The Journal of Systems and Software 127 (2017) 78–90

Contents lists available at ScienceDirect

The Journal of Systems and Software

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

Designing and applying an approach to software architecting in agile projects in education

S. Angelov*, P. de Beer

Software Engineering, Fontys University of Applied Sciences, Postbus 347, 5600 AH Eindhoven, The Netherlands

ARTICLE INFO

Article history:
Received 12 February 2016
Revised 13 January 2017
Accepted 31 January 2017
Available online 1 February 2017

Keywords:
Software architecture
Agile
Scrum
Teaching
Software engineering education
Students
Project
Course

ABSTRACT

Software architecting activities are not discussed in most agile software development methods. That is why, the combination of software architecting and agile methods has been in the focus of numerous publications. However, there is little literature on how to approach software architecting in agile projects in education. In this paper, we present our approach to the introduction of software architecting activities in an agile project course. The approach is based on literature sources and is tailored to fit our educational goals and context. The approach has been applied in two consecutive executions of the course. We observe improved understanding on the value of architecting activities and appreciation among students on the combination of architecting activities and agile development. We applied the approach predominantly in cases with an architecturally savvy Product Owner. Further research is required to understand how the approach performs in scenarios with architecturally unsavvy Product Owners and if it needs to be adapted for these scenarios. We also conclude that more research is needed on the challenges that architects face in agile projects in order to better prepare students for practice.

1. Introduction

Software architecting has received ample attention in academia and industry and many educational institutions address software architectures in their curricula, (e.g., Garlan et al., 1992; Lago and Van Vliet, 2005; Mannisto et al., 2008). However, teaching software architectures remains a difficult task. It requires a realistic context, teamwork, sufficient complexity, and substantial coaching (Galster and Angelov, 2016).

With the increasing popularity of agile software development methods, the software architecting community has faced a new challenge. Most agile methods "pay very little attention to common architectural design activities" (Babar, 2014). For example, Kent Beck sees architectures as emerging and evolving in the daily design (Beck, 1999). Kruchten et al. predict that "Software architecture will be recognized as a key foundation to agile software development" (Kruchten et al., 2006). The question of how much architectural effort is needed in agile projects was rated as "the second-equal most burning question facing agile practitioners" (Freudenberg and Sharp, 2010). Researchers and practitioners have embraced this challenge and published approaches, visions, experiences, surveys on how software architecting can be (or is) approached in agile development methods. A general consensus seems to exist around the value of paying explicit attention to software architectures in agile projects. However, there are different approaches to architecting in agile projects. Jan Bosch states that the community is switching to a "user-driven architecture work" from the traditional "quality based architecting" (Mirakhorli and Cleland-Huang, 2013). Following the agile principles, teams should focus upfront on the requirements delivering value to the client rather than on the architecturally significant requirements, accepting that an initial architecture may not be the optimal architecture. He sees the price of constant refactoring acceptable given the benefits achieved in terms of agility and customer satisfaction. Waterman et al. draw a similar conclusion in their study (Waterman et al., 2013). Friedrichsen analyses the idea of emergent architecture and concludes that it is a well-suited approach with respect to elaborating detailed architectures, but it is not adequately supporting critical architecting activities like making of architectural choices, elaboration of high-level designs, and architecture validation (Friedrichsen, 2014).

The coupling between agile methods and software architectures in education has not yet been sufficiently addressed in literature. Cleland-Huang et al. present an approach to treating software architectures in agile projects in education (Cleland-Huang et al., 2014a). The approach focuses on the architecture design phase in agile projects and specifically on the role of the stakeholders but does not address the actual dynamics of an agile project.

* Corresponding author.
E-mail addresses: s.angelov@fontys.nl (S. Angelov), p.debeer@fontys.nl (P. de Beer).

http://dx.doi.org/10.1016/j.jss.2017.01.029
0164-1212/

In our curriculum, we have a course in which groups of students develop a software system using an agile (Scrum-based) method. One of the skills that they need to demonstrate in this course is system design. In the past, we have required a software architecture design and its documentation but did not provide any guidance on how this had to be done in an agile project. The students approached this activity either as an "end-of-project" documentation activity (in most cases) or as a "Big Design Up Front" activity (less common). The former behavior was driven by the agile principle of working software over comprehensive documentation, while the latter behavior was influenced by previously acquired knowledge on software architecting in waterfall projects. Groups exhibiting the former behavior were unmotivated to deepen on the software architecture aspects, focusing primarily on the development process. The 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. The lack of a clear approach to software architecting in agile projects resulted in invalid and unreliable grading of the software architecting skills of students.

To improve our course, we design and introduce an approach to software architecting. In this paper, we present the approach. An approach to software architecting in agile projects needs to resolve the traditional problems of teaching software architectures (e.g., realistic context, introducing teamwork, sufficient complexity, substantial coaching). In addition, it has to intertwine architecting and agile activities in a way that does not contradict the agile principles and practices. Therefore, we design our approach using publications on software architecting in agile development and on teaching software architectures, considering at the same time our educational context. We apply our approach in two consecutive course instances. This paper is a continuation of our work from Angelov and de Beer (2015). In this paper, we extend our findings by providing results from the application of the approach in two consecutive course executions and improvements to the approach based on our findings. Furthermore, we discuss the challenges that require further research by the software architecting research and education communities.

The paper is structured as follows. In Section 2, we present our context, problem, and research approach. In Section 3, we discuss our literature findings on the interplay between software architectures and agile methods. In Section 4, we present our approach. In Section 5, we discuss the application of the approach and lessons learned. We end the paper with conclusions.

2. Context, problem, and research approach

In this section, we describe our context and define our problem. We discuss literature on teaching software architectures and show that it does not provide a solution to our problem. Then, we present our research approach.

2.1. Course context

We analyze our context using the *situational factors* defined by Fink: general and specific context, nature of the subject, characteristics of the learners (Fink, 2013):

- *General context*: Fontys University of Applied Sciences is the largest Dutch university of applied sciences. It offers bachelor and master programs. Being an applied university, its focus is on teaching practice oriented knowledge and skills. In particular, it offers a software engineering, bachelor program. Upon completion of their studies, students from this program are skilled software engineers. The four-year program consists of 4 semesters of courses, followed by an internship semester, another courses-based semester, a free-choice minor study, and a final internship (graduation project). Within each courses-based semester, there is a "project-course" called PTSn (where "n" indicates the number of the semester: PTS2 – second semester, PTS3 – third semester, etc.). Project-courses apply knowledge from other courses for the development of a software system. The students are introduced to software architectures in their third semester in a course called GSO3. In PTS3, the students develop a software system using a waterfall development method. They perform requirements elicitation and documentation, architecture elicitation and documentation, implementation, and testing activities. The students elaborate an architecture document consisting of functional and non-functional requirements, use case, class, sequence, component, and deployment diagrams.
- *Specific context*: The course with acronym PTS4 is the focus of this work. PTS4 is 6 ECTS credits (European Credit Transfer and Accumulation System). The students work in groups of 5–6 students for 20 weeks, one day a week. The students are allowed to form the groups themselves, which typically leads to balanced groups in terms of skills, interests, and motivation. Our curriculum implies the application of architecting activities in PTS4.
- *Nature of the subject*: PTS4 focuses on applying agile practices in a Java-based project. It follows mainly the Scrum method, augmented with practices from other agile methods like Lean (eliminate waste) and Crystal Clear (osmotic communication, personal safety). All top 20 agile practices listed in Yang et al. (2016) are followed except for the "system metaphor". The family of agile methods are thoroughly studied in another course (BS41) given in parallel to PTS4. On a weekly basis, the students are visited by their Product Owner (PO) and Tutor (typically both roles are performed by one teacher). Scrum Masters are students from the teams. A commercial tool for agile project management is used by all groups. The project is divided into 4 iterations (sprints) of 4 weeks.
- *Characteristics of the learners*:
  - *Capabilities*: Students of applied universities are predominantly interested in applying knowledge and skills. They are typically less inclined on reflecting on theoretical aspects of a problem. However, there are also students with capabilities for deeper reflections.
  - *Motivation*: A student needs to be motivated during the course for achieving best learning results. The motivation of our students is influenced by:
    - realism: overlap between knowledge taught and activities performed in courses and real-life practices;
    - *purposefulness*: having a clear purpose for the knowledge taught and activities performed in courses;
    - *clarity*: unambiguous, easy to understand knowledge taught and activities performed in courses;
    - *degree of challenge*: knowledge and activities that are not too easy or too difficult.
  - *Maturity*: Typically students are about 20 years old, in the early stages of their professional development. They are relatively inexperienced in design and architecting activities.

2.2. Problem definition and desired situation

In the old setup of PTS4, we used a case called "PhotoStore". The students need to develop an application for ordering of customized photos that can be optionally printed on a product (e.g., on a T-shirt). The case is realistic, technically challenging, and having many features to build. 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" (Kruchten, 2010). In terms of quality attributes, the focus in this case is on usability, security, and performance. Example architectural choices that students face are: desktop versus web application, type of clients connecting to the main application, communication techniques, security approach, storage and access of photos. When defining their overall architecture, more advanced groups might sometimes extend the typical configuration (browser, JSP, servlets, database) with frameworks (e.g., Spring[1]) and patterns (e.g., MVC). The topic of architecting in agile projects was not addressed in our courses and students were not given instructions on how to approach it. We observed two types of student behavior with respect to the architecting activities:

- Most groups embraced the implicit nature of architecting in Scrum and had a short discussion on their architecture at the beginning of the project. Often, architectural decisions were made implicitly, leading to surprises within teams. Typically, the students did not discuss quality attributes. If the PO would take the initiative to state desired quality attributes, the students would accept them, but would not address them in their architecture design and implementation. At the end of the project, they would typically not remember them. Documenting took place at the latest possible moment and mainly to satisfy the course requirements that mandated an architecture document at the end of the course. The students elaborated primarily low-level designs (a class model, a database model, and occasionally a component model) influenced by their way of working in PTS3.
- Few groups followed a Big Design Up Front (BDUF) approach. These groups applied their knowledge on architecting from PTS3, embarking on a prolonged design process, documenting heavily upfront, and colliding with the expectation of the PO to deliver working software in each sprint.

In both scenarios, the students did not perceive any value in performing architecting activities. The lack of clear guidelines on how to approach architecting activities resulted in unpredictable process and in subjective grading of students' software architecting skills. Typically, the tutor would grade the architecture documentation instead of the architecting process, in a clear contradiction to the agile principles. In our desired situation, the teachers and students would have a clear approach to architecting in agile software development projects and valid and reliable criteria for its grading.

2.3. Approaches to software architectures in education

Teaching software architectures is discussed in the literature in two contexts, i.e., as a stand-alone course on software architectures and as a software development course where software architecting is applied (our context). Next, we discuss literature addressing these two contexts of teaching software architecting.

Kiwelekar and Wankhede discuss the learning objectives for courses on software architectures (Kiwelekar and Wankhede, 2015). Mannisto et al. discuss challenges which university educators face when teaching software architectures (Mannisto et al., 2008). As main challenges they identify the isolated nature of academic courses and the lack of realistic architecting context (systems are designed from scratch, stakeholders do not exist, cases lack fuzziness and complexity inherent in industrial scenarios). Furthermore, students do not have experience in solving complex design problems and profound knowledge for the application domain. The authors describe an advanced, master-level course on software architectures in which architecting is a team effort in a complex environment. The architecting team reports to a "Board" (formed by staff members) which considers the designs that are proposed and makes architectural choices. Galster and Angelov discuss the challenges in teaching software architectures using an adapted semiotic triangle (Galster and Angelov, 2016). They study challenges related to the semantics of the term software architecture, to its representation, and its application in practice form the perspective of learners. Garlan et al. present a master's course in which the analysis and evaluation of software architectures are central topics (Garlan et al., 1992). Lago and Van Vliet classify the courses on software architectures as either focusing on the "tools" to design software systems (patterns, languages) or focusing on communication aspects of software architecting (Lago and Van Vliet, 2005). They discuss their experiences from two master's courses which are focusing on the communication aspects of software architecting. The objectives of the courses are to teach selection and development of architecture views and performance of architecture assessment. The courses follow an architecture-centric approach in which stakeholders are involved in the architecting process and the functional and non-functional requirements are simultaneously addressed during the design process.

Applying software architecting in development projects has received less attention. Cleland-Huang et al. present a course on software architecting in agile projects (Cleland-Huang et al., 2014a, 2014b). The authors propose a "design upfront" approach where architecture design and evaluation takes place using architecturally savvy personas to facilitate the analysis and prioritization of architecturally significant requirements. The approach provides directions on architecting activities at the initial stages of agile projects but does not pay attention to architecting activities in the rest of the project. Lee et al. described an agile, software development, project course (Lee et al., 2015). Architecting activities take place in week 3 of the course. However, the architecting issues in agile projects are not extensively discussed.

2.4. Research approach

There are currently few publications on applying software architecting in agile software development courses. The approaches in these publications make specific choices (use of personas) and do not provide a complete method (Cleland-Huang et al., 2014a) or do not provide sufficient information about the architecting activities (Lee et al., 2015). Therefore, we study general literature on approaches to architecting in agile projects. Based on the results, we define our approach to software architecting in PTS4. We evaluate the approach by applying it in two consecutive semesters.

3. Research context

In this section, we present our findings from an informal literature study on the interplay between software architectures and agile methods. Internet, keyword search in Google Scholar was applied as a primary step for identification of relevant sources. On the basis of titles and abstracts, an initial set of articles was selected as relevant and the articles were reviewed. Next, snowballing technique was applied for the retrieval of new relevant sources (following references from a paper to find other relevant papers). We applied snowballing again for the new sources that we have identified. As result, we have studied in total 55 peer-reviewed articles on the interplay between agile and architecture (articles published in publication venues as IEEE Software, IEEE Computer, conference proceedings published by IEEE, LNCS, and ACM and periodicals, i.e., Information and Software Technology, Journal of Software Systems) and 1 technical report. We refer to 31 of these articles in Sections 3.2–3.7. In addition, we have studied

[1] http://projects.spring.io/spring-framework/.

20 articles and books on the semantics of the term "software architecture". We refer to 8 of them in our discussion in Section 3.1.

Systematic literature reviews outperform informal literature reviews in most aspects except in the quantity of identified articles (Niazi, 2015). Therefore, we have compared our findings with a formal systematic mapping study on architecting in agile published after our research has been completed (Yang et al., 2016). Yang et al. study 54 articles of which 14 articles appear also in our study. Our study is performed from an educational perspective, where the core architecting activities and issues play role. We have omitted advanced architecting topics (e.g., articles on architectural impact analysis in agile). Therefore, the overlap between the two studies is reasonable.

Abrahamsson et al. identify seven factors that need attention when architecting in agile (Abrahamsson et al., 2010). The factors are: semantics of the term software architecting, context of projects, timing of architecting activities, role of architects, documentation of architectures, method for architecting, and value of architecting. We use these factors to structure our findings in this section as well as to define our approach in Section 4. Next, we discuss our findings.

3.1. Semantics of the term "software architecture"

The term "software architecture" is attributed various definitions. The choice of a definition influences the approach to architecting in agile. Buschmann and Henney state that a software architecture "can mean anything from a high-level design sketch with little relationship to technology, code, or the actual system being built to a big, rigid up-front design with a lot of class and code-level minutiae" (Buschmann and Henney, 2013). They note that in its core an architecture "is a service to guide and develop a software system toward fulfilment of a set of business and technical objectives, expressed in a form that best helps communication and sharing". Some definitions focus on the structure aspect of architectures. For example, according to Bas et al. a software architecture is "the set of structures needed to reason about the system, which comprise software elements, relation among them, and properties of both" (Bass et al., 2012). Other definitions view design decision as the focus of software architectures (Jansen and Bosch, 2005). Lago and Van Vliet state that a software architecture reflects "the major design decisions made" (Lago and Van Vliet, 2005) and de Boer and van Vliet that "the actual structure, or architectural design is merely a reflection of those design decisions" (de Boer and van Vliet, 2009). Fowler sees the major design decisions as the decisions that are hard to change (Fowler, 2003). According to Zimmermann, architectural decisions "capture key design issues and the rationale behind chosen solutions" (Zimmermann, 2011). However, Bass et al. argue that in agile projects some decisions are made later on throughout the project, and it is hard to justify whether a decision is major (Bass et al., 2012). Abrahamsson et al. note that architectural decisions should not be mixed with software design decisions (Abrahamsson et al., 2010).

3.2. Project context

The project context motivates the necessity of performing architecting activities explicitly and in an organized manner (Kruchten, 2010; Davide, 2010). A project context that stimulates architecting has certain complexity (Mannisto et al., 2008). A system should involve several quality attributes and its design should offer room for taking non-trivial architectural decisions in order to make architecting activities meaningful in an agile project (Kruchten, 2010; Nord and Tomayko, 2006). Waterman et al. conclude that system complexity rather than system size is a leading factor in determining the need for explicit architecting activities in agile projects (Waterman et al., 2013). Furthermore, Boehm notes that system requirements should be relatively predictable (Boehm, 2002). Hindered communications or coordination (e.g., team distribution) is another contextual factor that motivates architecting (Abrahamsson et al., 2010).

3.3. When to architect

Abrahamsson et al. advise on focusing on architecting "early enough" (Abrahamsson et al., 2010) as decisions taken at this stage are hardest to change later in the project. Furthermore, they recommend addressing stories with architectural aspects in the early iterations of a project. Similarly, Cleland-Huang et al. state that "it is particularly important to get architectural design correct upfront" (Cleland-Huang et al., 2014a). Nord and Tomayko recommend focus on the overall system structure (shaped by the quality attributes) in the first sprint and subsequently in later sprints if changes are needed (Nord and Tomayko, 2006). Conversely, Jan Bosch favors emergent architectures. In his opinion, the community is switching to a "user-driven architecture work" from the traditional "quality based architecting" (Mirakhorli and Cleland-Huang, 2013). Friedrichsen examines the concept of emergent architecture and concludes that an explicit architecting in the start of a project is an irreplaceable step in agile projects and that emergent architecting is a complimentary step for the creation of detailed designs (Friedrichsen, 2014).

3.4. The architect role

Similar to the architecting activities, the architect role is not explicitly discussed in most agile approaches. According to Babar, the role of architects is vital for the successful combination of agile and architecture (Babar, 2014). Architects need to support agile teams by removing impediments through architecting rather than creating obstacles (Buschmann and Henney, 2013). They have to act as service providers and facilitators for the agile teams (Faber, 2010). Their tasks, responsibilities, and expected qualities are discussed in Babar et al. (2009) and Babar (2014). Architects can be positioned inside or outside the agile team (Rost et al., 2015). A study among 10 professionals identified two types of architects in agile projects: one with management functions (solution architect), and one responsible for the actual implementation in the team (implementation architect) (Babar, 2009). A high-level architecture is drafted by the solution architect and design decisions and a concrete architecture by the implementation architect. Abrahamsson et al., inspired by Fowler (2003), suggest similar roles naming them "architectus reloadus" and "architectus oryzus" (Abrahamsson et al., 2010). Lindvall et al. report on the positive effect of having an architecting team in large-scale agile projects (Lindvall et al., 2002).

3.5. Representation and documentation of software architectures

Architecture documentation 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" (Tyree and Akerman, 2005). Coram and Bohner discuss the key values of documenting software architectures in agile projects (Coram and Bohner, 2005). Architecture documentation may serve to: get on board project members, for future usage (e.g., to transfer a project to another team), for quality assurance, or as domain knowledge repository. However, documenting of frequently changing issues can be a wasted effort. Jan Bosch is pragmatic about documentations (Mirakhorli and Cleland-Huang, 2013). He finds "slide presen-

Table 1
Literature sources and situational factors influencing our approach. Factors expected to be affected negatively are indicated with a "–" sign.

Architecting factors	Literature sources	Affected situational factors
Semantics of SA	Lago and Van Vliet (2005), Jansen and Bosch (2005), de Boer and van Vliet (2009) and Bass et al. (2012)	Realism, Maturity(-), Clarity(-)
Contextual factors	Boehm (2002), Nord and Tomayko (2006), Mannisto et al. (2008), Kruchten (2010), Abrahamsson et al. (2010) and Waterman et al. (2013)	Capabilities, Degree of challenge, Purposefulness, Realism
When to architect	Kruchten (2010) and Cleland-Huang et al. (2014a)	Realism(-)
The architect role	Beck (1999), Schwaber and Beedle (2002), Babar (2009), Cohn (2009), Abrahamsson et al. (2010), Faber (2010) and Friedrichsen (2014)	Capability, Degree of challenge
Documenting	Clements et al. (2003), Tyree and Akerman (2005), Coram and Bohner (2005), Babar (2009), Kruchten (2010), Clerc et al. (2010) and van der Ven and Bosch (2014)	Purposefulness, Realism
Method to SA	Nord and Tomayko (2006), Madison (2010), Friedrichsen (2014), Cleland-Huang et al. (2014a), Friedrichsen (2014), Babar (2014) and Yang et al. (2016)	Maturity, Purposefulness, Realism
Value of SA	Abrahamsson et al. (2010)	Purposefulness

tations capturing the high-level structure and rationale" and automatically generated views of software an efficient and effective way of documenting in agile projects. Boehm notes that for unpredictable projects decreasing the documentation effort may be acceptable, while for projects with predictable requirements this approach may have a negative impact (Boehm, 2002). Kruchten considers all possibilities for architecture documentation, i.e., documentation in code, with the use of metaphors, diagrams, or a complete architecture document (Kruchten, 2010). 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" (Tyree and Akerman, 2005). Lightweight alternatives for documentation, like wikis or photos are acceptable in agile projects (van der Ven and Bosch, 2014). For example, a study performed in a company revealed that documentation is reduced to component diagrams and design decisions published on a wiki (Babar, 2009). Clements et al. propose an approach to documenting software architectures in agile methods (Clements et al., 2003). In their approach, the views to be documented and the project stakeholders interested in them are identified first. Documenting is done on a need-to basis but it is advised to document rationale early and throughout the project.

3.6. Methods to architect

Nord and Tomayko present an overview of software architecting methods and their implementation in agile projects (Nord and Tomayko, 2006). Embedding the existing methods for software architecting in agile projects has been in the focus of Breivold et al. (2010) and Sharifloo et al. (2008). Babar et al. map architecting activities to common agile activities and suggest relevant places for the architecting activities within agile methods (Babar et al., 2009). However, existing works are theoretical and lack operational details. As Yang et al. conclude in their study: "There is a lack of guidance of using agile practices with architecture" (Yang et al., 2016). Two steps in software architecting methods are discussed in literature as important in the context of agile projects but receiving little attention in practice, i.e., the identification of non-functional requirements (NFR) and performance of architecture evaluation. In general, NFR are often neglected and poorly documented (Ameller et al., 2012). Cleland-Huang et al. state that there has been little attention to NFR in agile projects, but that NFR are important in agile projects as a way for decreasing ineffective and costly efforts (Cleland-Huang et al., 2014b). Nord and Tomayko also state that the focus on NFR can lead to improvements in projects executed with agile methods (Nord and Tomayko, 2006). Buchgeher and Weinreich focus on the architecture evaluation activity in agile projects (Buchgeher and Weinreich, 2014). They conclude that the architecture evaluation in agile projects needs to be continuous to match the continuous evolvement of the architecture. However, there are no evidences for the role of architecture evaluations in agile projects and the degree of its implementation. It is suggested that a lightweight approach would fit in an agile environment, e.g., TARA (Woods, 2011). In Eloranta and Koskimies (2014), the use of DCAR (Heesch et al., 2014) is proposed.

3.7. Value and costs of architecting in agile projects

Abrahamsson et al. 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 artefacts in agile development" (Abrahamsson et al., 2010). A study among IBM professionals reveals that architecting activities are highly valuable for agile developers (Davide, 2010). Communication, documentation of assumptions, and validation are supported through architectures. Given the educational context of our work, we do not further focus on the cost aspects of architecting.

4. Course design

In this section, we first present our course design decisions from the perspective of the seven factors influencing architecting in agile projects. Our decisions are based on our literature findings (Section 3) and on our situational factors (Section 2.1). We summarize in Table 1 the literature sources and the situational factors that influenced our choices. As a next step, we explain our grading scheme. At the end of the section, we discuss one of our choices which requires further research.

4.1. Course decisions for the term "software architecture"

We decided to provide to students multiple definitions on software architectures. The definitions reflect the view on architectures as "the set of structures needed to reason about the system" (Bass et al., 2012) as well as the view on architectures as "the major design decisions made" (Lago and Van Vliet, 2005; Jansen and Bosch, 2005; de Boer and van Vliet, 2009). By providing different definitions used in practice, we aimed at increasing the *realism* of our course. However, junior developers (students) have difficulties in reflecting on architectural decisions due to their lack of experience and knowledge (Tofan et al., 2013; Mannisto et al., 2008). Therefore, the introduction of definitions focusing also on the design decisions in an architecture can negatively impact our *maturity* situational factor. To tackle this, we provide examples for architectural decisions and reasoning behind them. Furthermore, the decision may negatively impact the *clarity* situational factor as students may be confused by different definitions for the same term.

4.2. Course decisions for project context

We decided to offer four cases to students from which they are allowed to choose the case that motivates them best. Each case offers sufficient complexity to make architecting activities meaningful in an agile project (Kruchten, 2010; Nord and Tomayko, 2006; Waterman et al., 2013). Furthermore, although cases can evolve throughout projects, their initial description ensures relatively predictable requirements (Boehm, 2002). Cases 1 and 2 differ in their architectural complexity (room for architectural decisions, number of quality attributes, number of features, need for non-trivial architectural decisions), aiming at students with different *capabilities* and preferred *degree of challenge*. Cases 3 and 4 are comparable to Case 1 in terms of functional and architectural complexity. However, Case 3 introduces a distributed team setup which stimulates architecting in agile projects (Abrahamsson et al., 2010). With this case we aim at increasing the *purposefulness* of the architecting activities. Case 4 offers higher *realism* and aims at increasing the student motivation. The cases are:

- Case 1: The PhotoStore case (PS) remains as a case with low architectural complexity (explained in Section 2.2).
- Case 2: The Crisis Information Management System (CIMS) is a case with high architectural complexity. It is about a system supporting communications (direct and asynchronous information sharing) between field operators (e.g., firemen, policemen), a central coordination point, and citizens. The case involves multiple quality attributes (e.g., security, usability, performance, reliability, interoperability) and requires multiple architectural decisions. Due to its size and complexity, the case allows (but does not necessitate) multiple groups to work together on it.
- Case 3: In the spring semester, we offer a team distributed project together with students from the Oulu University of Applied Sciences. The case that students select to implement needs to be comparable to Case 1.
- Case 4: We allow a company to offer a project to students. In discussions with the company, we ensure that the case is comparable to Case 1. The company appoints a PO for the project.

4.3. Course decisions for when to architect

Inspired by the approach of Cleland-Huang et al. (2014a) and [illegible]en by our *specific context*, we have decided to dedicate explicit time for the architecture design activities in the beginning of the project as architecting would take place there anyway (Kruchten, 2010). However, in the course BS41, students are taught that most agile methods do not discuss architecting activities and that different approaches are possible, e.g., "emergent architectures". Thus, students could perceive our decision as negatively affecting the *realism* situational factor.

4.4. Course decisions for the architect role

We have decided to introduce an architecture role as suggested in Abrahamsson et al. (2010), Babar (2009), Faber (2010), Cohn (2009) and Schwaber and Beedle (2002). As Friedrichsen [illegible]: "Architectural work should come from the team - either the whole team, or at least some specifically skilled team members" (Friedrichsen, 2014). In our approach, the architect role is assigned to a volunteer student from the team. In the absence of a volunteer, the role can be vested in the whole team, meaning that they are all responsible for the architecting activities but nobody is labelled with the title "architect" (Beck, 1999). This decision is aimed at stimulating the professional development of students who can act in the role of an architect and therefo[illegible]fects the *capability* and *degree of challenge* situational factors.

4.5. Course decisions for the representation and documentation of software architectures

Why to document: Our *specific context* implies that students need to document their architecture in some form. Therefore, we introduced a number of stimuli for documenting architectures aimed at increasing the *purposefulness* situational factor. In Cases 1 and 2, the need for the PO to be able to quickly look-up architectural choices per group (as the PO has to switch between numerous groups) is introduced as a leading stimulus. The PO uses the documentation to look-up targeted quality attributes, high-level architecture and architectural decisions. Thus, our decision is inspired by the "domain knowledge repository" key value of documenting suggested by Coram and Bohner (2005) and "tool for communication to customers and management" key value reported by Tyree and Akerman (2005). Furthermore, in Case 2, we encourage the involvement of two or more groups working on the case together, introducing communication and synchronization challenges (Kruchten, 2010). In Case 3, team distribution is used to stimulate documenting (Kruchten, 2010). We note that in Case 4 the involvement of a real PO may negatively affect documenting if he/she does not see value in it.

What to document: As a minimum, we require documentation of the stakeholders, the NFRs, a high-level architecture, and a rationale for the design decisions made (Tyree and Akerman, 2005). This content is selected as it is of direct value for the PO (Kruchten, 2010) and is therefore realistic requirement (related to the *realism* situational factor). Following the agile principles, the teams may document detailed designs (class diagrams, sequence diagrams, database models, etc.) if they perceive this as needed (related to the *purposefulness* situational factor).

When to document: We have decided to dedicate time in the beginning of the project (week 2) for the documentation of the initial architecture and to stimulate documentation of architectural changes at the end of each sprint (document rationale early and throughout the project (Clements et al., 2003)). The choice is driven by our decision that a PO needs an architecture documentation to prepare for a sprint demo or a sprint planning. Detailed designs that teams decide to document can continuously evolve and students decide themselves on when to update them.

Where to document: We offer the teams to document either in a traditional document or in a wiki page (Babar, 2009; Clerc et al., 2010) which are *realistic* possibilities in industry projects. We advise students to document the NFR within the architecture documentation, as NFR live beyond a sprint.

How to document: We expect from students "ad hoc documentation" (van der Ven and Bosch, 2014). The teams are allowed to document minimalistic in alignment with the agile lean principles (e.g., photos of drawings are accepted), reflecting existing practices (*realism*).

4.6. Course decisions for the architecting method

We visualize the organization of architecting activities in PTS4[2] and the related theory provided in BS41 by means of an informal process diagram in Fig. 1. In our approach, we aim at higher *realism* and *purposefulness* of the activities. In the first two weeks of the project, the students follow a number of steps that guide them through the initial architecture design. These steps are based on the principles of "Just enough up-front design" and "Just enough work" (Yang et al., 2016). As a first step, the students need to elicit the system stakeholders. "Architecture aims to maximize the satisfaction of all involved stakeholder parties. It is not enough to

[2] The set of slides that we use can be found at: https://sites.google.com/site/samuilangelov/pts4.

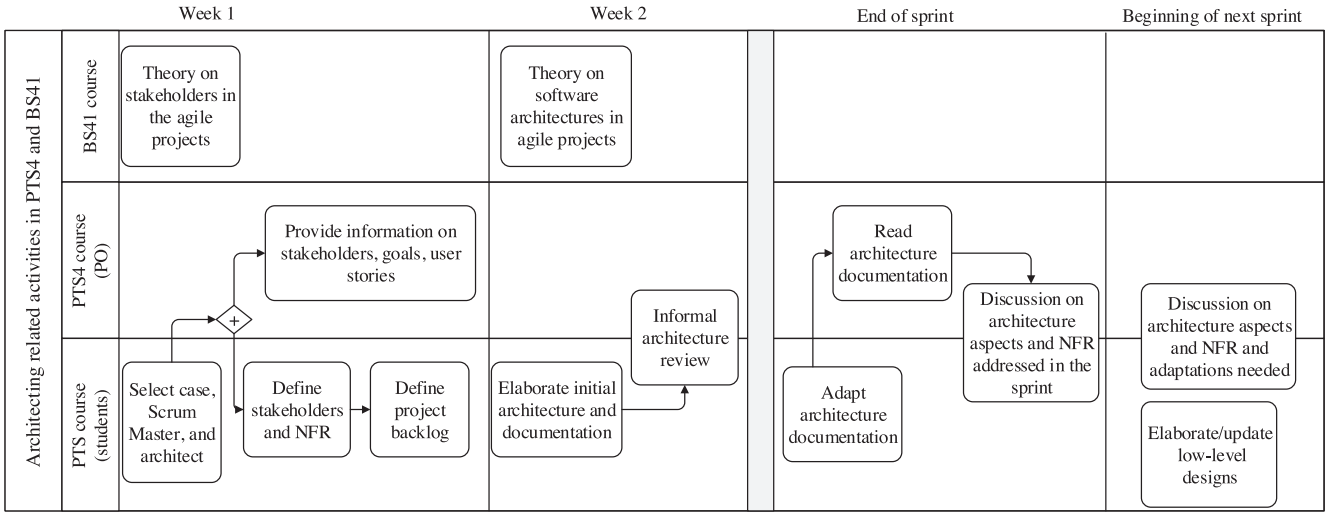

Fig. 1. Architecting activities in PTS4 and theoretical lessons on architecting in agile in BS41.

satisfy a single party – for example, the developers" (Friedrichsen, 2014). We do not follow the approach of Cleland-Huang et al. with architecturally savvy personas for two reasons (Cleland-Huang et al., 2014a). Elaboration of personas takes substantial amount of time which students do not have in the first two weeks as they are busy with plenty of other activities (backlog definition, planning poker, etc.). Providing ready personas to the students is not possible in all our scenarios (e.g., real client project) but most importantly, we find that this will be *unrealistic* and will decrease the learning effect on stakeholders. Our students have no experience with identifying stakeholders and thinking about their goals. Therefore, it is a useful exercise for them to identify the project stakeholders and think about their goals (related to the *maturity* factor).

Next, students have to define desired NFR. We follow the approach of Cleland-Huang et al. (2014a) and Nord and Tomayko (2006) and pay specific attention to NFR in the project beginning. Ignoring NFR may lead to a costly and ineffective effort (Cleland-Huang et al., 2014a) which inevitably would decrease the *motivation* (in terms of *purposefulness* of activities) of students. However, we acknowledge that it may be a useful lesson. To support students in the identification of NFR, we offer as a starting point a structured list of possible high-level goals (Kazman and Bass, 2005) from which students select those applicable to their context. This choice is based on our experiences that without a starting point, the students limit themselves with few, trivial quality attributes (*maturity* factor). Having a list from which to select, helps student in realizing the vastness of stakeholders' goals which may exist and may influence architecture choices.

The architecture is drafted after the project backlog has been defined. With respect to architecture evaluation, we follow an informal "lightweight architecture review" approach (Buchgeher and Weinreich, 2014). We do not introduce a formal review method as it would be time-consuming and introducing yet another method in the project. In our approach, after the initial architecture is drafted, the students need to discuss it with the PO who has to question and provide feedback on it (in order to ensure that the architecture proposed satisfies the stakeholders' goals). At the beginning of each sprint, the teams in a discussion with the PO revisit the software architecture and if needed adapt or extend it (Babar, 2014). At the end of a sprint, the PO and the students reflect on the architecture aspects addressed in the sprint. To reflect the agile nature of projects, students and PO discuss also the NFR at the beginning and end of each sprint (how to address them, what was done to address them). Low-level designs emerge at the beginning or during sprints (Friedrichsen, 2014).

To support the team in managing their workload, we allow to put in sprint backlogs architecting activities (Madison, 2010). 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).

4.7. Course decisions for the value of architecting

Our message to the students is that the core values introduced by architecting (fostering communications, coordination, and analysis) are in the basis for conducting architecting activities in the projects (Abrahamsson et al., 2010). If none of these core values is positively affected by an architecting activity, we do not necessitate its performance. This message aims at ensuring *purposefulness* of the student activities.

4.8. Grading

After each sprint, the students need to provide a demo of their working software, hand-in current architecture documentation, and demonstrate the applied testing strategy and results. The students are graded on these three elements and on the agile process in each sprint. The final grade is formed by the four average grades per element using the formula: 20% agile process, 20% architecting process, 20% testing process, 40% software product. One sprint that contains one or more insufficient grades for the elements may be omitted in the final grading if the failures are rectified in the consecutive sprints. The approach of providing feedback and grading per sprint follows some of the most effective influences for student achievements like formative evaluation and feedback (Hattie, 2008). The grading criteria for the architecture element in PTS4 are:

- Degree of application of the approach.
- Degree of autonomy in the application of the approach.
- Quality (sufficiency, minimalism, correctness) of intermediate and final architecture documentation.

Typically, group members receive the same grade, unless there are substantial differences in their performance. Thus, although the architect role may be vested in one student, all students receive the same grade for the architecting process. This stimulates them to help the architect and stay involved in the architecting process.

Table 2
Context description.

Semester	Spring 2015									Autumn 2015							
Class №	s41			s42			s43			s41				s42			
Group №	1	2	3	4	5	6	7abc	8	9	10	11	12	13	14	15	16	17
Group size	6	6	5	6	6	5	3 × 5	6	5	6	6	5	5	6	6	6	6
Group skills level	high	mid	low	high	mid	high	high	mid	mid	high	mid	low	mid	high	mid	mid	low
Teacher №	1	1	1	1	1	1	2	2	2	1	1	1	1	3	3	3	3
Architects in team	1	1	1	1	1	1	3 × 1	1	1	all	all	1	2	1	1	1	1
Case №	2	1	4	1	1	2	2	1	1	1	1	2	1	2	1	1	2

4.9. Discussion

A fundamental choice in our approach is the introduction of an architect role as part of the team and of a PO as a single point of communication for architects. This choice heavily relies on POs who are knowledgeable about architecting activities, ready to support them, and able to supply the information needed to the architect in the team. The practice is, however, different. POs often lack architecting and development knowledge and are ignorant about non-functional requirements (Angelov et al., 2016; Friedrichsen, 2014). It is very likely that our students may face working with POs who are unable or unwilling to support the architecting activities within the team (Angelov et al., 2016). This means that our setup simulates a nonrealistic context, which contradicts our educational vision (Beer and Angelov, 2015).

One possible way to resolve this challenge may be the introduction of a more realistic PO who is not architecture knowledgeable and even unsupportive to architecting activities. Having an unsupportive PO would demotivate students from performing architecting activities. Furthermore, this approach would heavily undermine the motivation for architecture documentation for teams without other factors motivating documenting. Thus, although realistic, this approach would collide with our *motivation* situational factor. Having an architecturally less knowledgeable (but supportive) PO would require that students "educate" him/her on the role and process of architecting in agile projects (Cohn, 2009). This would be a time-consuming activity in the busy schedule of the students. It might activate students and therefore improve their learning process (Hattie, 2008) but given the lack of experience and complexity of the task, students may easily fail in performing it (colliding with our *capabilities* and *maturity* situational factors). Involving a dedicated tutor to support students in this activity may be a viable approach.

Another possible approach is revealing to students the scenarios that they may face in practice in the form of a theoretical lesson. This would inevitably demonstrate the *limited realism* of our project, potentially decreasing student motivation. Furthermore, a major obstacle for following this approach is the lack of sufficient body of knowledge. Currently, we lack literature describing the architect/PO relationship types, strategies that an architect may follow to resolve problems in complex architectural environments like architecturally unsavvy POs, as well as more general studies on the frequency and types of problems occurring between architects and POs. Initial findings in this direction are provided in Angelov et al. (2016) and Galster et al. (2016).

5. Application of the approach and lessons learned

In this section, we present our experiences from the application of the approach. We discuss which course decisions reached their intended goals or require further attention, as well as the adaptations that we made to the approach based on our experiences. Our aim is to facilitate other educational institutions in the adoption of the approach and to indicate aspects in it that need further research.

We applied our approach in the spring and autumn semesters of 2015. The approach was applied in five classes (see Table 2), divided into 20 groups, each group consisting of 5 or 6 students (where group 7 consisted of 3 subgroups "a", "b" and "c" working on a collaborative project). Three teachers were involved in the experiment. In Table 2, we provide further details on the context of the experiment (case chosen by a group and teacher guiding a group) and the groups' composition (number of students in a group, average skills level of students, and number of students in a group acting as architects).

We discuss the implementation of our approach from the perspective of the course design decisions presented in Section 4. In particular, we pay attention to their influences on our situational factors and whether our goals have been achieved. As input, we used teachers' observations and records on student activities and the artefacts produced by them as well as input from the students. To evaluate the experiences with the approach from the perspective of the students, we set up on-line, anonymous questionnaires. The questionnaire was provided to the students at the end of their projects. In the questionnaire, we focused on the process aspects of our approach as the capabilities of our students to reflect on concepts and abstractions (e.g., definitions, roles) are typically more limited (as discussed in Section 2.1). We asked three questions to the students:

- ***Q1***: *How important do you consider upfront architectural design in agile development processes?*
- ***Q2***: *Does the architecting approach in PTS4 go together with the agile approach?*
- ***Q3***: *Did you improve your understanding of software architecting and value of architecting within PTS4?*

For each question, we provided answer options, ranging from very negative to very positive opinions. In Table 3, we list the questions and the answer options per question. For each answer option, we list the number of votes given to it by a group. Some students did not fill in the questionnaire, which explains the difference between the number of students in a group and the number of responses. We received only 7 votes from groups 7abc because of unfortunate planning for the distribution of the questionnaire to these groups. We discuss the response rate further in Section 5.9. The results from Q1, Q2, and Q3 contribute to our discussion in Sections 5.3, 5.6, and 5.7, respectively.

The data from Table 3 is aggregated in Table 4, where we provide the averages of student answers based on a 1 to 5 point system (starting from 1 point for the most negative answer options). In columns 1 and 2, we provide the averages based on the complete data in the spring and autumn semesters, and in columns 3 and 4, the averages based on the cleansed data for these semesters. Data cleansing involved exclusion of groups where the approach was not optimally implemented. We provide details on the data cleansing in Section 5.9.

Table 3
Complete data from questionnaire among students (numbers in answers show the number of students per group that have selected the answer).

Group №	1	2	3	4	5	6	7abc	8	9	10	11	12	13	14	15	16	17
Q1: How important do you consider upfront architectural design in agile development processes?																	
Not at all	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0
Little	1	1	2	0	0	0	2	0	0	2	1	1	0	0	0	0	1
To some extent	1	3	1	2	2	3	0	2	1	1	2	2	1	3	1	3	1
Quite a lot	3	1	1	2	3	1	4	2	2	2	2	2	1	2	3	3	2
Very significant	1	0	0	1	1	0	0	1	1	0	0	0	1	1	1	0	0
Q2: Does the architecting approach in PTS4 go together with the agile approach?																	
Does not work with agile	0	0	1	0	0	0	3	2	1	0	0	0	0	0	0	0	0
Little	0	0	1	0	0	0	1	1	0	0	0	0	0	0	0	1	0
To some extent	1	1	2	1	1	4	2	1	2	1	0	4	0	0	2	0	3
Quite a lot	1	3	0	1	5	0	1	1	1	3	4	1	1	6	2	5	0
Fully fits with agile	2	1	0	3	0	0	0	0	0	1	1	0	0	0	0	0	0
I cannot judge	2	0	0	0	0	0	0	1	0	0	0	0	3	0	1	0	1
Q3: Did you improve your understanding of software architecting and value of architecting within PTS4?																	
Not at all	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0
Little	0	1	1	1	0	2	1	3	0	1	0	0	0	0	0	1	2
To some extent	0	2	3	1	1	0	4	1	0	1	2	2	2	1	2	3	1
Quite a lot	6	2	0	2	5	2	1	1	3	2	3	3	0	5	2	2	1
Very significant	0	0	0	1	0	0	1	0	1	1	0	0	0	0	1	0	0

Table 4
Averaged data from questionnaire among students.

Semester Data	Spring All data	Autumn All data	Spring Clean data	Autumn Clean data
Q1	3.4	3.5	3.6	3.5
Q2	3.2	3.7	4.1	3.9
Q3	3.4	3.4	3.7	3.6

5.1. Discussion on course decisions for the term "software architecture"

Introducing the students to various definitions has helped in explaining and motivating our choices for the elements expected in an architecture documentation. It has contributed to increasing the *realism* of our approach, as these definitions are followed also by practitioners. However, referring to different definitions introduced *unclarity* to students. The students faced difficulties in answering questions on definitions of software architecture at later stages of the course. This phenomenon is further discussed in Galster and Angelov (2016).

The students did not face many difficulties in articulating their architectural decisions. We conclude that the examples with architectural decisions have helped in tackling the *maturity* situational factor.

5.2. Discussion on course decisions for project context

Ten teams selected Case 1 (PS), eight teams Case 2 (CIMS), one team Case 4 (HUB), and one team executed an internationally distributed project (not listed in Table 2 for reasons explained below).

Case 1: Although light and trivial in its architecting nature, the PS case requires a number of architecting choices that are not obvious for an inexperienced software engineer. Due to its limited complexity, students reflected easier on the non-functional requirements, architectural choices, and architecture adaptations. Its lighter nature allowed us to add ad hoc architecturally significant user stories which led to some architecture adaptations (in the spirit of agile methods).

Case 2: Because of the complexity of the CIMS case, only group 1 managed to implement all high- and medium-priority user stories. The high-level architecture changed slightly only in groups 1 and 6 (in terms of communication protocols) after sprint 1. The project case demotivated architectural changes, as those would further slow development and limit the number of implemented functionalities. Due to the complex nature of the case and multiple desired quality attributes, groups tackled few of them (interoperability, performance).

Case 3: Our intention to apply the approach in internationally distributed project could not be realized. We did not find the time to communicate the approach to the international students due to other factors that needed our attention (i.e., addressing cultural differences and different educational backgrounds). Therefore, the group working on Case 3 is not listed in Table 2.

Case 4: The experiment with a real client (HUB) revealed the heavy dependency of our approach on the PO. The PO was ICT knowledgeable but architecturally unsavvy. He focused on the platform choices but did not articulate desired quality attributes and did not ask about other architectural choices made by the group. Furthermore, he did not require any architectural documentation. Consequently, without having incentives from the PO, the group did not follow strictly our approach and focused on delivering working features to the client. A very limited set of architectural decisions was documented only in the last sprint to satisfy course requirements. This shows either lack of *realism* of our approach in this scenario or lack of sufficient *maturity* in students to deal with the situation. As we were unsure how to address architecturally unsavvy POs, we did not involve a real PO in the autumn semester.

In summary, Cases 1 and 2 allowed students to match their *capabilities* with preferred *degree of challenge* which increased their motivation (reflected in their evaluation of the semester). Cases 4 did not help in increasing the *realism* of our approach and even had a counter effect. The effects of team distribution on the *purposefulness* of architecting could not be evaluated, as the approach was not applied in Case 3.

5.3. Discussion on course decisions for when to architect

As it can be seen from Table 4, Q1, the students value upfront design activities and they did not perceive the approach as *unrealistic* (even being aware that emergent architectures are also a possible approach in agile projects). Moreover, in a follow-up discussion, several students stated that they would be demotivated if they would have to constantly refactor throughout the project. The other students in the class supported this. Thus, although unin-

tended, this course decision had a positive impact on the *purposefulness* of the student activities.

5.4. Discussion on course decisions for the architect role

All groups but two selected a dedicated team architect. In these groups, students took the architect role because of their *capabilities*. While some of them were positively challenged by the role, we noticed also that several students took the role of an architect as an obligation to the team while not being motivated by this.

The introduction of an architect role led to the explicit allocation of responsibility in the group on architecting and served as an additional stimulus for the students to focus on architecting. However, in most projects, it was the PO who would trigger discussions on architectural aspects in the beginning or end of a sprint (see Fig. 1). In order to stimulate students to take the initiative in architecting activities, we have introduced in our grading scheme a criteria on the degree of autonomy in performing architecting activities.

Interestingly, in the case of groups 7abc, the architects intuitively formed an architecting team. The students stated that this had a positive effect on their work, experiencing the positive value of the existing of an architecting team in large-scale agile projects (Lindvall et al., 2002).

5.5. Discussion on course decisions for the representation and documentation of software architectures

Why to document: The needs of the PO and the team were the main driving factor to document in Cases 1 and 2. In these cases, the students agreed that high-level architecture diagrams and decisions were valuable for the PO to prepare for the discussions (i.e., it is a *realistic* requirement) and realized the *purpose* of documenting. However, the architecturally unsavvy PO had a negative effect on documenting in Case 4.

What to document: All teams (except for group 3) followed our approach and documented the stakeholders, their goals and desired quality attributes. In the first execution, the students faced problems in finding proper notation and diagram style to model high-level architectures. 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. Therefore, in the second execution we discussed possible ways to represent it, which has helped students.

In terms of lower level design, all groups decided to document database models which were constantly updated to facilitate team synchronization. Several teams (6 in the spring semester and 4 in the autumn semester) have documented class diagrams and one group has discussed classes but did not document them. Interestingly, in a discussion with these students, most of them indicated that the documentation of the class diagrams had little value for them. They acknowledged that they have documented them influenced by school requirements in PTS3.

When to document: The groups followed the approach and documented the initial high-level architecture in week 2 (except for group 3). Most groups forgot to adapt their documentation of the high-level architecture at the end of sprints and were triggered to do this by the PO. We attribute this to the fact that although *purposeful* for the PO, this activity was not critical for the team. From the low-level designs, only database models were consistently maintained. Few teams maintained their class diagrams.

Where to document: The majority of groups have documented their architecture decisions in a traditional document shared via a cloud storage system. One group used an architecture wiki and one group resorted to photos in combination with diagrams shared via a cloud storage system. Class s42 (autumn semester) was not informed that they may document also in less traditional forms and several students expressed their disappointment from this during the project evaluation. Students stated that they forgot to maintain the architecture documentation because it resided in a separate location. Several students suggested in the future using an architectural wiki in their version control system (GitHub[3]) as a place to document. Alternatively, we think that the architectural documentation may be incorporated in an agile management tool that allows tight integration between project activities and architectural artefacts.

How to document: Generally, the groups working on Case 2 elaborated more detailed, traditional architecture documents. Most groups working on Case 1 had a more pragmatic and minimalistic approach. At present, we are not sure if this is a trend and what may be the reason for it. We noticed that about 30% of the groups documented in excess, violating our goal for *purposeful* documentation. Students stated that they were influenced either by previous experiences (PTS3) or by the belief that more documentation contributes to receiving a higher grade. To remedy this, we have adapted our grading scheme where minimalistic documentation is encouraged and unneeded documentation is discouraged. Furthermore, we observed that students are hesitant to document through photos (only one group made use of this technique).

5.6. Discussion on course decisions for the architecting method

The explicit focus in weeks 1 and 2 on the architecting activities had a 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. The students stated that they realized that stakeholders have different perspectives and may have conflicting goals. The students appreciated and used the initial list of generic stakeholder goals that we provided. About half of the groups, however, had difficulties in prioritizing and focusing on the most important quality attributes. They elaborated long lists of desired quality attributes which the PO had to shorten. As a result of the input and questions from the PO, changes of initial choices took place in about 30% of the groups and resulted in better argumentation of choices already made in another 30% of the groups. 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.

Architectural changes after the initial design were driven by changed/adapted requirements by the PO, by postponed decisions, or simply due to student omissions in the initial design. Only group 6 postponed architectural choices (i.e., push versus pull communication strategy between components) for later sprints where the context would be better known. Unfortunately, after the initial architecting effort, most groups did not take the initiative to revisit their high-level architecture and desired quality attributes, and had to be triggered by the PO (end of sprint and beginning of sprint activities in Fig. 1). Teacher "2", however, did not have sufficient time to take the initiative and trigger students to perform these activities. This had a clear impact on the students' experiences and was reflected in their answers to the questionnaire (see Table 3, groups 7abc, 8, and 9). To tackle this, we have included the degree of autonomy of following the architecting process in our grading scheme. Furthermore, tutors/POs are given more time in their planning so that they can sufficiently guide and support the students in their architecting activities.

[3] https://github.com/.

Table 5
Effects on situational factors by course decisions. Positive influence is indicated with a (+), negative with a (−), positive and negative with (+/−), and unknown with a (?) sign.

Architecting factors	Situational factors
Semantics of SA	Realism(+), Maturity(+), Clarity(−)
Contextual factors	Capabilities(+), Degree of challenge(+), Purposefulness(?), Realism(−)
When to architect	Realism(+), Purposefulness(+)
The architect role	Capability(+), Degree of challenge(+/−)
Documenting	Purposefulness(+/−), Realism(+/−)
Method to SA	Maturity(+), Purposefulness(+/−), Realism(+/−)
Value of SA	Purposefulness(+)

During project evaluations, it came out that students appreciate the approach. As it can be seen from Tables 3 and 4, Q2, they perceive the approach as a rather complementary to the agile way of working. The students who did not perceive it as fully fitting with agile considered the emergent architecture paradigm to be fundamental for agile methods.

5.7. Discussion on course decisions for the value of architecting

We asked Q3 (see Tables 3 and 4) to investigate whether our approach improves the understanding of the students about the value of architecting. The answers show that our approach has helped the students to perceive architecting and its *purpose* from a new perspective and therefore better realize its value. Groups 7abc, in particular, stated that the architecting activities helped them in coming to a common understanding of the system structure and for the work division among the teams.

5.8. Lessons learned

We summarize our findings on the effects of the approach on our situational factors in Table 5. The effects are indicated as positive (+), negative (−), or mixed (+/−). We use a (?) sign when a course decision was not implemented and hence the effect is unknown.

As it can be seen from Table 5, the majority of the situational factors were positively influenced by the approach. Still, the approach is not optimal in terms of *realism* and *purposefulness*. The involvement of an architecturally unsavvy PO in Case 4 demonstrated the heavy dependence of our approach on a PO who is open for and able to support architecting activities of a team (a less *realistic* scenario in industry). The omissions of students to execute architecting activities at sprint planning and sprint completion indicated lack of sufficient *purposefulness* of these activities for students. However, at the end of the projects, students evaluated positively the approach and its *purposefulness*. These seemingly contradicting observations can be explained by the learning process of students and the role of tutors in it. Similar to the substantial coaching that students need to master and comprehend agile practices and ceremonies, they also need coaching in mastering the architecting approach and understanding the purposefulness of the activities in it. The difference in the questionnaire results between groups guided by more and less active tutors further underpins this conclusion (see answer options for groups with teachers “1”, “3” compared to groups with teacher “2” in Table 3). Therefore, tutors should have sufficient time to constantly guide students on both agile and architecting activities.

Another relevant lesson that we learned is that demonstrating the *purposefulness* of architecting in geographically distributed teams is difficult. Students face many challenges in an internationally distributed project which prevent them from focusing on architecting activities. Applying the approach in a national, distributed context may circumvent some of the challenges (e.g., cultural differences, language issues, time zones).

In terms of tool support, it is essential that the tooling used for architecture knowledge sharing is an intrinsic part of the agile tool used by students. Many students forgot to update their architecture documentation because it resided outside their agile project management tool.

5.9. Validity and reliability of results

To strengthen the *reliability* of our findings, we applied the approach on the student population in two consecutive semesters, distributed in 20 groups (see Table 3). Three teachers were involved in the experiment. The approach was documented prior to the experiment and equivalent materials were available to all teachers.

Our findings are based on documented data on the progress on architecting activities and artefacts produced by all groups, as well as data reported by the teachers. All teachers analysed the information collected. The questionnaire was identical in both executions and performed at the end of projects. We sent the questionnaire to 106 students. We collected 86 responses (81% response rate). To strengthen *validity* of our findings, we discussed with students their answers in the questionnaire.

With respect to *internal validity*, a number of variables pose limitations to our conclusions and require attention. Next, we discuss these variables and our reactions to them:

Historical bias: As already discussed, the students were introduced to a waterfall-based approach in PTS3. This previous experience has induced the use of a waterfall approach in group 13 and of a BDUF in group 17. As these groups did not comply with the approach, we have excluded them from the cleansed data in Table 4.

Teacher bias: The perception of the teachers could have influenced the data reported by them in Section 5 (self-reported data). As already mentioned, we based our conclusions mostly on available artefacts (process and architecture documentation, questionnaire results). However, some results in Section 5 are based on data reported by the teachers. Furthermore, we have excluded groups 7abc, 8, and 9 from the cleansed data, as the teacher did not fully follow the approach for time reasons.

Selected case: The introduction of different cases to students influenced the application of the approach (e.g., in Case 3, the approach was not even implemented). We have excluded group 3 (Case 4) from the cleansed data, as our approach was not fully applied there due to the involvement of an architecturally unsavvy PO. Notably, answers from group 3 substantially deviate from other groups (see Table 3, group 3). For groups working on Cases 1 and 2, we provided in Section 5.2 some preliminary observations on the nuances introduced by the cases on the effectiveness of our approach.

Agile theory bias: Within the BS41 course, students are taught that in some agile methods emergent architectures are an accepted way of architecting. As already noted in Section 5.6, this has influenced students in their answers to Q2 of the questionnaire.

Group composition: Current data does not allow any statistically valid conclusions on the correlation between the students' skills and the application of the approach, although such may exist.

Group context: Group projects are often accompanied by problems among students in a group (e.g., conflicts, poor communication). In our experiment, groups 6 and 12 faced problems which required the attention of the tutor. Consequently, the approach received less focus by the tutor and the students. To strengthen the internal validity of the questionnaire, we have removed the results from these groups in Table 4.

With respect to *external validity* of the experiment, the following limitations apply:

Cultural bias: Our target population is Dutch students who are typically independent but often passive learners.

Educational context: The experiment was performed in an applied (professional), bachelor level education, where project courses take place each semester.

Age: Our target population is second year students, who are architecturally rather immature and inexperienced.

The approach might lead to different results when applied to students from other cultures, another type of education, or a more mature target group.

6. Conclusions

We present an approach to introducing software architecting in agile projects in education and share our experience from two consecutive applications of the approach. The strategies selected to introduce architecting activities have proven to be mostly effective. Using our approach, students perceive the value of the architecting activities and see the approach as complementary to agile software development. They appreciate the upfront architecting activity although they perceive it as a slight deviation from agile.

The realism of our approach requires further attention. It has been applied predominantly on architecturally savvy POs, while in practice POs often do not understand or value architecting activities performed in projects. Future work needs to look into the types of problems that architects and agile teams experience when working with architecturally unsavvy POs and the approaches that they can follow to alleviate them. The approach has also to be adapted for the scenario with architecturally unsavvy POs. Another direction for future work is inspired from our conclusion that students need substantial coaching on following the architecting and agile development processes in parallel. One possible future direction for addressing this problem is in creating and providing tooling that allows the integration of the architecture documentation and architecting activities with the agile development process.

Agile methods introduce challenges to applying software architecting in education, but they also offer opportunities to educators for use of more influential learning techniques. The iterative development in agile allows feedback influenced learning and formative assessment of the architecting activities (two of the top-ten most influential learning factors).

Our approach is targeted to second year, applied study, software engineering students. It can be used in higher years with modifications (e.g., in terms of cases and performance of more formal architecture evaluations).

References

Abrahamsson, P., Babar, M.A., Kruchten, P., 2010. Agility and architecture: can they coexist? Softw. IEEE 27, 16–22.

Ameller, D., Ayala, C., Cabot, J., Franch, X., 2012. How do software architects consider non-functional requirements: an exploratory study. In: Requirements Engineering Conference (RE), 2012 20th IEEE International, pp. 41–50.

Angelov, S., de Beer, P., 2015. An approach to software architecting in agile software development projects in education. In: Weyns, D., Mirandola, R., Crnkovic, I. (Eds.), Software Architecture. Springer International Publishing, pp. 157–168.

Angelov, S., Meesters, M., Galster, M., 2016. Architects in scrum: what challenges do they face? In: Software Architecture: 10th European Conference, ECSA 2016. Copenhagen, Denmark. Springer International Publishing, pp. 229–237. November 28-December 2, 2016.

Babar, M.A., 2009. An exploratory study of architectural practices and challenges in using agile software development approaches. In: Software Architecture, 2009 & European Conference on Software Architecture. WICSA/ECSA 2009. Joint Working IEEE/IFIP Conference on, pp. 81–90.

Babar, M.A., 2014. Chapter 1 - making software architecture and agile approaches work together: foundations and approaches. In: Babar, M.A., Brown, A.W., Mistrik, I. (Eds.), Agile Software Architecture. Morgan Kaufmann, Boston, pp. 1–22.

Babar, M.A., Ihme, T., Pikkarainen, M., 2009. An industrial case of exploiting product line architectures in agile software development. In: Proceedings of the 13th International Software Product Line Conference. Carnegie Mellon University, San Francisco, California, USA, pp. 171–179.

Bass, L., Clements, P., Kazman, R., 2012. Software Architecture in Practice, third ed. Addison-Wesley Professional.

Beck, K., 1999. Extreme Programming Explained.

Beer, P.d., Angelov, S., 2015. Fontys ICT, partners in education program: intensifying collaborations between higher education and software industry. In: Proceedings of the 2015 European Conference on Software Architecture Workshops. Dubrovnik, Cavtat, Croatia. ACM, pp. 1–4.

Boehm, B., 2002. Get ready for agile methods, with care. Computer 35, 64–69.

Breivold, H.P., Sundmark, D., Wallin, P., Larsson, S., 2010. What does research say about agile and architecture? In: Software Engineering Advances (ICSEA), 2010 Fifth International Conference on, pp. 32–37.

Buchgeher, G., Weinreich, R., 2014. Chapter 7 - continuous software architecture analysis. In: Babar, M.A., Brown, A.W., Mistrik, I. (Eds.), Agile Software Architecture. Morgan Kaufmann, Boston, pp. 161–188.

Buschmann, F., Henney, K., 2013. Architecture and agility: married, divorced, or just good friends? Software, IEEE 30, 80–82.

Cleland-Huang, J., Babar, M.A., Mirakhorli, M., 2014a. An inverted classroom experience: engaging students in architectural thinking for agile projects. In: Companion Proceedings of the 36th International Conference on Software Engineering. Hyderabad, India. ACM, pp. 364–371.

Cleland-Huang, J., Czauderna, A., Mirakhorli, M., 2014b. Chapter 4 - driving architectural design and preservation from a persona perspective in agile projects. In: Babar, M.A., Brown, A.W., Mistrik, I. (Eds.), Agile Software Architecture. Morgan Kaufmann, Boston, pp. 83–111.

Clements, P., Ivers, J., Little, R., Nord, R., Stafford, J., 2003. Documenting Software Architectures in an Agile World.

Clerc, V., Vries, E.d., Lago, P., 2010. Using wikis to support architectural knowledge management in global software development. In: Proceedings of the 2010 ICSE Workshop on Sharing and Reusing Architectural Knowledge. Cape Town, South Africa. ACM, pp. 37–43.

Cohn, M., 2009. Succeeding with Agile: Software Development Using Scrum. Pearson Education.

Coram, M., Bohner, S., 2005. The impact of agile methods on software project management. In: ECBS '05. 12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, 2005, pp. 363–370.

Davide, F., 2010. In: Giovanni, C., Salvatore Alessandro, S., Giuseppe, C., Paolo, S., Cristiana, D.A. (Eds.), Peaceful Coexistence: Agile Developer Perspectives on Software Architecture, pp. 23–25.

de Boer, R.C., van Vliet, H., 2009. On the similarity between requirements and architecture. J. Syst. Softw. 82, 544–550.

Eloranta, V.-P., Koskimies, K., 2014. Chapter 8 - lightweight architecture knowledge management for agile software development. In: Babar, M.A., Brown, A.W., Mistrik, I. (Eds.), Agile Software Architecture. Morgan Kaufmann, Boston, pp. 189–213.

Faber, R., 2010. Architects as service providers. Softw. IEEE 27, 33–40.

Fink, L.D., 2013. Creating Significant Learning Experiences: An Integrated Approach to Designing College Courses, second ed. Jossey-Bass.

Fowler, M., 2003. Who needs an architect? IEEE Softw. 20, 11–13.

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

Friedrichsen, U., 2014. Chapter 14 - opportunities, threats, and limitations of emergent architecture. In: Babar, M.A., Brown, A.W., Mistrik, I. (Eds.), Agile Software Architecture. Morgan Kaufmann, Boston, pp. 335–355.

Galster, M., Angelov, S., 2016. What makes teaching software architecture difficult? In: Proceedings of the 38th International Conference on Software Engineering Companion. Austin, Texas. ACM, pp. 356–359.

Galster, M., Angelov, S., Meesters, M., Diebold, P., 2016. A multiple case study on the architect's role in scrum. In: Product-Focused Software Process Improvement: 17th International Conference, PROFES 2016. Trondheim, Norway. Springer International Publishing, pp. 432–447. November 22-24, 2016.

Garlan, D., Shaw, M., Okasaki, C., Scott, C., Swonger, R., 1992. Experience with a course on architectures for software systems. In: Sledge, C. (Ed.), Software Engineering Education. Springer, Berlin Heidelberg, pp. 23–43.

Hattie, J., 2008. Visible Learning: A Synthesis of Over 800 Meta-Analyses Relating to Achievement, first ed. Routledge.

Heesch, U.V., Eloranta, V.-P., Avgeriou, P., Koskimies, K., Harrison, N., 2014. Decision–centric architecture reviews. IEEE Softw. 31, 69–76.

Jansen, A., Bosch, J., 2005. Software architecture as a set of architectural design decisions. In: WICSA 2005. 5th Working IEEE/IFIP Conference on Software Architecture, 2005, pp. 109–120.

Kazman, R., Bass, L., 2005. Categorizing Business Goals for Software Architectures.

Kiwelekar, A., Wankhede, H., 2015. Learning objectives for a course on software architecture. In: Weyns, D., Mirandola, R., Crnkovic, I. (Eds.), Software Architecture. Springer International Publishing, pp. 169–180.
Kruchten, P., 2010. Software architecture and agile software development: a clash of two cultures? In: Software Engineering, 2010 ACM/IEEE 32nd International Conference on, pp. 497–498.
Kruchten, P., Obbink, H., Stafford, J., 2006. The past, present, and future for software architecture. IEEE Softw. 23, 22–30.
Lago, P., Van Vliet, H., 2005. Teaching a course on software architecture. In: Software Engineering Education & Training, 18th Conference on, pp. 35–42.
Lee, J., Kotonya, G., Whittle, J., Bull, C., 2015. Software design studio: a practical example. In: 37th International Conference on Software Engineering (ICSE '15). IEEE, pp. 47–55.
Lindvall, M., Basili, V., Boehm, B., Costa, P., Dangle, K., Shull, F., Tesoriero, R., Williams, L., Zelkowitz, M., 2002. Empirical findings in agile methods. In: Wells, D., Williams, L. (Eds.), Extreme Programming and Agile Methods — XP/Agile Universe 2002. Springer, Berlin Heidelberg, pp. 197–207.
Madison, J., 2010. Agile Architecture interactions. Softw. IEEE 27, 41–48.
Mannisto, T., Savolainen, J., Myllarniemi, V., 2008. Teaching software architecture design. In: WICSA 2008. Seventh Working IEEE/IFIP Conference on Software Architecture, 2008, pp. 117–124.
Mirakhorli, M., Cleland-Huang, J., 2013. Traversing the twin peaks. Softw. IEEE 30, 30–36.
Niazi, M., 2015. Do systematic literature reviews outperform informal literature reviews in the software engineering domain? An initial case study. Arab. J. Sci. Eng. 40, 845–855.
Nord, R.L., Tomayko, J.E., 2006. Software architecture-centric methods and agile development. Softw. IEEE 23, 47–53.
Rost, D., Weitzel, B., Naab, M., Lenhart, T., Schmitt, H., 2015. Distilling best practices for agile development from architecture methodology. In: Weyns, D., Mirandola, R., Crnkovic, I. (Eds.), Software Architecture: 9th European Conference, ECSA 2015, Dubrovnik/Cavtat, Croatia, September 7-11, 2015. Proceedings. Cham. Springer International Publishing, pp. 259–267.
Schwaber, K., Beedle, M., 2002. Agile Software Development with Scrum. Prentice Hall.
Sharifloo, A.A., Saffarian, A.S., Shams, F., 2008. Embedding architectural practices into extreme programming. In: ASWEC 2008. 19th Australian Conference on Software Engineering, 2008, pp. 310–319.
Tofan, D., Galster, M., Avgeriou, P., 2013. Difficulty of architectural decisions – a survey with professional architects. In: Drira, K. (Ed.), Software Architecture. Springer, Berlin Heidelberg, pp. 192–199.
Tyree, J., Akerman, A., 2005. Architecture decisions: demystifying architecture. Softw. IEEE 22, 19–27.
van der Ven, J.S., Bosch, J., 2014. Chapter 5 - architecture decisions: who, how, and when?. In: Babar, M.A., Brown, A.W., Mistrik, I. (Eds.) Agile Software Architecture. Morgan Kaufmann, Boston, pp. 113–136.
Waterman, M., Noble, J., Allan, G., 2013. The effect of complexity and value on architecture planning in agile software development. In: Baumeister, H., Weber, B. (Eds.), Agile Processes in Software Engineering and Extreme Programming. Springer, Berlin Heidelberg, pp. 238–252.
Woods, E., 2011. Industrial architectural assessment using TARA. In: Software Architecture (WICSA), 2011 9th Working IEEE/IFIP Conference on, pp. 56–65.
Yang, C., Liang, P., Avgeriou, P., 2016. A systematic mapping study on the combination of software architecture and agile development. J. Syst. Softw. 111, 157–184.
Zimmermann, O., 2011. Architectural decisions as reusable design assets. Softw. IEEE 28, 64–69.