An Approach to Software Architecting in Agile Software Development Projects in Education

Samuil Angelov(✉) and Patrick de Beer

Software Engineering, Fontys University of Applied Sciences, Eindhoven, The Netherlands
{s.angelov,p.debeer}@fontys.nl

Abstract. The architecting activities in agile software development methods are weakly defined and even sometimes neglected. While there is sufficient literature on how software architectures and the architecting activities could be approached in agile projects, there is little information on how this topic should be treated in the education of software engineering students. In this paper, we propose an approach to the architecting activities in agile software projects in a bachelor software engineering course. The approach is inspired by theoretical and industry sources and is tailored to fit our educational goals and context. Our first experiences from the application of the approach show improved and deepened treating of software architectures, clarity on the purpose of the executed architecting activities, and improved student motivation.

Keywords: Software architecture · Agile · Method · Scrum · Teaching · Education

1 Introduction

The potential tension between software architecting and agile development methods has been discussed in numerous publications. A general consensus seems to exist around the value of paying explicit attention to software architectures in agile projects. Various suggestions have been made on how to approach software architecting in agile projects [1–3]. The introduction of software architectures and architecting related activities in educational curriculums has been a topic in several publications and multiple challenges and possible solutions have been outlined [4–6]. The coupling between agile projects and software architectures in education has not yet been sufficiently addressed in the literature. In [7], an approach for treating software architectures in agile projects in education is discussed. The approaches focuses on the architecture design phase in agile projects and specifically on the role of the stakeholders in it but does not address the actual dynamics of an agile project.

In our curriculum, we have a course in which software systems are developed by groups of students using an agile (Scrum-based) method. Among others, the students need to apply and demonstrate in this course competences in system design. In the past, we have required the design and documentation of the system's software architecture but did not provide any further guidance on how this had to be done in an agile project, relying on knowledge on software architectures from a previous course. We

D. Weyns et al. (Eds.): ECSA 2015, LNCS 9278, pp. 157–168, 2015.
DOI: 10.1007/978-3-319-23727-5_13

have observed the lack of motivation among students to deepen on the software architecture aspects, focusing primarily on the development process. The system architecture was hastily and superficially discussed in the teams at the beginning of the projects and documented at the end of the projects to pass the course, without realizing the purpose and benefits of architecting in an agile project. To improve the educational process, we need to redesign the course in a form where software architectures receive proper attention from the students in a non-enforced manner, ensuring understanding of the values of software architecting in agile projects. The challenges are, however, of dual nature. An approach for treating software architecting in agile student projects needs to resolve traditional problems in teaching and applying software architectures in combination with the challenges introduced by the agile practices which do not emphasize the architecture topic.

In this paper, we present our approach towards the treatment of software architectures in agile software projects in education. We have reviewed publications in the area of software architectures, approaches to software architectures in agile software development, and teaching of software architectures. In parallel, we conducted interviews with practitioners to collect data on how software architecting in agile projects is treated in the industry. The results from these two sources of information provide us with the basis for the definition of our approach.

The paper is structured as follows. In section 2, we present our context. In Section 3, we discuss our finding from the sources reviewed. In Section 4, we present our approach. In Section 5, we discuss our first experiences and lessons learned. We end the paper with conclusions.

2 Context and Status

Fontys University of Applied Sciences focuses on teaching practice oriented knowledge and skills. Upon completion of their studies, students from the software engineering bachelor school are skilled software engineers. The course with an acronym PTS4 is 6 ECTS credits (European Credit Transfer and Accumulation System credits) and is given in the second semester of the second year of the software engineering study. PTS4 is a project-based course, i.e., knowledge from other courses is applied in this course for the design and development of a software system. PTS4 focuses on applying agile practices in a Java-based, software development project. PTS4 is preceded by a project course PTS3[1]. The students work in groups of 5-6 students for 18 weeks. The students are allowed to form the groups themselves, which typically leads to balanced groups in terms of skills, interests and motivation. Each week, the students work one full day on the project in a dedicated project room. On a weekly basis, the students are visited in the project room by their Product Owner (PO) and Tutor (typically both roles are performed by one teacher). Scrum masters are students from

[1] In PTS3, a waterfall-based software development method is used. The students follow a path for requirements elicitation and documentation, architecture elicitation and documentation, implementation, testing. They elaborate an architecture document consisting of use case, class, sequence, component and deployment diagrams and a list of non-functional requirements.

the teams. A commercial tool for agile project management (which also allows time management) is used by all groups. The project is divided into 4 sprints of 4 weeks with a couple of weeks left as spare. The project is based on a predefined case ("PhotoStore") for the development of an on-line shop for ordering of customized photos optionally printed on a product. In terms of quality attributes, the focus in this project lies mainly on usability, security, and performance. Example architectural choices that student face are: desktop versus web application, type of clients to the main application, communication techniques, security approach, storage and access of photos. The case lacks novelty and is therefore of a somewhat trivial architecture nature: "most projects are not novel enough to require a lot of architectural effort" [8]. Typically, the discussion, evaluation and documentation of a software architecture is a nuisance for the students. They do not perceive any value in performing these activities and prefer to focus on software development. Discussions on the architecture take 5-10 minutes, with no attention being paid to the quality attributes. Although, the students know what quality attributes are, their understanding of the value of architecting and the role of quality attributes is rather limited. The combination of low architecture complexity and little attention on quality attributes and architectural choices often results in functionally satisfactory software products but partially or fully disregarding crucial quality attributes. Low-level designs (class diagram, database model) are made by the students primarily influenced by the way of working in PTS3. Due to the lack of an incentive (no future usages, lack of communication obstacles, and non-demanding POs), the motivation for documenting a software architecture is low and is done only to satisfy the course requirements.

3 Sources of Inspiration

To accumulate knowledge for our course re-design, we have studied literature on software architectures and on approaching them in agile projects and in education. Furthermore, we have interviewed software companies from the region where our students get predominantly employed.

3.1 Literature on Software Architectures and their Usage in Agile Projects

On software architectures: The architecting process involves the analysis, synthesis, documentation, and evaluation activities [15]. The software architecture of a software-intensive system is created in the early development phases and evolves throughout the whole development cycle [16]. An overview of software architecture methods can be found in [17]. The term software architecture has been attributed various definitions. For example: "the set of structures needed to reason about the system, which comprise software elements, relation among them, and properties of both" [9] focuses on the structure aspect. Other definitions view design decision as the focus of software architectures [10], stating that a software architecture reflects "the major design decisions made" [5], and that "the actual structure, or architectural design, is merely a reflection of those design decisions" [11]. Architectural decisions

"capture key design issues and the rationale behind chosen solutions" [12]. However, as argued in [9], in agile projects some decisions are made later on throughout the project, and it is hard to justify whether a decision is major. Furthermore, as Kruchten notes, architectural decisions should not be mixed with software design and code decisions [8]. The study in [13] shows that "compared to senior architects, junior architects spend a quarter of the time on making a decision". Documenting decisions may be highly practical in agile projects: "If an architect doesn't have time for anything else, these decisions can provide a concrete direction for implementation and serve as an effective tool for communication to customers and management" [14]. With respect to the documentation of decisions a number of publication propose solutions. The IBM's e-Business Reference Architecture Framework is used in [14] to propose a template for the documenting of decisions. It is suggested that if a system quality is affected by a decision it needs to be documented.

On software architectures in agile development: In [15], the editors predict that "Software architecture will be recognized as a key foundation to agile software development". Few years later, this line is continued in a special issue of IEEE Software focused on the relationship between software architectures and agile practices. In its editorial, Abrahamsson et al. [2] provide a number of advises on the usage of software architectures in agile projects: making the architectural decisions "early enough", defining an architecture owner who is part of the team, usage of architectures for improving communication and coordination in complex projects and settings (e.g. distributed teams). They also state: "If agile developers don't consider software architecture relevant to their day-to-day activities, it would be difficult, even impossible, to convince them to use architectural principles and integrate artifacts in agile development". In [8], Kruchten binds the need for architectural activities and their size with the project context (e.g., project size, criticality, risks). In [16], the authors conclude that practices of software architectures and especially the focus on quality attributes can lead to improvement of projects executed with agile methods. They recommend focus on the overall system structure (shaped by the quality attributes) in the first iteration and subsequently in later iterations when changes are needed. In [17], the value of an architecting team in large-scale agile projects is reported. In [18], a study among IBM professionals reveals that architecting activities are highly relevant for agile developers. Communication, assumptions documentation, validation are supported through architectures. The project complexity also influences the architecture relevance (distribution, size, number of stakeholders).

On documenting software architectures in agile projects: Architecture documentation (and time associated with it) in agile projects is one of the roots of the tension between agile development methods and software architecting: "In an agile process, the team doesn't have time to wait for the architect to completely develop and document the architecture" [14]. In [19], the values and impediments of documenting software architectures in agile projects are discussed. Architecture documentation may serve to: get on board new project members quickly, for future usage (project transfer to another team/phase), quality assurance, or as domain knowledge repository. However, documenting of changing issues can be a wasted effort. Boehm [20] notes that for unpredictable projects decreasing the documentation effort may work,

while for projects with predictable requirements this approach may miss useful architectural practices and may have even negative impact by not allowing external reviewers to detect mistakes due to the lack of documentation. An approach to the usage of the SEI method for documenting software architectures in agile methods is proposed in [21]. Initially, the views to be documented and the project stakeholders interested in them are identified. Then, documentation is done on the need-to basis with minimum need for rework. It is advised to document rationale early and throughout the project as postponing this for the end of the project may lead to omissions of choices made, insights, etc. Kruchten considers all possibilities for architecture documentation: documentation in code, with metaphors, diagrams, complete architecture document, etc. depending on the project context [8]. In [22], the documentation was reduced to component diagrams and design decisions published on a wiki.

On architects in agile projects: A study among 10 professionals [22] revealed the existence of three types of architects: one on the client side (software architect), one with management functions (solution architect), and one responsible for the actual implementation in the team (management architect). The first and the latter roles are proposed also in [2,8]. The high-level architecture would be drafted by the software architects and the design decisions and concrete architecture by the implementation and solution architects.

3.2 Literature on Teaching Software Architectures

A course focusing on the analysis and evaluation of software architectures and building a knowledge-base for their design is presented in [4]. The course is targeted at master level students and presents the main ingredients of classical software architecture courses. The authors of [5] classify the courses on software architectures as either focusing on the "tools" to design software systems (patterns, languages, etc.) or focusing on communication aspects of software architectures. They discuss their experiences from two master-level courses focusing on the communication aspects. The objectives of the courses are to teach selection and development of architecture views and architecture assessing and do not incorporate system development activities. The courses follow an architecture-centric approach, where stakeholders should be involved and functional and non-functional requirements are simultaneously addressed during the architecture design process. In [6], the problems of teaching software architectures in academic, non-industrial settings are identified. As main challenges are seen the isolated nature of an academic course and therefore the lack of realistic context (starting from scratch, non-existent stakeholders) and the lack of the inherent fuzziness and complexity of industrial scenarios. Furthermore, as the authors point out, the students lack the experience of solving complex design problems and profound knowledge for the application domain. The authors describe an advanced master-level course designed to teach software architecting as a team effort in a complex problem environment. The architecting team reports to a "Board" (formed by staff members) which considers the solutions and makes choices. The board exemplifies to some extend the multiple stakeholders, working in context challenges delineated above. In [7], a course on tackling software architectures in agile projects is presented. The approach in [7] provides us with valuable directions

on architecting in agile projects. However, our course setup leads to different pedagogical goals and challenges (which we discuss in Section 4.1). In [23], an agile software development project course is described, where architecting activities take place in week 3. However, the architecting issues in agile projects are not extensively discussed.

3.3 Industry Study

In total, 12 companies implementing some form of agile software development (predominantly Scrum) were interviewed by students using a standardized questionnaire. The companies were small (3-4 developers), medium, and large (e.g., a bank). Interviews were held with developers, scrum masters, and project leaders. Naturally, the architecting practices varied substantially among the companies, depending on their domain, maturity, size, etc. Ten of the companies spend dedicated time in the beginning of the project to shape the overall architecture. One company does it on the fly and one does not spend time on architecting. One company has a two-step process of architecting. They discuss first a "functional architecture" with the Product Owner and then they elaborate a "technical architecture". Documenting of architectures varies from documenting at the beginning of a project (3 companies), per sprint (5 companies), or at the end if necessary (2 companies). Generally, the aim is to minimize documentation effort. One company stated that documentation effort depended on the project client and the team. As minimum companies document components/structures and their relations. One company stated explicitly that they document the architecture design choices. At one company is made use of a wiki for the architecture documentation. In [24], the experiences of a partner company of our school (among those interviewed) are reported. It is strongly advocated that quality attributes, architecture envisioning and system architecting are paid substantial attention from the beginning of the project, having in mind to only document what needs to be documented.

4 Course Design

Next, we present our main pedagogical considerations, the decisions that we made, rationale behind them, and the course setup.

4.1 Considerations

Our course design was influenced by a set of situational factors (as defined in [25]):

- *General context*: Our curriculum implied the application of an architecture-centric approach in the course (similar to the approaches in [5] and [7]).
- *Characteristics of the learners (second year bachelor students):*
  - *capabilities:* The Dutch education system predefines the students of applied universities as students predominantly interested in applying knowledge. This opens opportunities for students who are less inclined to reflect on the theoretical aspects of a problem, but it does not exclude students with capabilities for deeper reflections.

  - *motivation*: A student needs to be motivated during the course for achieving best learning results. Our students motivation is heavily influenced by the project case (realism, comprehension), degree of challenge (too easy or too difficult projects demotivate students), practical relevance of the course activities, and direct knowledge application.
  - *maturity:* As indicated in literature, junior developers (and even more bachelor students in an applied university) have difficulties in reflecting on architectural decisions due to their lack of experience and knowledge [6,13].
- *Nature of the subject (architecting in agile projects):* The project context defines the reasons to conduct architecting activities explicitly and in an organized manner [8,18]. To create an architecting suitable context, a project needs to be of sufficient complexity [6]: big enough, involving several quality attributes and should offer room for taking non-trivial architectural decisions (non-stable architecture) [8,16]. The project needs to be with relatively predictable requirements [20]. A desired contextual factor is team distribution or other communication or coordination hindering factors [2].

4.2 Course Design Decisions and Rationale

Based on the literature and industry input, and the situational factors discussed in Section 4.1, we have taken a number of decisions for our approach.

- *Project case*: We have decided to offer several cases to address the *motivation factor*. Students are allowed to choose the case that motivates them the best. The cases differ in degree of complexity (to address the diversity of student *capabilities*) and in types of architectural challenges offered (team distribution, room for architectural decisions, number of quality attributes, etc.).
- *Process organization:* We have decided to introduce an architecture role (as suggested in [2,22,26]) in order to anchor the architecting activities with clear responsibilities and to stimulate the professional development of students (related to the *general context* factors). This role is assigned to a volunteering student from the group for the project duration.
- *Architecture elicitation:*
  Similar to the approach in [7], we have decided to dedicate explicit time for the architecture design activities in the beginning of the project as there architecting would take place anyway [8] (see Section 4.3, week 1 and 2). In addition, in each sprint, the teams revisit their architecture and if needed adapt it.
- *Documenting*:
  - *When:* We have decided to dedicate time in the beginning of the project for the documentation of the initial architecture design (document rationale early and throughout the project [21]) and to stimulate documentation of architectural changes at the end of each iteration.
  - *What*: As minimum, we require documentation of a list of stakeholders identified, their concerns (based on [27]), a high-level architecture and rationale for the decisions made [14]. This content was selected as it is of direct value for the

Product Owner [8] and is therefore realistic requirement (related to the *motivation* factors). Following the agile principles, the teams could (but not necessarily) document detailed design aspects (class diagrams, sequence diagrams, database models, etc.) if they perceive this as relevant and useful for the team or needed in the context of the project.

  - *Why*: To motivate documenting in addition to the practical relevance stimuli, we introduced a number of stimulating contextual factors. As part of the school international activities, one project case involved co-operation on the project with Finish students (introducing team distribution [8]). A second project case stimulated the involvement of two or more groups working on it, introducing communication and synchronization challenges (which foster documenting). A third case involved a real client, which makes documenting a needed activity. Last but not least, the need was introduced for the PO to be able to quickly look-up architectural choices per group (as the PO has to switch between numerous groups loosing track of the group choices).
  - *Where*: We offered the teams to document either in a traditional document or in wiki page in the agile management tool aiming at reducing the risk of decoupling the development process from the architecture document [22,28]. We did not offer to include the stakeholders' concerns in sprint backlogs as concerns live beyond a sprint.
  - *How:* The teams were allowed to document minimalistic and efficient (e.g. photos of drawings instead of diagrams) in alignment with the agile lean principles.
- *Backlog content*: To support the team in managing their workload, we allowed them to put in the sprint backlog architecting activities [29]. As students are unexperienced in having options to consider and in making choices, we encouraged them to plan architecture and technology spikes (related to the *maturity* factor).

4.3 Course Organization for the Architecting Activities

Next, we describe the organization of the architecting activities within PTS4.

1. In week 1 (4h), the students form groups, familiarize themselves with the set of available cases, select a scrum master and an architect. The existing knowledge on software architectures is revisited, focusing on stakeholders, non-functional requirements and their interplay with the architectural decisions. After the selection of a case, each group, in a discussion with the PO, identifies and lists the stakeholders and their business goals. Next, in a discussion with the PO, the project backlog is populated with user stories. Throughout the week, the architects in collaboration with the rest of the team need to elaborate an architectural proposal with rationale for the choices made. The reason to expand this activity beyond the lesson is that students need time to reflect on the problem, to research possible solutions and their advantages and disadvantages. Team members may be involved in elaborating detailed designs when a team decides to do so.
2. In week 2 (4h), the architects present their proposals to the PO (the high-level architecture and the rationale for the choices made). The PO reflects on the proposal.

The discussion is centered around the interplay between the stakeholders' goals and the architecture proposed. In the rest of the lesson time, the team works on the scrum-related activities (e.g., estimates user stories, prepares sprint 1).

3. In week 3, the PO may make certain adaptation to the sprint backlog and the team can begin working on the sprint tasks.
4. During the sprint or at its end, the team is allowed to adapt the architecture if needed. The rationale for a change is recorded and is discussed with the PO.
5. At the end of the sprint and the beginning of the next one, the PO reviews the architecture document (and if there were changes applied to it in the last sprint). The team and the PO reflect on the architecture and the stakeholders concerns - are changes needed, are the architecture choices still in compliance with the goals, etc.

Steps 4 and 5 are repeated for each sprint. The final architecture document/wiki is handed in in week 18 and the architecting process and decisions are revisited in a discussion with the Tutor.

5 Initial Experiences and Lessons Learned

We have applied our approach for the first time in the beginning of 2015. 64 students in 12 groups with 2 teachers acting in the role of PO were involved in the experiment. The students were offered three cases: CIMS (Crisis Information Management System) representing an architecturally complex case, PhotoStore (representing an architecturally less challenging case), and The Hub (a system comparable to the PhotoStore system but requested by an actual client). The international project planned as a distributed effort was not included in the experiment due to organizational problems in it. Five groups have chosen to work on the CIMS case (of which 3 in a collaborative project), 1 group executed a project for a real client and 6 groups selected the PhotoStore case. Currently, the project is in its 10^{th} week.

The explicit focus in week 1 and 2 on the architecting activities had a clear, positive effect. Compared to earlier course executions, the students spent time on the software architecture and the decisions were made in a conscious way in discussions with the PO. Input and questions from the PO resulted in changes of initial choices or in their better argumentation. The students have realized (and stated this) that stakeholders have different perspectives and may have conflicting concerns. One PhotoStore group and 2 CIMS groups needed also week 3 for architecture elicitation activities. Our conclusion is that the work on the major architectural choices may be given space also in week 3 of the project. The CIMS groups left architectural choices open for later sprints where more time would be available to accumulate knowledge and where the context would be better known (e.g., push/pull strategy between certain components). The three collaborating CIMS groups stated explicitly that the architecting activities helped them in coming to a common understanding of the system structure and for the work division among the teams. In Sprint 2, several of the groups needed to revisit their initial choices. For example, in a CIMS group a new type of client application had to be added and in a PhotoStore group, a desktop client needed to be added to the initially envisioned web solution.

In terms of documentation, the needs of the PO and the team needs were the main driving factor to document. Several teams have documented class diagrams and one group has discussed classes but did not document them. The general feeling among students was that class diagrams had little value for them and only one group using the Spring framework needed them. They have acknowledged that they have documented them influenced by previous school requirements. Most groups made ERD diagrams claiming that this was important for the team synchronization and communication. All CIMS and PhotoStore teams had difficulties in communicating their architectural choices to the PO and documenting them. They found it difficult to translate the desired quality attributes into relevant architectural decisions. At the end, all groups used some variant of deployment diagrams accompanied by textual clarifications as a means to communicate their major architectural choices. However, deployment diagrams were unsuitable to express many of their choices which introduced a communication gap with the PO. We conclude that the refreshing lecture needs to focus on the high-level nature of architectures and be extended with information on functional modeling [30] and with architectural patterns that address certain project related quality attributes. In a discussion with the students, they have all approved this conclusion. They have also pointed out that for the PhotoStore case the functional modeling and patterns knowledge would be less crucial. The majority of groups have documented their architecture decisions in a word document. One group used an architecture wiki and one group resorted to photos in combination with diagram pictures. The group working for an actual client did not document any of their decisions in their first sprint. They were focused on making the decisions and discussing them with the stakeholders instead of documenting them. The omission to document their choices led to a misunderstanding with the PO and a delay of their project.

All groups embraced the inclusion of architectural activities in the backlog which they saw as protective mechanism, explicating their work on non-coding activities to the PO. The introduction of the architect role led to the explicit allocation of responsibility in the group on the architecting activities and served as an additional stimuli for the students to focus on architecting. The collaborating CIMS groups decided to have architecting discussion with a limited number of team representatives. They have involved the group architect and a second team member, limiting the discussions to a group of 6 people (instead of 15 they started off with).

The offering of cases of higher and lower architectural complexity has proven a valuable experiment. All groups felt sufficiently motivated. They could also work within the scope of their architectural capabilities and interests. Clearly, the PhotoStore case offers less architectural challenges than the CIMS case. This differentiation in the architecting competences demonstrated by the students in the project would be reflected in their personal development portfolios, currently adapted to reflect differences in competences demonstrated in a course.

6 Conclusions

We present an approach to introducing architecting activities in agile software development projects in education. Our approach is based on industry and academic approaches to architecting in agile projects. The application of the approach has led to

an improved and deepened treating of software architectures in the agile projects performed by the students. The strategies selected to introduce and motivate architecting activities in agile projects have proven to be effective and students have performed the activities (e.g., deliberating, designing, documenting) with the realization of their value for the project and in a non-enforced manner. We observed a knowledge gap in our education which hindered the students in producing architectural views relevant for the PO. This omission will be remedied in the next course execution.

The results presented in this paper are of value to other educational institutions where agile software development projects are part of the curriculum. Our approach is targeted to second year, applied study, software engineering students but it can be also used in higher years with minor modifications.

References

1. Breivold, H.P., Sundmark, D., Wallin, P., Larsson, S.: What does research say about agile and architecture? In: 2010 Fifth International Conference on Software Engineering Advances (ICSEA), pp. 32–37 (2010)
2. Abrahamsson, P., Babar, M.A., Kruchten, P.: Agility and Architecture: Can They Coexist? IEEE Software **27**, 16–22 (2010)
3. Sharifloo, A.A., Saffarian, A.S., Shams, F.: Embedding architectural practices into extreme programming. In: 19th Australian Conference on Software Engineering, ASWEC 2008, pp. 310–319 (2008)
4. Garlan, D., Shaw, M., Okasaki, C., Scott, C., Swonger, R.: Experience with a course on architectures for software systems. In: Sledge, C. (ed.) Software Engineering Education, vol. 640, pp. 23–43. Springer, Heidelberg (1992)
5. Lago, P., Van Vliet, H.: Teaching a course on software architecture. In: 18th Conference on Software Engineering Education & Training, pp. 35–42 (2005)
6. Mannisto, T., Savolainen, J., Myllarniemi, V.: Teaching software architecture design. In: Seventh Working IEEE/IFIP Conference on Software Architecture, WICSA 2008, pp. 117–124 (2008)
7. Cleland-Huang, J., Babar, M.A., Mirakhorli, M.: An inverted classroom experience: engaging students in architectural thinking for agile projects. In: Companion Proceedings of the 36th Int. Conf. on Software Engineering, pp. 364–371. ACM, Hyderabad (2014)
8. Kruchten, P.: Software architecture and agile software development: a clash of two cultures? In: ACM/IEEE 32nd Int. Conf. on Software Engineering, pp. 497–498 (2010)
9. Bass, L., Clements, P., Kazman, R.: Software Architecture in Practice. Addison-Wesley Professional (2012)
10. Jansen, A., Bosch, J.: Software architecture as a set of architectural design decisions. In: 5th Working IEEE/IFIP Conference on, Software Architecture, WICSA 2005, pp. 109–120 (2005)
11. de Boer, R.C., van Vliet, H.: On the similarity between requirements and architecture. Journal of Systems and Software **82**, 544–550 (2009)
12. Zimmermann, O.: Architectural Decisions as Reusable Design Assets. IEEE Software **28**, 64–69 (2011)
13. Tofan, D., Galster, M., Avgeriou, P.: Difficulty of architectural decisions – A survey with professional architects. In: Drira, K. (ed.) ECSA 2013. LNCS, vol. 7957, pp. 192–199. Springer, Heidelberg (2013)

14. Tyree, J., Akerman, A.: Architecture decisions: demystifying architecture. IEEE Software **22**, 19–27 (2005)
15. Kruchten, P.: The past, present, and future for software architecture. In: Henk, O., Judith, S. (eds.) IEEE Software, vol. 23, pp. 22–30 (2006)
16. Nord, R.L., Tomayko, J.E.: Software architecture-centric methods and agile development. IEEE Software **23**, 47–53 (2006)
17. Lindvall, M., et al.: Empirical findings in agile methods. In: Wells, D., Williams, L. (eds.) XP 2002. LNCS, vol. 2418, pp. 197–207. Springer, Heidelberg (2002)
18. Davide, F.: Peaceful coexistence: agile developer perspectives on software architecture. In: Giovanni, C., Salvatore Alessandro, S., Giuseppe, C., Paolo, S., Cristiana, D.A. (eds.) IEEE Software, vol. 27, pp. 23–25 (2010)
19. Coram, M., Bohner, S.: The impact of agile methods on software project management. In: 12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, ECBS 2005, pp. 363–370 (2005)
20. Boehm, B.: Get ready for agile methods, with care. Computer **35**, 64–69 (2002)
21. Clements, P., Ivers, J., Little, R., Nord, R., Stafford, J.: Documenting Software Architectures in an Agile World. Technical Note CMU/SEI-2003-TN-023, Carnegie Mellon University (2003)
22. Babar, M.A.: An exploratory study of architectural practices and challenges in using agile software development approaches. In: Joint Working IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture, WICSA/ECSA 2009, pp. 81–90 (2009)
23. Lee, J., Kotonya, G., Whittle, J., Bull, C.: Software design studio: a practical example. In: 37th Int. Conference on Software Engineering (ICSE 2015), pp. 47–55. IEEE (2015)
24. Schoeber, G.: Architecture and agile, friends or enemies? (presentation). In: SATURN 2010, Minneapolis, MN (2010)
25. Fink, L.D.: Creating Significant Learning Experiences: An Integrated Approach to Designing College Courses, 2nd edn. Jossey-Bass (2013)
26. Faber, R.: Architects as Service Providers. IEEE Software **27**, 33–40 (2010)
27. Kazman, R., Bass, L.: Categorizing Business Goals for Software Architectures. Technical Report CMU/SEI-2005-TR-021, ESC-TR-2005-021, Carnegie Mellon University (2005)
28. Clerc, V., Vries, E.D., Lago, P.: Using wikis to support architectural knowledge management in global software development. In: Proceedings of the 2010 ICSE Workshop on Sharing and Reusing Architectural Knowledge, pp. 37–43. ACM, Cape Town (2010)
29. Madison, J.: Agile Architecture Interactions. IEEE Software **27**, 41–48 (2010)
30. Brinkkemper, S., Pachidi, S.: Functional architecture modeling for the software product industry. In: Babar, M.A., Gorton, I. (eds.) ECSA 2010. LNCS, vol. 6285, pp. 198–213. Springer, Heidelberg (2010)