Agile Approaches for Teaching and Learning Software Architecture Design Processes and Methods

Muhammad Aufeef Chauhan, Christian W. Probst and Muhammad Ali Babar

Abstract Software architecture plays a vital role in the analysis, design, evaluation and evolution of large-scale projects. Successful adoption of an agile methodology in large-scale projects requires not only tailoring of the software architecture analysis, design and evaluation methods but also a fundamental understanding of these methods. In this chapter, we provide agile teaching and learning approaches for software architecture analysis, design and evaluation. In particular, we focus on agile teams in architecturally significant (quality) requirements analysis and change management for collocated and distributed agile projects, iterative and continuous architecture design delivery using story boards and collaboration platforms, and using software reference architectures to monitor and control the design and evolution of a software architecture. The methods presented in this chapter are based upon the following research methods. We have explored the literature to identify key characteristics of agile software architecture processes and roles of agile teams in software architecture. We have presented agile teaching and learning approaches with reference to the case studies conducted in classes over 2 years of software architecture courses. We have specifically focused on designing course activities that can support lean education and collaboration among the students and course instructors. We foresee that the presented approaches can be used by academics to teach software architecture design methods and processes in particular, and software engineering techniques in general. Practitioners can also take advantage of the proposed approaches to continuously

M. A. Chauhan (✉)
Netcompany A/S, Copenhagen, Denmark
e-mail: auch@netcompany.com

C. W. Probst
Unitec Institute of Technology, Auckland, New Zealand
e-mail: cprobst@unitec.ac.nz

M. A. Babar
The University of Adelaide,Adelaide, Australia
e-mail: al.babar@adelaide.edu.au

D. Parsons and K. MacCallum (eds.), *Agile and Lean Concepts for Teaching and Learning*, https://doi.org/10.1007/978-981-13-2751-3_16

educate their staff when applying agile methods for architecture design and evolution of complex software systems.

Keywords Agile learning · Software architecture · Software engineering
Architecturally significant requirements (ARSs) · Architecture design
Architecture evaluation · Architecture evolution
Software reference architecture (SRA) · Internet of Things (IoT)

1 Introduction

Software architecture (Gorton, 2006) and agile software development (Hoda, Noble, & Marshall, 2013) have received ample attention from academics and practitioners. However, teaching software architecture in an agile context and learning to apply software architecture analysis, design and development methods for agile software projects remains a challenging undertaking. Adopting agile methods for teaching and learning software architecture requires teamwork, substantial coaching (Babar, Brown, & Mistrik, 2013) and tailoring of the software architecture analysis, design and evaluation methods. To equip the students with comprehensive understanding of the agile methods, it is important to enable them not only to learn in an agile environment but also to provide ample opportunities for the students to apply and practice the agile methods.

Agile software education and learning brings unique challenges with respect to achieving the learning objectives, selection and delivery of the course contents, and participation of the students in agile teaching and learning activities. Agile education objectives have to be in-line with agile software development (Sievi-Korte, Systä, & Hjelsvold, 2015; Angelov & de Beer, 2017) including but not limited to involvement of peers in learning activities, regular meetings, short learning cycles, continuous feedback and measurable artefacts at the end of each learning sprint. Introduction of electronic learning and collaboration platforms such as Moodle[1] and GitHub[2] can support collaboration among the students and course instructors.

Like other aspects of software engineering methods, software architecture practices in agile environment including agile education requires specific considerations. Software architecture describes not only different elements of a software system (such as classes, components and services) and relations among the elements but also encompasses the process and methods that are used for analysis, design, evaluation and evolution of a software architecture (Gorton, 2006). Traditionally, software architecture design processes have been categorized as activities that are carried out in the early phases of the software development life cycle. In the early days of agile software development, software architecture in agile projects was not explicitly focused because the agile process was being used during development of small-scale software systems only (Abrahamsson, Babar, & Kruchten, 2010). However, software architecture has become an important element with adoption of the agile

[1] https://moodle.org/.
[2] https://github.com/.

development methodology in complex projects, for which software development is carried out over a longer period. Software architecture can help to keep the focus on architecturally significant quality requirements can facilitate internal and external coordination, and can keep software development in line with core architecture design decisions (Abrahamsson et al., 2010).

A number of activities and design practices associated with software architecture need to be tailored and enhanced to educate practitioners and students for smooth adoption of software architecting processes in agile projects. Software architecture design and analysis activity encompasses architecturally significant requirement's analysis, identification of a system's elements (in terms of components, services, classes, persistence entities, etc.) and relations among the elements, software architecture evaluation and careful monitoring of an architecture's evolution to avoid software architecture erosion (Chauhan, Babar, & Probst, 2016b; Gorton, 2006). The high-level software architecture design focuses on architecturally significant quality (or non-functional) requirements, whereas detailed design focuses on quality as well as functional requirements. In non-agile software development approaches, high-level architecture analysis and design of the complete system is performed in early phases of a software development life cycle. However, as only features in the current sprint are addressed in agile projects, an overview of the overall architecture can get out of focus. Moreover, the absence of incremental analysis and impact analysis of the new features, which are added in each sprint can negatively impact overall quality of a system in general and architecture quality in particular.

Raising awareness of architecture decisions, as these are made during an agile project, in not trivial. In agile projects, rather than having an upfront detailed architecture design, architecture evolves as a project progresses. As a consequence, architecture design decisions are delayed until corresponding modules are developed. Communication among the team members in the form of face-to-face kick off meetings and daily stand-up meetings is a key characteristic of agile teams (Sievi-Korte et al., 2015). However, such meetings often encompass only a quick overview of the tasks and activities of the team members, and do not include details of architecture design decisions and choices made during day-to-day activities. Agile tools such as storyboards and project back-logs only focus on functional requirements. Architecture often remains invisible in agile projects, which can result in low-quality software in medium and large-scale projects. Hence, it is important to not only teach agile architecting methods to the students but also to teach the methods in an agile learning environment so that the students can practice agile architecture design and analysis methods in an environment that enables agile and lean learning.

To adequately address the afore-mentioned challenges, there is a need to have a specialized approach for teaching and learning software architecture analysis and design methods for agile projects. In particular, we focus on the following objectives in this chapter:

- We discuss the importance of agile and lean education with reference to software engineering in general and software architecture in particular. We discuss different dimensions of agile learning and teaching for software architecture-centric activities, which can facilitate analysis of the new courses and redesign of the existing

courses aiming at teaching agile software architecture analysis and design for students and practitioners.
- We analyse the impact of agility, teams distribution and complexity of the projects on architecture quality, which is elaborated in terms of Architecturally Significant Requirements (ASRs), e.g. security, scalability and reliability. The impact analysis can help to select methods corresponding to the activities that are related to key learning objectives of a course.
- We present agile teaching and learning methods for software architecture analysis, design and evaluation. We elaborate application of the presented methods with reference to case studies that have been conducted in graduate courses with a primary focus on software architecture analysis, design and development. We also discuss in a broader context how the presented methods can be used for teaching and learning software engineering topics. We discuss how the presented methods and case studies of their application can provide a practical guide for adoption of the agile methods for tertiary education.

This chapter is organized as follows. Section 2 provides an insight to the key dimensions of agile learning. Section 3 builds foundation for the presented agile learning approach by describing an analysis of key dimensions of software architecture with reference to agility and how agility impacts activities associated with software architecture analysis, design and evaluation. Section 4 elaborates methodology for agile software architecture education and learning. Section 5 describes application of the proposed agile education and lean learning methodology with reference to a course on software architecture. Section 6 summarizes our experiences of the case studies and discusses applicability of the proposed methods for software engineering education in broader context. Section 7 discusses related work and Sect. 8 concludes this chapter.

2 Key Dimensions of Agile Learning

Different dimensions of agile and lean methods have specific relevance to software engineering education in general and software architecture education in particular. In this section, we provide an overview of the key elements and roles in agile processes, correspondence of the roles to agile education and insight on the structure of a software engineering focused course for incorporating agile and lean learning.

2.1 Key Elements and Roles in an Agile Process

Agility is regarded as ability of an organization, team or a person to adapt and respond to changes in its operating environment to make profit (Abrahamsson et al., 2010). The profit can be a direct reward such as monetary benefit or an indirect reward such as an increase in organizational productivity. Self-organization is one of the key

characteristics of agile teams. Self-organizing teams are formed spontaneously to work on a task, are not part of the formal organizational structure, and have a shared purpose (Hoda et al., 2013). Moreover, while keeping the focus on the team goals, individual team members make decisions for their own tasks (Hoda et al., 2013). Even though teams are self-organizing, the role of mentor in agile teams cannot be underestimated. An agile team member can perform one or more of the following roles: mentor, coordinator, translator, promoter and terminator (Hoda et al., 2013). The ***mentor*** guides and supports the development and team members during initial set-up and encourages self-organization practices during execution of the tasks. The mentor provides not only guidance and support for continuous adherence to the agile principles but also helps to remove misconceptions of agile activities (Hoda et al., 2013). The ***co-ordinator*** manages and coordinates customers expectations with the agile team members, whereas the ***translator*** translates the business language used by the customers to technical terms used by the team members. The ***promoter*** promotes the agile methodology with the customers and the ***terminator*** facilitates the movement of unsuitable team members, who do not conform to the agile development practices, away from an agile team.

In agile teams, one's knowledge is not limited to one's own self. Members of an agile team use and take advantage of the knowledge of other team members as well (Nevo & Chengalur-Smith, 2011). As a result, communication and knowledge sharing among the agile team members play a vital role in agile processes. The capability of a communication medium and the process that is used for communication are also critical to smoothly carry out activities associated with an agile project (Nevo & Chengalur-Smith, 2011). Shared patterns of communication are evolved, and coordinated behaviours are established among the agile team members as they progress and work together on the shared tasks. Communication media can also facilitate awareness of the activities among the team members. Another attribute of the communication medium is to provide a sense of social presence among the team members with as little effort by the team members as possible. Convergence of the communication media facilitates conflict resolution and problem-solving (depending upon the convergence ability of a communication medium). Hence, the stronger the communication attained by a team's members, the better they perform in agile settings.

2.2 Correspondence of Agile Education to Key Agile Elements and Roles

The key elements and roles of agile (as discussed in Sect. 2.1) are also relevant to agile education and lean learning. To enable students to work in agile settings, learning activities are to be carried out in groups of students, which can be analogous to teams in an agile process. Unlike traditional classroom teaching, agile teaching encompasses a number of cohesive and loosely coupled tasks, which signifies the

importance of self-organization among the agile student teams. The teaching staff members can perform the roles of mentor, coordinator, translator and promoter by enabling the students to carry out learning activities in agile settings. As learning activities following agile methods are carried out in groups, communication among the team members and teaching staff, as well as use of communication media for awareness, conflict resolution and problem-solving is important. Some important aspects of agile software engineering education is pair programming, face-to-face meetings or meetings using digital platforms, clearly defined roles and well-defined distribution of work tasks (Sievi-Korte et al., 2015).

2.3 Structure of a Software Engineering Course for Agile and Lean Learning

For enabling the students to learn agile software engineering and agile software architecting, the course should provide a platform where the students can engage in activities characterizing agile software development. These activities include inter-group and intra-group collaboration, face-to-face and digital communication, clarification on the semantics of the course contents, incremental feedback to the students on course assignments and projects, and clear description of inter-group and intra-group activities. Angelov and de Beer (2017) have emphasized the need to explain the following points to the students in an agile learning environment:

- project context and objectives,
- roles that are performed by the stakeholders in real-life scenarios,
- nature and type of the documentation to be used in the projects and
- explicit explanation of the specific methods that are to be used by the students to carry out learning activities in a course.

3 Agility, Software Architecture and Lean Learning

Architectural activities in a project vary widely depending upon the project domain, organizational structure, project size, business model and rate of change in a system under development (Abrahamsson et al., 2010). Activities related to software architecture can be classified generally into four main categories: (i) architecture analysis, (ii) architecture design, (iii) architecture evaluation and (iv) architecture evolution (Gorton, 2006). The architecture of a software intensive system also reflects organizational structure, application domain and mental models of the stakeholders (Coplien & Bjørnvig, 2011). In non-agile and incremental software development models, initial phases of the projects have extensive focus on software architecture including (i) high-level architecture design in terms or architecture styles and patterns and (ii) detailed architecture design in terms of design patterns, frameworks and

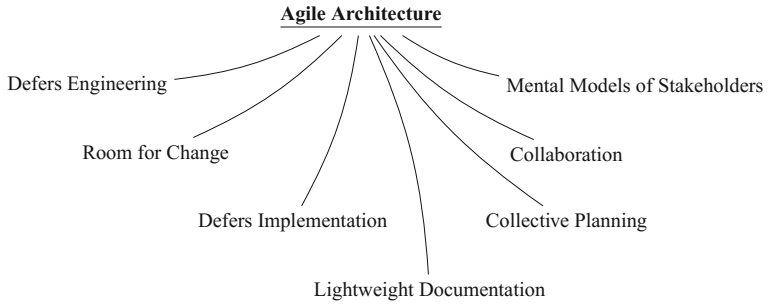

Fig. 1 Agile architecture elements (Coplien & Bjørnvig, 2011)

programming languages, data structures and persistence approaches. Many of the architecture decisions can be hard to undo at later stages without a major re-factoring. Software architects perform the role of a liaison to bridge the gap between business and technical stakeholders, support high-level and detailed architecture design activities, evaluate software architecture designs and monitor architecture evolution to control architecture erosion.

Contrary to non-agile software design, agile software design is characterized by postponing the architecture decisions as late as possible and to provide flexibility to make changes in the design at later stages. Organizational as well as software architectural analysis and design activities for agile development differ from non-agile development in terms of (Coplien & Bjørnvig, 2011):

- when the architecture decisions are made,
- what architecture decisions are documented,
- how an agile project is planned and manage and
- how people are organized in an agile project in terms or roles, responsibilities and task assignments.

Each of the afore-mentioned questions is handled differently in agile architecting. Engineering activities are deferred until the time the related requirements are to be implemented in an agile project. A classic software architecture design tries to limit the changes, whereas an agile software architecture provides room for changes in the architecture while detailed system requirements are analysed and implemented. Software architecture documentation is also limited to key design decisions in agile projects (Abrahamsson et al., 2010). Team members are assigned clearly defined roles during different phases of an agile project, which might be different from that of the organizational hierarchy. Collective planning and cooperation is ensured by the managers to keep a project's activities on track (Coplien & Bjørnvig, 2011). Instead of focusing on technical details such as coupling and cohesion of the detailed architecture elements (e.g. components and classes), an agile architecture focuses more on end users' mental models, which can inspire implementation of the project at later stages (Coplien & Bjørnvig, 2011). The characteristics of agile software architecture process is presented in Fig. 1.

The lean and light weight nature of the agile architecture design practices impose a number of challenges for teaching agile architecture design processes. Educating students about incremental design of comprehensive systems architecture in complex domains such as providing software design and implementation Tools as a Service (TaaS) following pay per use model (Chauhan, Babar, & Sheng, 2015), require interactive learning methods requiring active participation of the students. Especially, when such systems are used as part of the collaborative workspaces and have high degree of dependency upon other systems (Chauhan, Babar, & Sheng, 2017).

3.1 Activities of the Software Architects in Agile Projects

Software architects in agile projects need to focus on traditional as well as agile specific activities. Traditional activities include describing the elements of a software system and relationship between the elements, and establishing the processes and methods that are used for software architecture analysis, design, evaluation and evolution (Chauhan et al., 2017; Gorton, 2006). Agile specific activities include (Coplien & Bjørnvig, 2011):

(a) Focusing on essence of the system rather than functionality.
(b) Defining components and subsystems according to their rate of change as well as functionality.
(c) Identifying components and subsystems so that each of these can be managed as autonomously as possible.
(d) Focusing on locality of the changes.
(e) Prioritizing human considerations while driving the partitioning for software engineering concerns.

The agile specific activities are particularly important to structure the architecture and encompass the activities that are suitable for agile development. Defining components and subsystem based upon the expected rate of change and functionality helps to achieve point b, i.e. to separate parts of the system requiring rapid development and changes from that of more stable areas of the system. Defining modularity based upon the rate of change also help to achieve point c, i.e. autonomous development on each module without having significant impact and interdependency upon other modules. Keeping the domain knowledge of a specific part of the system in the same geographic locations and within the same architecture unit during design and development of a system positively contributes to achieve cohesion. Partitioning of modules following the domain knowledge also achieves cohesion. If the architects do not have a comprehensive understanding of the domain while architecture is being designed, they can use end user cognitive models of the domain to derive a system's modularity (Coplien & Bjørnvig, 2011). Using product lines development approaches for domain partitioning can also facilitate reuse of the system components and domain knowledge to a certain extent (Coplien & Bjørnvig, 2011), i.e. to achieve points d and e.

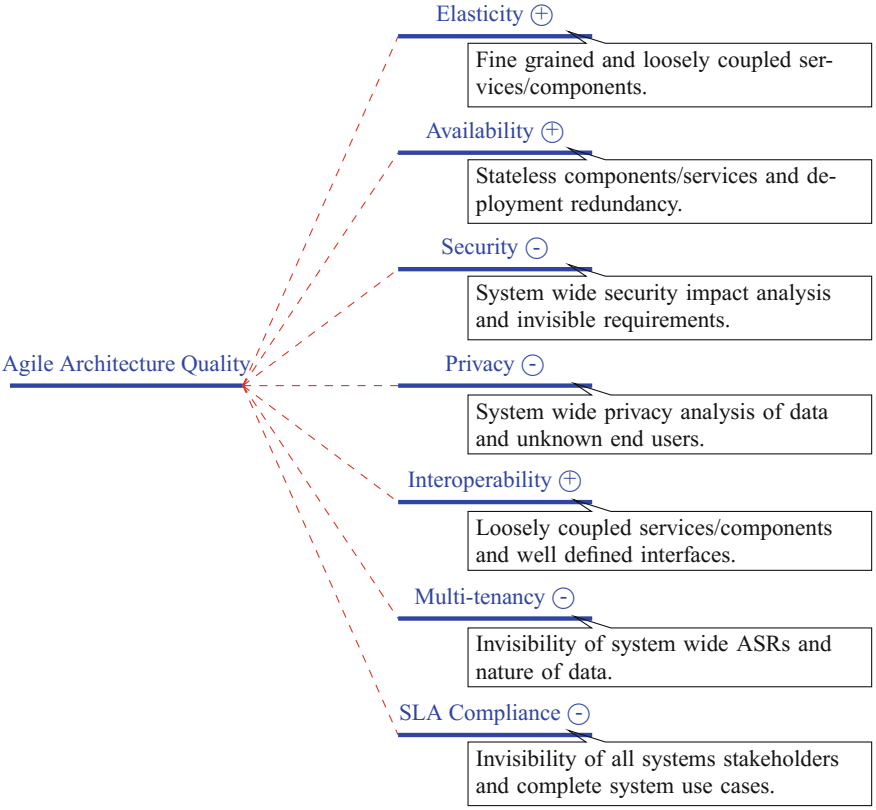

Fig. 2 Impact of agility on software architecture quality

3.2 Impact of Agility on Architecture Quality

Based on the principles of agile software development, agile architecture evolves as different parts of the system are designed and implemented. As software architecture primarily focuses on architecture quality attributes (also referred as architecturally significant or non-functional requirements), some quality requirements are positively impacted by architecture agility, whereas other requirements are negatively impacted by architecture agility. Figure 2 shows a pictorial representation of the relation between architecture quality attributes (Chauhan, Babar, & Benatallah, 2016a) and agility. A description in the box corresponding to each quality attribute explains what influences agility (positively ⊕ or negatively ⊖) for the respective quality attribute. An agile learning approach for software architecture education should enable the students to learn how to focus on desired architecture quality attributes.

3.3 Impact of Distribution of Agile Teams on Software Architecture

Agile teams focus on coordination, collaboration and communication, also referred as three 'C's of agile (Dingsøyr, Dybå, & Moe, 2010). The agile teams can be distributed or co-located. Agile teams need to make architecture decisions within teams and communicate the decisions with other teams. Hence, the three 'C's have a key place in agile architecture development in different phases from elicitation of the architecturally significant requirements to the detailed architecture design. Agile teams need to communicate architecture design decisions that are made locally to other distributed and co-located teams, and need to collaborate with each other to resolve conflicting architecture design decisions. Agile teams' distribution signifies the importance of having architecture control mechanisms as well. The control mechanisms make sure that architecture design choices that are made by each individual team are in line with system-wide software architecture design and quality objectives. Hence, lean software architecture education method should focus on three C's of agile.

3.4 Impact of Complexity and Domain of the Projects on Agile Processes

The complexity of the domain of a specific project also impacts the process that is adopted for agile software architecture analysis, design and evaluation. For complex domains, core design decisions and design artefacts, which are central to the system should be controlled and governed by a core team of architects with domain expertise.

4 Methodology for Agile and Lean Software Architecture Education

A comprehensive methodology for agile and lean learning should encompass all important topics for software architecture and organize the course to incorporate all the key elements of an agile process. The methodology should provide an opportunity to the students so that they can perform software architecture analysis, design and evaluation activities following agile and lean processes.

Incorporating agility in software architecture education requires incremental and iterative delivery of the course contents, and quick feedback on the student's deliverables. Heavyweight software architecture design and evaluation methods need to be tailored to match specific needs of agility in software architecture. In this section, we describe an agile education strategy for software architecture intensive courses. We also describe strategies for software architecture design and evaluation activities,

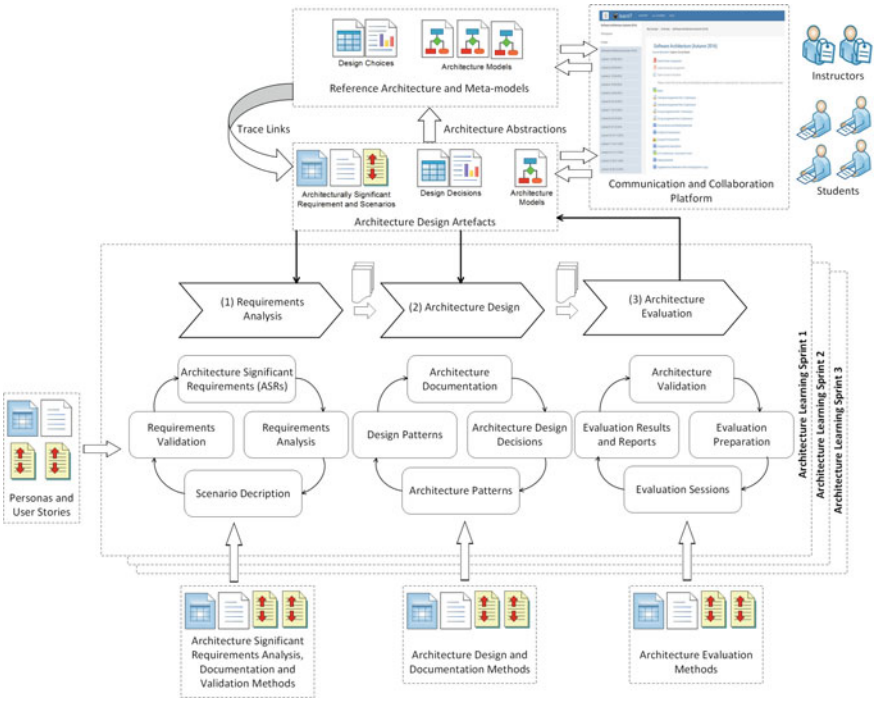

Fig. 3 Agile teaching and learning process for a course on software architecture

which are tailored to maximize engagement of the students in the learning activities and bring the learning experience closer to real-life agile projects.

An aggregated view of the agile architecture education and learning process is presented in Fig. 3. The centre of the figure shows learning activities that can be included in each sprint of the agile software architecture learning process. The students can be taught requirements elicitation, architecture design, and evaluation activities corresponding to the selected set of user stories in each sprint. Related literature (depicted in the figure as an input to the course activities) can be taught to perform the activities in accordance with the schedule of a course. The teaching and learning activities can be organized based upon the complexity of the course topics. Each sprint can have more than one iteration, and each proceeding iteration can have more complex course contents along with exercises as compared to previous iterations.

Electronic communication and collaboration platforms, which can be used by the course instructors and students, are represented at top right of the figure. All the exercises and assignment descriptions, deliverables submitted by the students, feedback on the deliverables by the teaching staff and preparation notes along with results of the evaluation sessions can be managed using the communication and collaboration platform.

In the following subsections, we first describe the key elements of a software architecture course for agile and lean learning. We then provide details of the key elements of the agile and lean learning methodology specific to software architecture education.

4.1 Key Elements of the Agile Software Architecture Course and Students' Activities

Designing software architecture involves: (i) identifying requirements that can have an impact on a software architecture, (ii) detailed design and presentation of the architecture to represent structure and behaviour of a system and (iii) evaluating the candidate architecture solution to make sure that it can comply with the desired functional and quality requirements (Gorton, 2006). For long-term maintainability and evolution of the software, software architecture evolution needs to be monitored and controlled to reduce the risk of architecture erosion (Chauhan et al., 2016b).

Architecturally Significant Requirements Elicitation, Documentation and Management: Elicitation, documentation and management of Architecturally Significant Requirements (ASRs) is a key activity of the architecture design process. It involves identification of architecture savvy personae (Cleland-Huang, 2013) and capturing their user stories for functional and quality aspects of the system. Each persona represents a specific stakeholder of a system. It can be a domain expert, a software developer or an end user. Each persona specifies what is acceptable and what is not acceptable for them from the system under design. The system's expected behaviour is specified in terms of user stories. The user stories can capture both functional and non-functional (also referred as ASRs) requirements. However, for software architecture perspective, only ASRs are important.

The ASRs specified by the users are described in terms of three attributes: (i) stimulus, (ii) response and (iii) response measure (Clements et al., 2002). The process of refining the requirements helps to identify incomplete requirements. For example, if an ASR is associated with availability and it does not provide any metric to measure satisfiability of the requirement (e.g. 24×7), then the requirement is incomplete. The refinement process also helps to identify complementary or conflicting requirements (Chauhan et al., 2016b). For example, if availability ASR states that the system should be available 24×7 and maintainability, ASR states that the system should be down while installing new upgrades, then there is a conflict.

Software Architecture Design: The artefacts produced as a result of software architecture design activity are used as a baseline for software development. The design artefacts can correspond to any of the logical, process, development or deployment view (Kruchten, 1995). These views include different types of design diagrams, e.g. class diagrams are a part of logical view, sequence and collaboration diagrams are a part of process view, component diagrams are a part of development view, and deployment diagrams are a part of physical view (Kruchten, 1995). The primary

activity of the design phase is to incorporate a number of architecture and design patterns to achieve architecture quality, i.e. to satisfy the ASRs.

Software Architecture Evaluation: The purpose of architecture evaluation activity is to make sure that the architecture is compliant to the desired quality, i.e. all the important ASRs are properly transformed into the architecture design documents. A number of software architecture evaluation methods such as Architecture Trade-Off Analysis Method (ATAM) (Kazman et al., 1998), Software Architecture Analysis Method (SAAM) (Kazman, Bass, Webb, & Abowd, 1994) and Quality-Driven Architecture Design and quality Analysis (QADA) (Matinlassi, Niemelä, & Dobrica, 2002) can be adopted based on the requirements of a specific project.

Monitoring and Controlling Software Architecture Evolution: Control over the process of software architecture evolution is important to protect the software architecture from erosion and deviation from the long-term architecture quality objectives. Hence, it is an important course topic. Though it is challenging to measure the impact of change on software architecture, using software reference architecture to capture a blueprint of the architecture and adaptation of semi-formal approaches such as attack-defence trees can help to monitor architecture evolution (Chauhan et al., 2016b).

Different course topics and key elements of each topic are summarized in Table 1.

4.2 Iterative Delivery of the Course Contents Combined with Short Hands-On Exercises

In order to incorporate agility in software architecture education, rather than following a waterfall delivery approach, the course topics should be delivered iteratively. That requires splitting up course contents on the basis of the difficulty level of each topic (e.g. ranging from basic to advanced) and grouping the course contents from the different topics based upon the difficulty level. For example, the process of requirements elicitation covers architecturally significant requirements identification and documentation (in terms of quality to be achieved, stimulus of the requirements, response of the system when the requirement is implemented in the system and metrics for response measure) (Gorton, 2006).

For agile learning, in the first phase of the architecture design sprint, only quality attributes corresponding to the most important quality requirements can be focused on, e.g. scalability or elasticity. Next, architecture scenarios in terms of stimulus, response and response measures can be taught. In the second phase, the quality attribute can be taught and combined with the high-level architecture design tactics and architecture patterns. For example, scalability and elasticity can be achieved using multi-tier architecture and stateless system services. Multi-tier architecture and stateless services facilitate adjustment in the runtime configuration of the system as the system needs to be scaled up or down. In next phase, stimulus, response and response measures can be taught in combination with detailed design tactics

Table 1 Software architecture course contents and elements

Course topic	Element	Description
Personae	Profile	Details on a persona including picture, name, designation, education, professional background and role in the system under development
	User stories (what is expected)	What functional and non-functional (quality) requirements a persona expects to be implemented in the system
	Anti stories (what is not expected)	System behaviour that is not expected by a persona
	Quality attributes of interest	A list of quality attributes that a persona is interested to have in the system
Architecturally Significant Requirements (ASRs)	Quality	A specific quality or a non-functional requirement, e.g. scalability, reliability, security, privacy, etc.
	Scenario	Description of the quality requirement in terms of stimulus (what triggers a quality requirement), response (what should be the system's behaviour satisfying the quality requirement) and response measure (metric to measure the quality response)
Architecture design	Design views	Multiple views of software architecture design including process view, logical view, development view and physical view
	Diagrams in each view	Diagrams corresponding to different views including sequence and activity diagrams for process view; service, class and state diagrams for logical view; component and package diagrams for development view; and deployment diagram for physical view
Evaluation	Software architecture evaluation methods	Evaluating software architecture using methods such as ATAM, SAAM and QADA
Architecture evolution	Controlling architecture evolution	Methods and techniques to monitor and control software architecture evolution

for scalability and elasticity such as cache, facade, redundant system deployment, data synchronization methods, and distributed computing (Buschmann, Henney, & Schmidt, 2007). The whole process can be repeated in the second and third sprints for user stories and quality requirements that are more difficult to achieve. A course can have multiple sprints depending upon the duration of the course and education background of the students.

4.3 Using Digital Platform(s) for Communication, Collaboration and Feedback

Communication, collaboration and self-organization are key practices of agile teams (Campanelli & Parreiras, 2015). When multiple agile teams are working on related or dependent tasks, they frequently need to communicate and collaborate with each other. The same is true for student teams and teaching staff for agile software architecture education. Use of digital platforms for communication, collaboration and feedback on assignments and project deliverables can support frequent interaction and rapid exchange of the artefacts among the student team members, and between student teams and teaching staff. Digital platforms can also facilitate teaching staff to share learning material related to a specific phase of the agile education project and can provide a single point of access for all the relevant course information.

4.4 Incremental Deliverables and Rapid Feedback

A key strategy for incorporating agility in software architecture education is to have iterative deliverables of the students' assignments and projects, following by a rapid feedback on the deliverables. A key step in this regard is to devise a strategy for assignment and project delivery so that output of the preceding phase can be used as an input to the proceeding phase.

For example, Architecturally Significant Requirements (ASRs) (Chauhan et al., 2016a) elicitation and management process can be divided into the following phases. In the first phase, the students can be asked to capture the user stories (Gorton, 2006), i.e. to identify what can be potential users of a given software system and what can be their expectations (in terms of features) from the system. In the second phase, the students can be asked to identify ASRs (quality requirements or non-functional requirements) from the user stories. In the third phase, the students can be asked to capture the details of the ASRs in terms of stimulus (source of the requirement), response (outcome if the requirement is incorporated in the system) and response measure (how to verify if the requirement is implemented correctly) (Clements et al., 2002).

Software architecture can be incrementally designed corresponding to the requirements' elicitation and management phases. After the first and the second phase of requirements elicitation, high-level architecture decisions can be made. After the third phase of requirements management, detailed architecture design decisions can be made such as using model-view-control pattern (Buschmann et al., 2007) to increase modularity and modifiability, or using redundant deployments of the system's services (Buschmann et al., 2007) to increase availability.

4.5 Learning Software Architecture Design

Agile software architecture design encompasses the incremental design of different aspects and views of a software architecture (Kruchten, 1995), which gradually contributes to a complete system architecture. Agile and lean architecture learning can be achieved by combining small class exercises on software architecture design and by complementing the exercises with dedicated architecture design evenings.

The first step for agile architecture design is to use the personae-based approach to capture user stories (as discussed in Sect. 4.1) which can later be used to refined ASRs and architecture scenarios. The personae can be an important input for prioritizing ASRs for different sprints of the agile design process. The personae can be helpful to resolve conflicting ASRs because these can identify the sources of the ASRs.

The following architecture design activities can include transforming ASRs into concrete architecture scenarios, identifying relevant architecture patterns (Buschmann et al., 2007) to satisfy high-level ASRs, selecting design patterns (Shalloway & Trott, 2004) to material detailed architecture design, and selecting appropriate views of $4 + 1$ architecture view model (Kruchten, 1995) that best suit the domain of a system being designed. In the first sprint, primary ASRs, appropriate system view and high-level architecture patterns can be decided. In the second sprint, while the detailed design is elaborated in terms of design patterns and concrete implementation strategies based upon the output from the first sprint, ASRs from the remaining pool of the requirements can be selected.

A key practice for the agile architecture design is to maintain a backlog of all the important decisions made and traceability between ASRs and architecture design decisions (architecture patterns, design patterns and concrete scenarios corresponding to the architecture significant requirements). The scrum meetings can be organized at the beginning of every exercise session, where group members quickly review and analyse ASRs and corresponding architecture scenarios, architecture patterns and design patterns that other members of the group have worked on. The project backlog can be the list of tasks that the students are expected to perform throughout the course. The project backlogs should be shared between a group and teaching staff. In the beginning of the course, the project description and tasks list should be shared with the students.

One of the key characteristics of the agile project is continuous delivery of a product under development. For a semester-based course on software architecture, the course can be divided into two sprints of 5 weeks each. The first sprint can begin after 2 weeks of the first lecture. After each sprint, the students submit a formal project report. Within the sprint, the teaching staff can have semi-formal meetings with each student team to track the progress. Inter-group and intra-group correspondence can be carried out through a digital platforms, e.g. by using Moodle.[3]

[3] https://moodle.org/.

4.6 Learning Agile Software Architecture Evaluation

Software architecture evaluation is a key activity of the software architecting process. Hence, it should also be a part of the agile learning activities in a software architecture course. To make architecture evaluation part of the agile learning process, 3 weeks period can be scheduled between the sprints (considering that a course has two sprints). The output of the first sprint can be used as an input for the evaluation activities and output of the evaluation activities can be used in sprint two for refinements in the architecture design along with incorporation of the new requirements. The number of sprints in a course can be increased depending upon the course duration and contents.

Architecture evaluation sessions should be organized in such a way that members of each group participate in at least two evaluation sessions. In the first evaluation session, the group members can participate to present the key areas of their architecture design and get their architecture design evaluated. In the second session, they can evaluate the architecture of another group and provide feedback on the designed architecture. The architecture of the system that is to be evaluated, should be shared with the evaluation team members in advance (e.g. 1 week before the evaluation session) so that they can have a good understanding of the system design. For example, if group B is to evaluate group A's architecture, group A's architecture should be shared with group B. Evaluation sessions can be organized by following guidelines for any of the architecture evaluation methods [such as ATAM (Kazman et al., 1998), SAAM (Kazman et al., 1994) and QADA (Matinlassi et al., 2002)], depending upon which method suits the project. Each group has to prepare two reports. Before the evaluation session, the group that is evaluating the architecture (group B) has to present a short list of architecture concerns (ASRs, patterns, architecture diagrams, etc.) that they want to focus on during the evaluation session. After the evaluation session, each group that has evaluated an architecture (group B) has to prepare an evaluation report which can be used as an input for the second sprint.

4.7 Learning Agile Software Architecture Evolution

Controlling architecture evolution when multiple teams work on designing related and dependent subsystems in large-scale software systems is not trivial. In agile projects, architecture evolves in each sprint. Therefore, it is critical to monitor and control the architecture evolution to keep it on track and avoid architecture erosion. Reference Architectures (RAs) (Chauhan et al., 2016b) can be a useful tool to monitor and control architecture evolution. Depending upon the maturity of the domain, a RA can be adopted to control evolution in two different ways.

- For mature domains (where one or more comprehensive RAs already exist), a RA that is more closely related to the project scope and domain, can be selected. The selected RA can be used as a starting point to guide the design of the software

system to be developed. For each sprint, trace links of the detailed architecture artefacts (including personae, ASRs, architecture design decisions, architecture patterns, design patterns and different views of the architecture) should be established with the RA and this information should be shared among all the student teams working on the system (in project settings when multiple teams are working on different parts of the same system) and teaching staff.

- For domains in which comprehensive RAs are not available, an abstraction of architecture can be derived from the detailed design after each sprint. This abstraction can contain trace links to the detailed artefacts. This step should be repeated after each sprint.

As a project progresses, the RA can server as a single point of access for all architecture artefacts designed in each sprint. As the new artefacts are added, trace links to the new artefacts are added and previous links are updated if needed. Relations between different types of the artefacts and their impact upon each other can be analysed by the trace links. The RA along with the trace links to the detailed artefacts can be used to identify conflicting and complementing architecture decisions. For example, ASRs can be evaluated by using a probabilistic analysis method to analyse the impact of the ASRs on each other in terms of whether the requirements are dependent on each other, complement each other, or contradict each other (Chauhan & Probst, 2017). The continuous impact analysis allows monitoring of the evolution of the architecture as it matures and protects it from erosion even if the complete architecture is not designed upfront (in one go).

5 Case Studies on Application of the Proposed Methodology

The agile and lean software architecture methodology presented in Sect. 4 was used in a Software Architecture course for a Master's Degree in Software Development Technologies at IT University of Copenhagen (ITU) Denmark[4] in the fall 2015 and 2016 semesters. There were 35 students on average in the class in both years, and 3 course instructors. One of the course instructors was course manager and main lecturer. The other two instructors were responsible for delivery of lectures on specific topics and execution of the exercises. In this section, we describe details on application of the presented approach in the course.

[4] www.itu.dk.

5.1 Course Structure and Distribution of the Roles

The courses consist of 14 weeks of teaching with 1 week of teaching break after the 17th week. In both years, the course was divided into 2 sprints. The first 2 weeks of the course were dedicated to an introduction to basic topics on software architecture, the Unified Modelling Language (UML)[5] and formation of the students working groups. Each group had at least three members. The groups were free to choose the roles within the group (e.g. scrum master and architect team members). The first sprint was organized from week three till week seven. Weeks within the teaching break and the eighth week of teaching were allocated for preparation of the software architecture evaluation sessions (in which each student group evaluated architecture of another group). In addition to the evaluation sessions, the teaching team also evaluated the architecture of each group. The second sprint was organized from week 9 till week 13. In the 14th week, the teaching team evaluated the final architecture report of each group and shared evaluation findings with the respective groups.

The course in both sessions (2015 and 2016) covered the following topics (as described in Sect. 4.1). The students are asked to use UML and Service-oriented architecture Modelling Language (SoaML[6]) to design the architecture diagrams. The students were given the tasks to design the architecture of a part of the Internet of Things (IoT) for smart homes. Only the high-level requirements of the system were shared with the students. The students were asked to choose a part of the IoT smart home system. For example, electricity plus appliance management and control of entrance into the home were the areas chosen by majority of the groups. All members of the teaching staff were actively involved in reviewing the students' intermediate submissions and providing feedback on the submissions.

5.2 Digital Platform Structure

The digital course management platform LearnIT[7] was used in the course for sharing the course material, updates on daily activities, and collaboration among the students and course instructors. Figure 4 shows a screenshot of the course home page on LearnIT. The course page had areas for submission of the students' deliverables (individual and group assignment submission areas) as well as discussion forums (e.g. IoT Architecture Discussion Forum). The course page had sections for each lecture session where details on the learning and exercise materials were shared with the students by the course instructors. There was also a news forum on which teaching staff and the students could post important details on the course.

[5] http://www.uml.org/.

[6] http://www.omg.org/spec/SoaML/About-SoaML/.

[7] https://learnit.itu.dk.

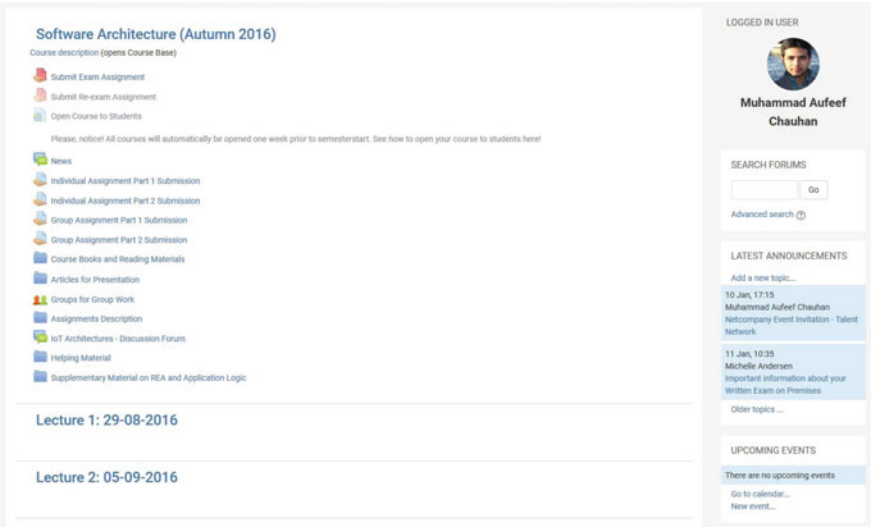

Fig. 4 LearnIT—a digital content sharing and collaboration platform

5.3 Weekly Architecture Analysis and Design Sessions Using Drawing Boards and CASE Tools

The was an exercise session each week during both sprints. A dedicated area was allocated to each student group where they could work on the given tasks. The students used drawing boards and Computer Aided Software Engineering (CASE) tools to design architecture artefacts. The tasks carried out during each exercise session contributed to the detailed architecture design of the respective sprint. In the first sprint, the students were asked to focus on simple ASRs (such as modularity, scalability, etc.) along with functional requirements. In the second sprint, the students were asked to focus on more complex ASRs (such as security, reliability, etc.). In beginning of the first sprint, the groups were asked to define personae for their respective part of the system. As an example, personae identified by one of the groups is shown in Fig. 5. In beginning of the second sprint, the groups were asked to refine the personae for the additional quality requirements. However, during each sprint, the groups were also improving the personae and user stories following the feedback from the teaching staff. After identification of the personae and extraction of the ASRs from the personae, the groups designed the architecture using UML and suitable design plus architecture patterns in each sprint.

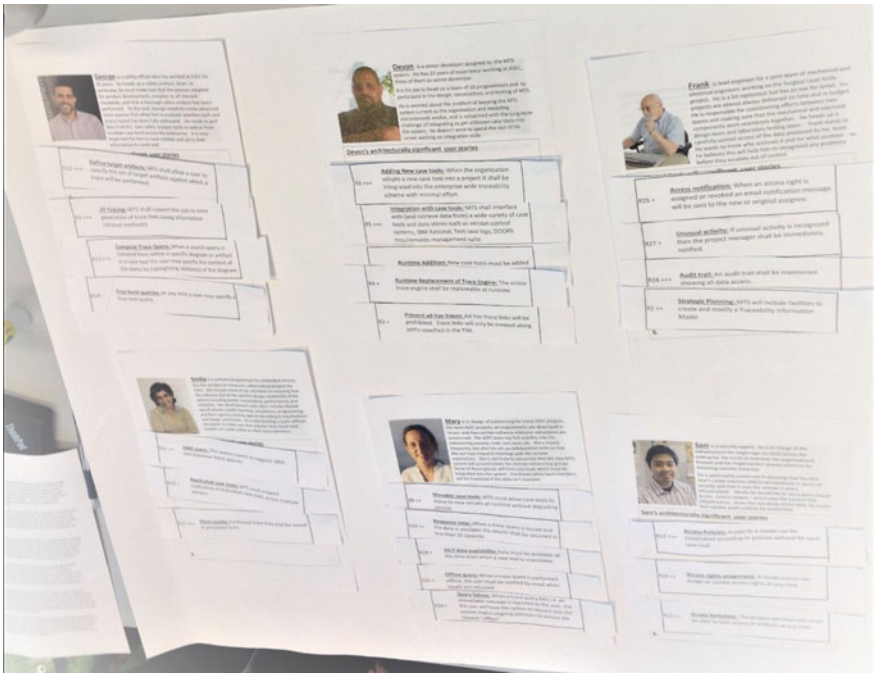

Fig. 5 Example architecture personae generated during weekly exercises

5.4 Deliverable and Feedback Cycles

The students were given a quick feedback on their deliverables at the end of each exercise session. The students were asked to deliver a short informal presentation (of 5–10 mins) to one of the teaching staff members, describing the architecture analysis and design activities they performed along with outcome of the activities. The teaching staff members provided feedback on the produced artefacts. The students were then asked to submit the corrected artefacts in the assignments and projects submission folder as part of the continuous delivery of the architecture design. At the end of each sprint, there was a formal feedback on the final submitted report of the sprint following a separate discussion with each group. The same process was repeated in the second sprint.

5.5 Architecture Design Sessions

In addition to the weekly exercise sessions, there was also a dedicated session on the design in the middle of each sprint for which the groups were asked to make modifications in the design following on the spot input from the teaching staff.

The students worked on the same part of the IoT subsystem during design session as part of their project. The students were asked to use drawing boards, case tools and flip charts so that they could engage in intra-group communication effectively. Design sessions consisted of only intra-group activities plus reviews and feedback from the teaching staff.

5.6 Architecture Evaluation Sessions

Architecture evaluation sessions were scheduled at the end of the first sprint and before the beginning of second sprint. The evaluation sessions were organized so that each group's architecture was evaluated by another group. For this purpose, the architecture design document produced as a result of the first sprint was shared with the group who was going to evaluate the architecture, 1 week prior to the evaluation session. The students were asked to prepare two reports for each evaluation session. (i) A short report (one to two pages) describing their initial analysis of the architecture prior to the session. This report was used as a guide for the students during the evaluation session. (ii) A second report (four to five pages) describing evaluation results of the architecture after the evaluation session. The students were asked to submit the first report before the evaluation session and second report 1 week after the evaluation session. Both of these reports were individual tasks and each student was asked to submit the reports as part of their individual assignments. However, the students were encouraged to collaborate with each other before, during and after the evaluation sessions.

The students were given a rough guideline on organization of the evaluation sessions. Each evaluation session was scheduled for one hour and thirty minutes. The organization of the evaluation sessions is presented in Table 2. The groups were

Table 2 Organization of each evaluation session

Time (min)	Activity	Description
10	Introduction	Members of the architecture evaluation session introduced each other and their core expertise
30	Presentation of the architecture	The group whose architecture design was to be evaluated gave a presentation to the group who was evaluating their architecture
30	Questions and discussion	The group who was evaluating the architecture asked questions and presented their view on different parts of the architecture
20	Debriefing and notes	The groups summarized the findings of the session and took notes to be used in the evaluation reports

allowed to use any of the evaluation methods such as ATAM, SAAM, QADA or their own tailored evaluation approach. However, they were asked to provide the details on the process they followed in their evaluation report. The students were asked to incorporate feedback of the evaluation sessions along with feedback provided by the teaching staff in the second sprint of the architecture design. There was no dedicated evaluation session after the second sprint and the teaching staff evaluated the architecture deliverables submitted by the students.

5.7 Using Architecture Meta-models and a Reference Architecture to Support Architecture Evolution

A key activity to manage agile architecture evolution is to explore the availability of Reference Architecture (RA) and meta-models so that an appropriate strategy to handle traces and variability in the architecture can be adopted (as discussed in Sect. 4.7). Because of the availability of the RA for IoT, the students were given the reference architecture presented by Boussard et al. (2013). The students were asked to follow the high-level architecture description presented in the RA to design the architecture skeleton of a particular area of the smart home system they chose and establish its trace links to the reference architecture abstractions. As all the groups were using the same RA as a baseline, the students were asked to share their design choices for a particular area of smart homes system with reference to the RA on the discussion forum. The course manager and main lecturer also maintained an aggregated enhanced version of the RA for smart homes IoT, based upon abstractions from Boussard et al. RA (Bauer et al., 2013) and by using the students' architecture deliverables at the end of each sprint. The enhanced RA enabled the students to learn how their architecture fits into overall aggregated smart homes system domain.

6 Students Feedback and Discussion on Application of the Proposed Methodology on General Software Engineering Education

Feedback from the students participating in the case studies (described in Sect. 5) reported a positive feedback of the proposed agile and lean software architecture design processes and methods. Table 3 shows average evaluation of the course material and teaching methodology by industrial students from ITUs official Software Architecture course evaluation of 2015 session. The evaluation was conducted via online questionnaire. Each of the students selected a value (between 1 and 6, where 1 is least positive) corresponding to a question. The questions along with average score corresponding to each question is shown in Table 3. The evaluation results show a positive feedback and all the students (in both years) who appeared in the final exam passed the course. Hence, it can be concluded that the adopted methods

Table 3 Feedback on the agile learning approaches from 2015 session

Questions	Feedback score
Overall course satisfaction	5.14
Correlation between course topics and exam requirements	5.00
Relevance for future job	5.14
Satisfaction with course workload and effort	4.86
Selection of learning activities	4.57
Teaching at sufficiently high level	4.71

Lowest possible value = 1, highest possible value = 6, higher is better

were effective in teaching software architecture design to the students. However, it is to be noted that course evaluation was not mandatory and not all the students participated in the course evaluation.

A number of additional factors can also influence adaptation of agile education and learning in software engineering and related disciplines. The main essence of the agile education and learning approach for software engineering is to organize and deliver the course contents iteratively rather than following traditional waterfall lectures delivery approach. Engagement of the students in a learning process where they can practise applications of the agile approaches while doing class exercises and projects is also vital. Therefore, the focus of agile education methodology for software engineering disciplines should be on providing a combination of iterative delivery of the course topics during lectures combined with exercises to enable learning-by-doing.

A generic process for agile and lean learning of software engineering and related disciplines (derived from Fig. 3) is presented in Fig. 6. As the figure shows, a course following an agile learning approach can have multiple iterations on the topics during sprints. At the end of each sprint, the students should provide concrete artefacts that can be evaluated by the teaching staff members and can be used as a baseline for the next sprint. For courses that involve software development, the last iteration of each sprint can include implementation of a part of the software system. Learning material and lectures should be adjusted according to the exercises planned in each sprint. A digital education platform such as Moodle should be used so that rapid exchange of information among the students and teaching staff can take place for continuous collaboration, which is a core characteristic of every agile process. High-level abstractions can be used as a guideline for the group activities. An electronic collaboration platform can facilitate collaboration among the students, as well as between the students and teaching staff, in terms of collaborative work on the artefacts and feedback on the deliverables.

Adoption of agile education and lean learning approaches for software engineering courses can introduce additional challenges. For a course with a large number of students, managing inter-group communication and collaboration can be a challenging undertaking. A large number of students can also put additional overhead on teaching staff for providing continuous feedback on the deliverable as well as on educational institute for providing logistic support to the students so that they can practice agile.

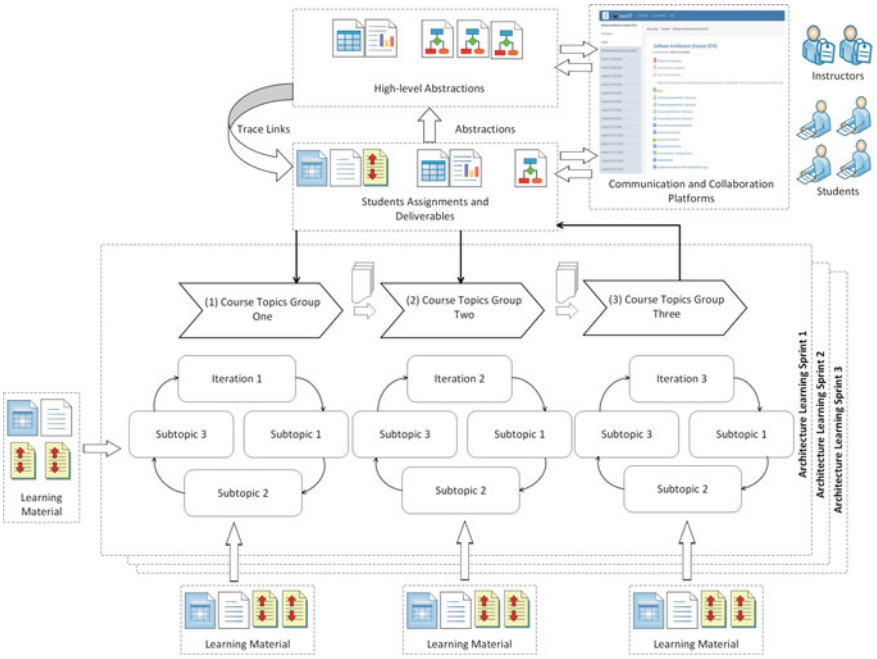

Fig. 6 A generic agile teaching and learning process for software engineering courses

For the courses that are of theoretical nature, agile education approaches might not be applicable for whole length of the course. Specialized courses focusing on specific ASRs including security, privacy, multi-tenancy and service level agreement compliance, might not be the best candidate to adopt agile learning approaches because a rigorous architecture analysis of whole system is needed for such ASRs before implementation can begin.

7 Related Work

The prospects of agile education for software architecture have not been explored much. However, a few studies have focused on agile architecture development in academic and industrial contexts. A case study on software architecting for agile projects in education has been reported in Angelov and de Beer (2017). The authors suggest dividing the achitecting activities into multiple sprints and students to be taught about the basics of agile development and software architecture in the first sprint. After that, the students can design architecture for a selected set of non-functional requirements and can participate in informal architecture reviews in each sprint. Abrahamsson et al. (2010) have discussed software architecture in the context of the agile projects and have suggested that software architecture in agile projects should focus on key

architecturally significant requirements and minimum documentation. Coplien and Bjørnvig (2011) have presented key principles of agile architecture development including deferred engineering and implementation, room for change, lightweight documentation and collaborative planning. Babar (2009) have presented common challenges faced during agile architecture development, which include incorrect prioritization of user stories and lack of focus on important architecturally significant requirements.

We have attempted to address the gaps in the related literature by providing a comprehensive approach for teaching and learning software architecture in an agile manner.

8 Conclusions

In this chapter, we have presented an approach for agile and lean learning. The presented approach has been developed with focus on software architecture education (tertiary level), however, the approach can be applied on other disciplines of software engineering education as well. The presented approach suggests iterative delivery of the course contents and lectures to the students followed by incremental deliverables of students' assignments and projects, and splitting students' exercises and project tasks into multiple sprints. Splitting design exercises into multiple sprints enables the students to get familiar with the agile development process and have hands-on experience with it. A lightweight software architecture evaluation process for educational environment highlights the importance of collaboration among the students for learning design of complex and large-scale systems when multiple teams work on different aspects of the same system.

We foresee that the proposed approach can help academics to align software engineering focused courses with agile practices and facilitate educational institutes to prepare their students for current and future industrial needs.

Acknowledgements We acknowledge the students of software architecture course in 2015 and 2016 at IT University of Copenhagen for their participation in the process and activities described in this chapter. Dr. Chauhan likes to thank his colleagues from Netcompany A/S as well, for providing valuable insight into the software architecture design challenges in agile projects.

References

Abrahamsson, P., Babar, M. A., & Kruchten, P. (2010). Agility and architecture: Can they coexist? *IEEE Software*, *27*(2).

Angelov, S., & de Beer, P. (2017). Designing and applying an approach to software architecting in agile projects in education. *Journal of Systems and Software*, *127*, 78–90.

Babar, M., Brown, A., & Mistrik, I. (2013). Making software architecture and agile approaches work together: Foundations and approaches. Agile software architecture: Aligning agile processes and software architectures.

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

Bauer, M., Boussard, M., Bui, N., De Loof, J., Magerkurth, C., Meissner, S., ..., Walewski, J. W. (2013). IoT reference architecture. In *Enabling Things to Talk* (pp. 163–211). Springer.

Buschmann, F., Henney, K., & Schmidt, D. C. (2007). *Pattern-oriented software architecture, on patterns and pattern languages* (Vol. 5). Wiley.

Campanelli, A. S., & Parreiras, F. S. (2015). Agile methods tailoring—A systematic literature review. *Journal of Systems and Software*, *110*, 85–100.

Chauhan, M. A., Babar, M. A., & Benatallah, B. (2016a). Architecting cloud-enabled systems: A systematic survey of challenges and solutions. *Software: Practice and Experience.*

Chauhan, M. A., Babar, M. A., & Probst, C. W. (2016b). A process framework for designing software reference architectures for providing tools as a service. In *Product-Focused Software Process Improvement: 17th International Conference, PROFES 2016, Trondheim, Norway, 22–24 November 2016, Proceedings 17* (pp. 111–126). Springer.

Chauhan, M. A., Babar, M. A., & Sheng, Q. Z. (2015). A reference architecture for a cloud-based tools as a service workspace. In *2015 IEEE International Conference on Services Computing (SCC)* (pp. 475–482). IEEE.

Chauhan, M. A., Babar, M. A., & Sheng, Q. Z. (2017). A reference architecture for provisioning of tools as a service: Meta-model, ontologies and design elements. *Future Generation Computer Systems*, *69*, 41–65.

Chauhan, M. A. & Probst, C. W. (2017). Architecturally significant requirements identification, classification and change management for multi-tenant cloud-based systems. In *Requirements Engineering for Service and Cloud Computing* (pp. 181–205). Springer.

Cleland-Huang, J. (2013). Meet Elaine: A persona-driven approach to exploring architecturally significant requirements. *IEEE Software*, *30*(4), 18–21.

Clements, P., Garlan, D., Bass, L., Stafford, J., Nord, R., Ivers, J., & Little, R. (2002). *Documenting software architectures: Views and beyond*. Pearson Education.

Coplien, J. O. & Bjørnvig, G. (2011). *Lean architecture: For agile software development*. Wiley.

Dingsøyr, T., Dybå, T., & Moe, N. B. (2010). *Agile software development: Current research and future directions*. Springer Science & Business Media.

Gorton, I. (2006). *Essential software architecture*. Springer Science & Business Media.

Hoda, R., Noble, J., & Marshall, S. (2013). Self-organizing roles on agile software development teams. *IEEE Transactions on Software Engineering*, *39*(3), 422–444.

Kazman, R., Bass, L., Webb, M., & Abowd, G. (1994). SAAM: A method for analyzing the properties of software architectures. In *Proceedings of the 16th International Conference on Software Engineering* (pp. 81–90). IEEE Computer Society Press.

Kazman, R., Klein, M., Barbacci, M., Longstaff, T., Lipson, H., & Carriere, J. (1998). The architecture tradeoff analysis method. In *Fourth IEEE International Conference on Engineering of Complex Computer Systems, 1998. ICECCS'98. Proceedings* (pp. 68–78). IEEE.

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

Matinlassi, M., Niemelä, E., & Dobrica, L. (2002). *Quality-driven architecture design and quality analysis method. A revolutionary initiation approach to a product line architecture.* Espoo: VTT Technical Research Centre of Finland.

Nevo, S., & Chengalur-Smith, I. (2011). Enhancing the performance of software development virtual teams through the use of agile methods: A pilot study. In *2011 44th Hawaii International Conference on System Sciences (HICSS)* (pp. 1–10). IEEE.

Shalloway, A., & Trott, J. R. (2004). *Design patterns explained: A new perspective on object-oriented design*. Pearson Education.

Sievi-Korte, O., Systä, K., & Hjelsvold, R. (2015). Global vs. local—Experiences from a distributed software project course using agile methodologies. In *Frontiers in Education Conference (FIE), 2015. 32614 2015. IEEE* (pp. 1–8). IEEE.