the richest way to share AK (when the time zones allow it) considering communication and features (e.g. screen sharing, virtual boards etc.); however, it is not the preferred one by developers. This is because videoconferencing could highlight the cultural and language distances among developers, as this practice implies oral communication, corporal language, and physical location awareness across sites. For instance, virtual window applications were not well adopted in AGSD as reported in [55].

It is worth analysing a particular meeting based in videoconference: community of practice (CoP) or knowledge community. CoP is a 'group of people who share a concern, a set of problems, or a passion about a topic, and who deepen their knowledge and expertise in this area by interacting on an ongoing basis' [77]. CoPs have been identified as an increasing KM practice in AGSD between local and foreign teams [56–58]. In [58] four different purposes of CoPs were identified: (i) knowledge sharing and learning (depicted in [56] as 'expert teaching'), (ii) coordination, (iii) technical work (showed in [57] as 'tech calls'), and (iv) organisational development. CoPs focused on technical work, knowledge sharing and learning are the most common [72, 77], and they are prominent on AK sharing. Particularly, technical work CoPs are conducted to take design decisions between local and remote members [58]. Temporal and physical distances deplete CoPs implementation [62] when there is not a proper technological support (software and IT infrastructure), and schedules' teams are not matched. In those cases, CoP implementations delegate the team interaction to web pages and newsgroups [17] (e.g. PRIME).

Another communication method is an information radiator (concept introduced by Cockburn [78]). An information radiator is a graphical chart (physical or virtual) that developers use to be aware about the project at a glance. It must be accessible to show valuable information to team members, and it must be easy to update and easy to understand [74]. An information radiator is implemented on a smartboard or an electronic display, it is useful to support technical meetings with its foreign counterpart [52, 55, 60]. After the technical meeting the information on the radiator could be kept, so that the developers could consult the radiator to know about technical issues.

4.2.3 Methodological modification-based: Agile methods do not have a phase to define an SA. The Agile manifesto [20] states that 'Working software over comprehensive documentation' and 'The best architectures, requirements, and designs emerge from self-organising teams'. It means that coding is preferred over designing an architecture (the architecture emerges as the project progress), and the architect role does not exist, as all team members participate in architecture design. However, when development teams are geographically distributed and the projects are bigger, at least a minimum architecture definition must be used to coordinate the development efforts. We found two approaches in AGSD to include architectural aspects in Agile methodologies, including an architect role, and including an architecting phase.

Regarding the inclusion of an architect role, it could be a single person or a team of architects [33, 37–39, 61–66]. By including this role, AGSD teams have an entity that defines, refines, communicates, codifies (in architectural artefacts) and follows up the projects' SA, architectural principles, patterns, coding conventions and so on. A team of architects could be organised either hierarchically (with a chief architect and subordinated architects) [33, 62, 64, 65], or discipline [63] (e.g. application

Table 4 Reported benefits and impacts of AKM approaches in AGSD

Ontology entities	Reported impact/benefit
software development support	[43] – Users were more inclined to have daily meetings to review the overall architecture of the project supported with Archinotes.
auto-generated	[44] – The matching accuracy between email and user story of the auto-tagging mechanism was 70%. [47] – WRCC was useful to speed up the product's description to new team members, to visualise product's complexity, and to understand architectural changes' impact.
architecture design	[48] – The need of refactoring was sized down, since architecture was early designed. [49] – The ontology was helpful to reason about what AK is in SA documentation, and what AK should be in the SA documentation.
decision record	[50] – It was easy to locate decisions' impact and to follow up decisions' rationale.
videoconferencing	[58] – CoPs keep developers updated about what happens in the organisation at large in terms of AK.
architecting phase	[67] – User Stories were derived based on a logical architecture, allowing to be properly aligned within reduced time.

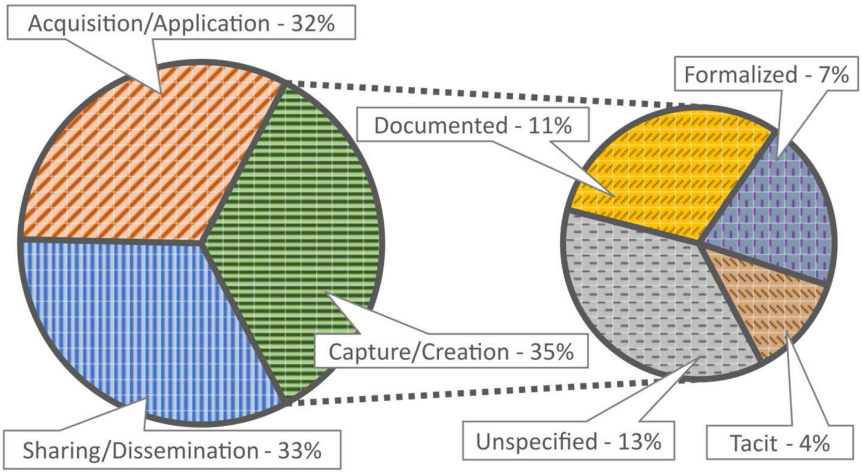

Fig. 10 *Classification of papers supporting the capture/creation phase according to Nonaka and Takeuchi [81]*

architect, security architect, data architect etc.). Architects also interact with customers, so they have more elements to prioritise user stories, and to design an overall solution [37]. In other cases, a Scrum Master (a team facilitator and a mediator in Scrum) plays the architect role or technical mentor [37].

Also, the need of architectural coordination in AGSD has led to adding an architecting phase in some AGSD projects. In [50, 62, 67–69, 79], a phase called sprint zero (also called pre-development phase [68] or domain engineering [50]) has been introduced to develop a common understanding of requirements and to propose a first architecture proposal. A similar approach is presented in [67], where a phase of analysis and design is executed based on the V + V process [80] (the first execution of the V-model relates to analysis, while the second relates to design). After executing the V + V process, a logical architecture is obtained, which could be segmented into modules, and each module is assigned to a Scrum team.

Moreover, we found hybrid methodology proposals based on Scrum and RUP [70, 71], in order to use the Agile project managing framework of Scrum and the software engineering discipline of RUP. This mix seems as a good option to manage AK in AGSD, but it could be too heavy for small teams, since it requires many activities and artefacts [70]. Both cases [70, 71] do not report a case study where the methodology had been implemented, and particularly in [71] there are no details about artefacts and activities.

Further, architectural practices have been applied/developed to improve an architecting phase in AGSD [68]. Those practices are: (i) travel and delegation, a group of developers travel to the counterpart site to tackle a design issue; (ii) inter-team communication, where a person represents the technical contact to other development teams; (iii) minimal up-front planning, equals to the sprint zero approach; (iv) effective preparation work, refers to not spending too much time in up-front planning; and (v) distribution of design and analysis work, to spread out AK.

Finally, we observed that only eight of the selected papers, report impacts or benefits about the use of a determined AKM approach. In Table 4, we show the benefits and/or impacts that the selected papers report.

4.3 KM phases supported by the AKM approaches in AGSD

As we stated in Section 1, it would seem that AKM support in AGSD is complete, due to the evenly distribution of the KM support shown in Fig. 3; however, there are hidden details particularly in the capture/creation phase. We have to realise that creating knowledge does not imply to capture it. If we observe Fig. 3, almost all the approaches support capture/creation phase; however, there are cases where AK is only created, and others where AK is captured but it is not specified how the AK is codified. Thus, we analysed each case of capture/creation support and we classified them according to the three states of knowledge proposed by Nonaka and Takeuchi [81]: tacit – knowledge is in the stakeholders' mind; documented – knowledge is codified in an informal/*ad-hoc* manner; and formalised – knowledge is codified in a formal structure.

We observed that only 7% of the papers report a formalised way of coding AK (see Fig. 10). These papers correspond to the following approaches: decision record, architecture design, auto-generated and SwDev support (particularly groupware tools: COACH-IT [42] and Archinotes [43]). Further, another 11% of the papers report a volatile way to keep AK, i.e. tacit or documented. We state that documented AK is volatile, since it can lose meaning over the time or in another context (AK is vaporised). The remaining 13% of papers do not specify how AK is coded.

With this evidence, we cannot ensure that the capture/creation phase of the KM cycle is fully supported by the AKM approaches used in AGSD. Even, we neither can ensure that the rest of the KM phases are fully supported, due to the volatility of the AK that is shared or acquired by AGSD developers. Finally, to close this section, we can state that AKM in AGSD is still an open field to explore and to research better ways to work with AK in this environment.

5 Discussion

AKM is clearly a challenge in AGSD due to its inherent distributed and dynamic nature. We found nine approaches to overcome this challenge, but they have opportunities for improvement and diverse paths to explore. Also, we found that these nine approaches cover the three phases of the integrated KM cycle, although only 7% of the papers supports capture/creation in a formal way, i.e. structured or aligned to a standard. Thus the shared AK is more prone to be vaporised. In the following paragraphs we conduct a general discussion about these two findings.

Regarding SwDev support, we found attempts to use repositories, wikis and groupware tools to store AK (as in Yanzer Cabral *et al.* [26] review); however, they are highly dependent on the stakeholders discipline to efficiently manage knowledge. For example, developers must add quality comments along code commits, or they must fill wikis or knowledge repositories with quality AK content. Since the Dingsøyr and Conradi study in 2002 [21], it has been emphasised that developers should collect and distribute knowledge. However, in ASD this is difficult due to time pressures [82]. Thus, it is important that tools integrated to the flows of development cycles, such COACH-IT [42] (based on ADL) and Archinotes [43] (based on informal notation), be tested on industrial settings to consider them as a reliable solution.

Also, since time pressures are common in Agile environments, auto-generated artefacts seem as a good option to codify AK, as they require a lower effort from developers. We found only four papers about this topic, which present solutions to show system structures [47], to generate experts' directories [76] (as in Bjørnson and Dingsøyr [22]), to trace artefacts [45, 46], and to relate email messages with user stories [44]. We consider that a new path to explore is automatic artefact generation from repositories commit messages or from logs of electronic textual media, since these are everyday tools for developers. Thus, artefacts would generate from natural interactions in the Agile process (in an unobtrusive way), and tacit AK could be coded during teammates interactions.

Furthermore, we found that AK documentation (designs and decisions) in AGSD tends to be *ad-hoc* and informal [34, 43] (such in Yanzer Cabral *et al.* [26] review), and consequently context-dependent. In fact, around 30% of the selected papers that report capture/creation support use an unstructured way to code AK. Lightweight artefacts are an alternative to code AK in an Agile manner (Yang *et al.* [27] found a works predominance towards AK coding in ASD), since they are designed to be less time-consuming than the traditional ways, and to be understandable regardless the stakeholder's language. Nonetheless, we must consider that Agile developers are reluctant about documentation, since they consider it as a secondary and non-creative activity [83]. Here we could have a blend with methodological modifications, an architect role could assume the responsibility to code AK using lightweight artefacts. This was also suggested in the reviews of Ali *et al.* [24] and Beecham *et al.* [25]. Another option is that AK be coded in groups during Agile ceremonies, such as sprint planning or project kick-offs. In this manner, the lightweight models generated in those meetings could be used as a base to discuss AK issues during the development cycle. Considering that architectural meetings could be held between local and remote teammates, a digital information radiator could display lightweight models to support discussions.

Continuing with methodological modifications, architect role and architecting phases could introduce an overhead that affects agility and delivery times. Thus, architects must be careful to manage AK in an Agile way. Also, we observed that all the AGSD studies where an architect role was introduced, were conducted on big companies (+ 50 employees). This means that adding an architect in small AGSD companies could be inconvenient due to structure and budget issues. Adding an architecting phase could be better for small companies.

In addition, we found that a popular way to share AK is by videoconference; however, knowledge is still tacit, unless videoconferencing sessions were recorded. Even though these sessions were recorded, AK would be unstructured and this would affect AK retrieving. In order to ease AK retrieving from recorded sessions, video files must be analysed with natural language processing techniques, or at least, these video files must be tagged later.

Concerning the reported benefits about using an AKM approach in AGSD, we only identified eight papers that report them (see Table 4). These benefits are aligned with the results of Yang *et al.* [27], particularly in terms of supporting architectural design and description, architectural decision process, and architectural understanding and communication. This coincidence suggests that a considerable number of benefits obtained in the architecture-agility combination could be obtained in the implementation of AKM in AGSD as well; however, we require more studies to assert this.

Finally, our discussion was about increasing the amount of AK available in a formalised way. This systematic mapping shows that AGSD companies work with the risk of AK vaporisation, since they prefer personalisation strategies to manage AK (Yang *et al.* [27] identified AK vaporisation as challenge in the Agile-architecture combination). Working in this manner causes that when the personnel leaves a team or the company, the knowledge also leaves, since there is a minimum amount of AK coded. Thus, we consider important to promote AK codification in AGSD.

6 Validity of the review process

The presented review process conforms to the guidelines proposed by Petersen *et al.* [28]. We only identify three variations, which we consider do not jeopardise the validity of our results. These variations are described as follows:

- *Research question definition.* At the beginning of the review process, we only considered one question to guide the paper searching (**RQ1** – How do AGSD companies manage AK in their projects?). As a result of the revision of Petersen *et al.* guidelines, this question was formally restructured, and we defined a second question (**RQ2** – Which KM phases are prominent in the AKM of AGSD companies?), which emerged during the review.
- *Search string definition.* The search string, although not initially defined following the guidelines proposed by Petersen *et al.*, included the elements that the PICO strategy [30] suggests (which is also suggested by Petersen *et al.*). This search string was structured based on previous experiences of the authors in the participation in processes of systematic reviews and mappings.
- *Keywording.* Petersen *et al.* specify that this step must be based only on the papers' abstracts. However, we identified keywords based on conceptual maps obtained from a rapid reading of each paper; this aiming at finding hidden AKM approaches regardless the main objective of the paper.

Another threat to validity could be that only one researcher (first author) executed paper searching, selection and classification, because this could have introduced a bias in the final result of the review. However, we consider that bias could be decreased in the process since the searching and filtering processes were executed on two different moments, and the classification scheme and paper classification were refined on three moments (which is referred as test and re-test in [28]), as we specified in Section 3. Also, the first author of this paper has been a practitioner of software engineering for + 10 years, and particularly for 4 years on AGSD.

7 Conclusions and future work

In this paper, we presented a systematic mapping review where we identified and described nine approaches to manage AK in AGSD teams. These approaches can be grouped as documentation artefact-based, communication-based, and methodological-based; where documentation approaches were prominent (45% of the papers). Further, we found that the selected papers evenly support the three phases of the integrated KM cycle: capture/creation (35%), sharing/dissemination (33%), and acquisition/application (32%). However, only 7% of the papers report a formalised way to code AK. Thus, we cannot firmly state that AKM is fully supported

in AGSD, since almost half of the papers reporting capture/creation support, are working with volatile knowledge, i.e. they work with unstructured and tacit AK.

Also, we discussed about the nine approach characteristics and about how to improve them to increase the amount of formalised (at least documented) AK, and then decrease AK vaporisation in AGSD. Another axis on our discussion was about how to create AKM solutions in an unobtrusive way, i.e. trying to manage AK on the natural flows of the Agile cycle. We can highlight critical points to implement an efficient AKM in AGSD: (i) add a role or a phase to address AK issues; (ii) use lightweight ways to code AK, (iii) auto-generate artefacts from electronic textual media interactions (to capture part of the tacit AK), and (iv) disseminate auto-generated and lightweight artefacts through wikis, repositories or groupware tools.

In addition, we found that KM and AKM strategies in software engineering are similar, as evidenced by the coincidences with related literature reviews; however, each emphasises different areas depending on the approach (i.e. GSD, ASD or AGSD). For instance, in AGSD the AKM strategies must emphasise automatic or lightweight manners to codify knowledge, in order to spend less time in documentation activities.

Finally, in this paper, we observed that AKM is still an open issue in AGSD; however, there are promising efforts to close this gap. As future work we aim at designing a mechanism to gather and classify developers' interactions over electronic textual media in an unobtrusive way. Then, we will evaluate what is the value of this mechanism for AGSD developers.

8 Acknowledgments

The authors were grateful to CONACyT for supporting this study with scholarship 394125 for the first author.

9 References

[1] Herbsleb, J.D.: 'Global software engineering: the future of socio-technical coordination', Int. Conf. on Software Engineering, 2007, p. 10

[2] Conchúir, E.Ó., Ågerfalk, P.J., Olsson, H.H.: 'Global software development: where are the benefits?', *Commun. ACM*, 2009, **52**, (8), pp. 127–131

[3] Vizcaíno, A., García, F., Piattini, M., *et al.*: 'El Desarrollo Global del Software', in Vizcaíno, A., García, F., Piattini, M. (Eds.): '*Desarrollo global de software*' (Rama, 2014), pp. 37–58

[4] da Silva, F.Q.B., Prikladnicki, R., França, A.C.C., *et al.*: 'Research and practice of distributed software development project management: a systematic mapping study', *Inf. Syst. Technol.*, 2012, **24**, (6), pp. 625–642

[5] Sommerville, I.: '*Software engineering*' (Addison-Wesley, 2010, 9th edn.)

[6] Boehm Turner, R.: '*Balancing agility and discipline: a guide for the perplexed*' (Addison-Wesley Longman Publishing Co., Inc., 2003)

[7] Smith, G., Sidky, A.: '*Becoming Agile: in an imperfect world*' (Manning, 2009)

[8] Dumitriu, F., Oprea, D., Mesnita, G.: 'Issues and strategy for Agile global software development adoption', Gavriluta, N., Raducanu, R., Iliescu, M., *et al.* (Eds.): '*Proceeding of the 3rd World Multiconference on Applied Economics, Business and Development, AEBD*' (WSEAS Press, 2011), pp. 37–42

[9] Jalali, S., Wohlin, C.: 'Global software engineering and Agile practices: a systematic review', *J. Softw. Evol. Process*, 2012, **24**, pp. 643–659

[10] Hansen, M.T., Nohria, N., Tierney, T.: 'What's your strategy for managing knowledge?', *Harv. Bus. Rev.*, 1999, **77**, (2), pp. 106–116

[11] Dorairaj, S., Noble, J., Malik, P.: 'Knowledge management in distributed Agile software development'. 2012 Agile Conf., 2012, pp. 64–73

[12] Jiménez, M., Piattini, M., Vizcaíno, A.: 'Challenges and Improvements in distributed software development: a systematic review', *Adv. Softw. Eng.*, 2009, **2009**, (710971), p. 14

[13] Bass, L., Clements, P., Kazman, R.: '*Software architecture in practice*' (Addison-Wesley Professional, 2003, 2nd edn.)

[14] Dingsøyr, T., Vliet, H.V.: 'Introduction to software architecture', no date, (**7491**)

[15] Kruchten, P., Lago, P., Vliet, H.V.: 'Building up and reasoning about architectural knowledge'. Proc. of the Second Int. Conf. on Quality of Software Architectures, 2006, pp. 43–58

[16] Vliet, H.V.: 'Software architecture knowledge management'. 19th Software Engineering, ASWEC 2008, 2008, pp. 24–31

[17] Holz, H., Maurer, F.: 'Knowledge management support for distributed Agile software processes', *Adv. Learn. Softw. Organ.*, 2003, **2640**, pp. 60–80

[18] Uikey, N., Suman, U., Ramani, A.: 'A documented approach in Agile software development', *Int. J. Softw.*, 2011, **2**, (2), pp. 13–22

[19] Levy, M., Hazzan, O.: 'Knowledge management in practice: the case of Agile software development'. 2009 ICSE Work. Coop. Hum. Asp. Software Engineering, 2009, pp. 60–65

[20] Beck, K., Beedle, M., van Bennekum, A., *et al.*: 'The Agile manifesto', 2001

[21] Dingsøyr, T., Conradi, R.: 'A survey of case studies of the use of knowledge management in software engineering', *Int. J. Softw. Eng. Knowl. Eng.*, 2002, **12**, (4), pp. 391–414

[22] Bjørnson, F.O., Dingsøyr, T.: 'Knowledge management in software engineering: a systematic review of studied concepts, findings and research methods used', *Inf. Softw. Technol.*, 2008, **50**, (11), pp. 1055–1068

[23] Earl, M.: 'Knowledge management strategies: toward a taxonomy', *J. Manag. Inf. Syst.*, 2001, **18**, (1), pp. 215–233

[24] Ali, N., Beecham, S., Mistrik, I.: 'Architectural knowledge management in global software development: a review'. Fifth IEEE Int. Conf. on Global Software Engineering (ICGSE), 2010, 2010, pp. 347–352

[25] Beecham, S., Noll, J., Richardson, I., *et al.*: 'Crafting a global teaming model for architectural knowledge'. Proc. 5th Int. Conf. Global Software Engineering, ICGSE 2010, 2010, pp. 55–63

[26] Yanzer Cabral, A.R., Ribeiro, M.B., Noll, R.P.: 'Knowledge management in Agile software projects: a systematic review', *J. Inf. Knowl. Manag.*, 2014, **13**, (1), p. 1450010

[27] Yang, C., Liang, P., Avgeriou, P.: 'A systematic mapping study on the combination of software architecture and Agile development', *J. Syst. Softw.*, 2016, **111**, pp. 157–184

[28] Petersen, K., Vakkalanka, S., Kuzniarz, L.: 'Guidelines for conducting systematic mapping studies in software engineering: an update', *Inf. Softw. Technol.*, 2015, **64**, pp. 1–18

[29] Dalkir, K.: 'Knowledge management in theory and practice', 2005

[30] Kitchenham, B., Charters, S.: 'Guidelines for performing systematic literature reviews in software engineering', 2007

[31] Stol, K.J., Babar, M.A., Avgeriou, P., *et al.*: 'A comparative study of challenges in integrating open source software and inner source software', *Inf. Softw. Technol.*, 2010, **53**, (12), pp. 1319–1336

[32] Capilla, R., Jansen, A., Tang, A., *et al.*: '10 years of software architecture knowledge management: practice and future', *J. Syst. Softw.*, 2015, **116**, pp. 191–205

[33] Clerc, V., Lago, P., Vliet, H.V.: 'Architectural knowledge management practices in Agile global software development'. 2011 IEEE Sixth Int. Conf. on Global Software Engineering Workshop, 2011, pp. 1–8

[34] Bruun, L., Hansen, M.B.: 'Handling design-level requirements across distributed teams: developing a new feature for 12 Danish mobile banking apps'. 22nd Int. Requirements Engineering Conf. (RE), 2014, 2014, pp. 335–343

[35] Rost, D., Naab, M., Lima, C., *et al.*: 'Architecture documentation for developers: a survey', in Drira, K. (Ed.): '*Software architecture*' (Springer Berlin Heidelberg, 2013), pp. 72–88

[36] Nowak, M., Pautasso, C.: 'Architectural decision modeling with reuse: challenges and opportunities'. 'SHARK '10 Proc. of the 2010 ICSE Workshop on Sharing and Reusing Architectural Knowledge, 2010, pp. 13–20

[37] Babar, M.A.: 'An exploratory study of architectural practices and challenges in using Agile software development approaches'. 2009 Joint Working IEEE/IFIP Conf. on Software Architecture & European Conf. on Software Architecture, 2009, pp. 81–90

[38] Razzak, M.A., Smite, D.: 'Knowledge management in globally distributed Agile projects – lesson learned'. Tenth Int. Conf. on Global Software Engineering (ICGSE), 2015, 2015, pp. 81–89

[39] Van Hillegersberg, J., Ligtenberg, G., Aydin, M.N.: 'Getting Agile methods to work for Cordys global software product development', in Kotlarsky, J., Willcocks, L.P., Oshri, I. (Eds.): '*New studies in global IT and business service outsourcing*' (Springer Berlin Heidelberg, 2011), pp. 133–152

[40] Kim, W., Chung, S., Endicott-popovsky, B.: 'Software architecture model driven reverse engineering approach to open source software development'. Proc. of the 3rd Annual Conf. on Research in Information Technology, 2014, pp. 9–14

[41] Stettina, C.J., Heijstek, W.: 'Necessary and neglected? An empirical study of internal documentation in Agile software development teams'. Proc. of the 29th ACM Int. Conf. on Design of Communication (SIGDOC 2011), 2011, pp. 159–166

[42] Read, K., Maurer, F.: 'Issues in scaling Agile using an architecture-centric approach: a tool-based solution', *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinfor.)*, 2003, **2753**, pp. 142–150

[43] Urrego, J., Munoz, R., Mercado, M., *et al.*: 'Archinotes: a global Agile architecture design approach', *Lect. Notes Bus. Inf. Process.*, 2014, **179 LNBIP**, pp. 302–311

[44] Sohan, S.M., Richter, M.M., Maurer, F.: 'Auto-tagging emails with user stories using project context', *Lect. Notes Bus. Inf. Process.*, 2010, **48 LNBIP**, pp. 103–116

[45] Antonino, P.O., Keuler, T., Antonino, P., *et al.*: 'A non-invasive approach to trace architecture design, requirements specification, and Agile artifacts'. 23rd Australian Software Engineering Conf. (ASWEC), 2014, 2014, pp. 220–229

[46] Gayer, S., Herrmann, A., Keuler, T., *et al.*: 'Lightweight traceability for the Agile architect', *IEE Comput.*, 2016, **49**, (5), pp. 64–71

[47] O'Reilly, C., Bustard, D., Morrow, P.: 'The war room command console: shared visualizations for inclusive team coordination'. Proc. of the 2005 ACM Symp. on Software Visualization, 2005, pp. 57–65

[48] Pérez, J., Díaz, J., Garbajosa, J., *et al.*: 'Flexible working architectures: Agile architecting using PPCs', *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinfor.)*, 2010, **6285 LNCS**, pp. 102–117

[49] de Graaf, K.A., Liang, P., Tang, A., *et al.*: 'An exploratory study on ontology engineering for software architecture documentation', *Comput. Ind.*, 2014, **65**, (7), pp. 1053–1064

[50] Díaz, J., Pérez, J., Garbajosa, J.: 'Agile product-line architecting in practice: a case study in smart grids', *Inf. Softw. Technol.*, 2014, **56**, (7), pp. 727–748

[51] Zimmermann, O., Wegmann, L., Koziolek, H., *et al.*: 'Architectural decision guidance across projects-problem space modeling, decision backlog

management and cloud computing knowledge'. Proc. 12th Working IEEE/IFIP Conf. on Software Architecture, WICSA 2015, 2015, pp. 85–94
[52] Sharp, H., Giuffrida, R., Melnik, G.: 'Information flow within a dispersed Agile team: a distributed cognition perspective', in Wohlin, C. (Ed.): '*Agile processes in software engineering and extreme programming SE – 5*' (Springer Berlin Heidelberg, 2012), pp. 62–76
[53] Young, C., Terashima, H.: 'How did we adapt Agile processes to our distributed development?' Proc. Agile 2008 Conf., 2008, pp. 304–309
[54] Šmite, D., Ågerfalk, P.J., Moe, N.B.: 'Coordination between global Agile teams: from process to architecture'. Agility Across Time and Space: Implementing Agile Methods in Global Software Projects, 2010, pp. 1–341
[55] Esbensen, M., Tell, P., Cholewa, J.B.*, et al.*: 'The dBoard: a digital scrum board for distributed software development'. Proc. of the 2015 Int. Conf. on Interactive Tabletops & Surfaces – ITS '15, 2015, pp. 161–170
[56] Schmidt, N., Meures, C.: ''Mind the Gap': an analysis of communication in Agile global outsourced software development projects'. 49th Hawaii Int. Conf. on System Sciences, 2016, pp. 501–510
[57] Korkala, M., Maurer, F.: 'Waste identification as the means for improving communication in globally distributed Agile software development', *J. Syst. Softw.*, 2014, **95**, pp. 122–140
[58] Paasivaara, M., Lassenius, C.: 'Communities of practice in a large distributed Agile software development organization – case Ericsson', *Inf. Softw. Technol.*, 2014, **56**, (12), pp. 1556–1577
[59] Harrison, R., Veerappa, V.: 'Social media collaboration in software projects', in Ruhe, G., Wohlin, C. (Eds.): '*Software project management in a changing world*' (Springer Berlin Heidelberg, 2014), pp. 401–4014
[60] Yagüe, A., Garbajosa, J., Díaz, J.*, et al.*: 'An exploratory study in communication in Agile global software development', *Comput. Stand. Interfaces*, 2016, **48**, pp. 184–197
[61] Phalnikar, R., Deshpande, V.S., Joshi, S.D.: 'Applying Agile principles for distributed software development'. Proc. Int. Conf. on Advanced Computer Control, ICACC 2009, 2009, pp. 535–539
[62] Moe, N.B., Börjesson, A., Andréasson, P.: 'Networking in a large-scale distributed Agile project'. Proc. of the 8th ACM/IEEE Int. Symp. on Empirical Software Engineering and Measurement, 2014, p. 8
[63] Eeles, P.: 'Building a platform for innovation: architecture and Agile as key enablers', in Babar, M.A., Brown, A.W., Koskimies, K., Mistrik, I. (Eds.): '*Agile software architecture: aligning Agile processes and software architectures*' (Elsevier Inc., 2013), pp. 315–333
[64] Bass, J.M.: 'Artefacts and Agile method tailoring in large-scale offshore software development programmes', *Inf. Softw. Technol.*, 2016, **75**, pp. 1–16
[65] Martini, A., Bosch, J.: 'Role of architects in Agile organizations', in Bosch, J. (Ed.): '*Continuous software engineering*' (Springer International Publishing, 2014), pp. 1–226
[66] Shen, X., Li, Y., Sun, Y.: 'An Agile enterprise architecture-driven model for geographically distributed Agile development', *Transform Healthcare Through Inf. Syst.*, 2016, **17**, pp. 185–197
[67] Costa, N., Santos, N., Ferreira, N.: 'Delivering user stories for implementing logical software architectures by multiple scrum teams', in Murgante, B., Misra, S., Rocha, A.C.*, et al.* (Eds.): '*Computational science and its applications – ICCSA 2014 SE – 55*' (Springer International Publishing, 2014), pp. 747–762
[68] Avritzer, A., Bronsard, F., Matos, G.: 'Improving global development using Agile – how Agile processes can improve productivity in large distributed projects', in Šmite, D., Moe, N.B., Ågerfalk, P.J. (Eds.): '*Agility across time and space: implementing Agile methods in global software projects*' (Springer-Verlag, 2010), pp. 133–148
[69] Boehm, B., Lane, J.A., Koolmanojwong, S.*, et al.*: 'Architected Agile solutions for software reliant systems', in Dingsøyr, T., Dybå, T., Moe, N.B. (Eds.): '*Agile software development*' (Springer Berlin Heidelberg, 2010), p. 31
[70] Del Nuevo, E., Piattini, M., Pino, F.J.: 'Scrum-based methodology for distributed software development'. Proc. – 2011 6th IEEE Int. Conf. on Global Software Engineering, ICGSE 2011, 2011, pp. 66–74
[71] Tanveer, M.: 'Agile for large scale projects – a hybrid approach'. 2015 National Software Engineering Conf. (NSEC 2015), 2015, pp. 14–18
[72] Dingsøyr, T.: 'Strategies and approaches for managing architectural knowledge', in Babar, M.A., Dingsøyr, T., Lago, P., van Vliet, H. (Eds.): '*Software architecture knowledge management*' (Springer Berlin Heidelberg, 2009), pp. 59–68
[73] Klobas, J.E., Beesley, A.: '*Wikis: tools for information work and collaboration*' (Chandos, 2006)
[74] Paredes, J., Anslow, C., Maurer, F.: 'Information visualization for Agile software development teams'. 2014 Second IEEE Working Conf. on Software Visualization, 2014, pp. 157–166
[75] Ellis, C.A., Gibbs, S.J., Rein, G.L.: 'Groupware: some issues and experiences', *Commun. ACM*, 1991, **34**, (l), pp. 39–58
[76] Moraes, A., Silva, E., da Trindade, C.*, et al.*: 'Recommending experts using communication history'. Second Int. Workshop on Recommendation Systems for Software Engineering – RSSE '10, 2010, pp. 41–45
[77] Wenger, E., McDermott, R., Snyder, W.: '*Cultivating communities of practice: a guide to managing knowledge*' (Harvard Business School Press, 2002)
[78] Cockburn, A.: 'Agile software development joins the 'Would-Be' crowd what's your center?', 2002, (January), pp. 6–12
[79] Eloranta, V., Koskimies, K.: 'Aligning architecture knowledge management with Scrum'. Proc. of the WICSA/ECSA 2012, 2012, pp. 112–115
[80] Ferreira, N., Santos, N., Soares, P.*, et al.*: 'Transition from process- to product-level perspective for business software', in Poels, G. (Ed.): '*Enterprise Information Systems of the Future: 6th IFIP WG 8.9 Working Conference, CONFENIS 2012, Ghent, Belgium, 19–21 September 2012, Revised Selected Papers*' (Springer Berlin Heidelberg, 2013), pp. 268–275
[81] Nonaka, I., Takeuchi, H.: '*The knowledge-creating company: how Japanese companies create the dynamics of innovation*' (Oxford University Press, 1995)
[82] Sneed, H.M.: 'Dealing with technical debt in Agile development projects', *Lect. Notes Bus. Inf. Process.*, 2014, **166 LNBIP**, pp. 48–62
[83] Clear, T.: 'Documentation and Agile methods: striking a balance', *SIGCSE Bull.*, 2003, **35**, (2), pp. 12–13