Agile Methodologies in Education: A Review

Pasquale Salza[1], Paolo Musmarra[2], and Filomena Ferrucci[2]

[1]USI Università della Svizzera italiana, Switzerland
pasquale.salza@usi.ch

[2]University of Salerno, Italy
pmusmarra@unisa.it, fferrucci@unisa.it

Abstract One of the main challenges faced by teachers in education, both at K-12 and academy levels, is related to the need to attract and retain the attention and the commitment by students, and ensure they achieve the required learning outcomes. Thus, new and exciting methodologies were developed to support teachers. Many of them have been inspired by approaches devised for Agile software development. This chapter aims to review the main Agile methodologies that have inspired educational approaches and to provide a description of the features retained in the educational context. Several experiences reported in the literature are also described.

Keywords Agile, Education, Review, eXtreme Programming, Scrum

1 Introduction

Agile is one of the most used process frameworks for software development. It is based on some essential values and principles, i.e., the Agile Manifesto, with the aim at lightening the traditional and linear waterfall approach to face the real world in which requirements and solutions evolve continuously (Beck et al., 2001). Agile favors an iterative and team-based approach, attempting to reduce the waste of resources, development time and effort. Several methodologies have been defined within the Agile culture, extending and implementing its values and principles, such as eXtreme Programming, and Scrum.

One of the most relevant implications to managers working using Agile is that it places more emphasis on people factors, focusing on the talents and skills of individuals. If people on a project are good enough, they can use almost any process and accomplish their assignment. Agile makes people work together with excellent communication and interaction, using their individual talents in teams to reach common goals efficiently (Cockburn & Highsmith, 2001).

Agile places less focus on fixed and well detailed a priori plans and is based on the empirical process control theory suggesting that significant knowledge comes from what we learn through experience. An iterative development approach is exploited aiming to deliver product increments that provide value to the customer.

Since Agile methodologies have proved very useful in managing software development teams and projects, the intuitions of many researchers was to adapt them to the educational context (Dewi & Muniandy, 2014). Agile methodologies were first introduced as part of software engineering courses, where the teacher manages student teams making them practice in real software projects (Alfonso & Botia, 2005). Then, Agile methodologies proved to be also effective in teaching other subjects, e.g., Mathematics (Duvall, Hutchings, & Kleckner, 2017).

This chapter aims to give an overview of those Agile methodologies for which some research work, describing the adaptation to the classroom environment, is present in the literature. The aim is to highlight Agile values, principles, and practices applied to improve students' learning. Our review does not aim to be complete and systematic but a starting point for researchers and educators.

The rest of the chapter is organized as follows. Section 2 presents the literature search phase with some statistics on the selected pool of publications. Section 3 introduces Agile, describing the motivations, values, and principles that led it to be one of the most popular software development processes. eXtreme Programming is described in Section 4 and Scrum in Section 5. Each of these sections are organized as follows. The history and key points of the methodology are first provided within the original software development context. Then, the focus is placed on the education environment and the most employed adaptations in the literature are given. Each section ends with an overview of some of the most relevant, or recent, experiences in the literature. Section 6 concludes the chapter with some final remarks.

2 Search Strategy

In this section we present an overview of our research method, the goals of the review and some general statistics about the findings. In terms of timeline, the searches were conducted during late 2017.

Even though this chapter cannot be technically defined as a Systematic Literature Mapping or Systematic Literature Review, we partially developed our search strategy based on well-known guidelines (Kitchenham & Charters, 2007). These were used to organize the research phase and structure the contents of the chapter.

2.1 Search Goals

The purpose of this study is to introduce agile concepts and connect them to the classroom environment, giving a first overview to the readers who are interested in starting to investigate the field.

We identified the following main search goals:

- organize the Agile methodologies with respect to their application in the classroom environment;
- identify the main trends of Agile methodologies for teaching and learning;
- discover what are the main education levels, e.g., K-12, academy, investigated in the literature;
- find out if these methodologies are employed for online, other than on-site education;
- understand if the Agile methodologies for education have gone beyond the software engineering education field to be used to teach and learn other subjects.

2.2 Source Engines and Search Keywords

We employed the main sources of relevance for scientific publications in the field of Computer Science:

- IEEExplore;
- ACM Digital Library;
- ScienceDirect;
- Scopus;
- Google Scholar.

We did not use only the Google Scholar aggregator since several sources have mentioned that "it should not be used alone" as it may miss some sources (Haddaway, Collins, Coughlin, & Kirk, 2015). Indeed, this was something that we easily verified during our search.

We used a query composed by terms related to the Agile context and methodologies, mixed with terms related to the education field. Specifically, our query was expressed by:

"(agile OR scrum OR kanban OR extreme programming OR pair programming) AND (education OR teaching OR learning OR classroom OR K-12)"

Even though it was not systematic, we tried to include all the relevant sources as much as possible using the snowballing technique (Wohlin, 2014). Snowballing, in this context, is based on the use of the reference list or citations to a paper to identify additional papers.

We collected only the papers written in English and published from 2003 to 2017 in international journals and the proceedings of international conferences.

2.3 Selected Papers

From our search, we found a total of about 200 papers. Nevertheless, we did not use all of them to compose this chapter. We only selected those we considered as the most relevant to represent the concepts and goals of our study. In this section, we give some statistics about them.

The majority of the papers, i.e., 80.5%, were published in the proceedings of international conferences. 18% was composed of publications in international journals. The rest consisted of 1 book chapter and 1 technical report.

In terms of date of publication, 36 papers (18%) were published from 2003 to 2009. Starting from 2010, the number increased significantly year by year, from 14 to 27 in 2017 for a total of 164 papers in that range of years (82%).

As for the methodologies, we identified some main categories:

- Agile: 83.5% of the papers refer generically to "Agile" as a methodology for teaching and learning;
- Scrum: 41% of the papers refer specifically to Scrum;
- eXtreme Programming: 11% of the papers refer specifically to eXtreme Programming;
- Pair Programming: 19.5% of the papers refer specifically to Pair Programming. The number is higher than for eXtreme Programming since it includes many works considering "Pair Programming" as an independent methodology;
- Kanban: only 3% of the papers refer specifically to Kanban.

We also divided the papers by education level. Very few of them, i.e., 10%, are papers targeting K-12 students, from a minimum of 4- to 19-year-old (the ranges can change according to different countries). The rest is focused on academy students, where 87.5% is for undergraduates and 18.5% specifically for master students.

The majority of the papers refer to on-site education. However, we found 7 papers that address the adaptation of Agile methodologies within the e-learning context.

Finally, we distinguished between works that use Agile methodologies for teaching and learning specific subjects. As expected, the majority of them, i.e., 91.5%, are about Computer Science and Software Engineering, since it easy to adapt methodologies originally devised for software development by reproducing a subset of concepts for the classroom environment. However, 8.5% refer also to different fields, e.g., Mathematics.

3 Agile

In 2001, a team of 17 leading software practitioners devised and published what they defined the "Manifesto for Agile Software Development". It is a document that defines the values and principles of the Agile software development movement as a summary of how they found "better ways of developing software by doing it and helping others to do it" (Beck et al., 2001).

The key idea is to have an incremental and iterative approach instead of an in-depth planning at the beginning of a software project. Agile methodologies are open to changes in requirements and encourage constant feedback from end users/customers.

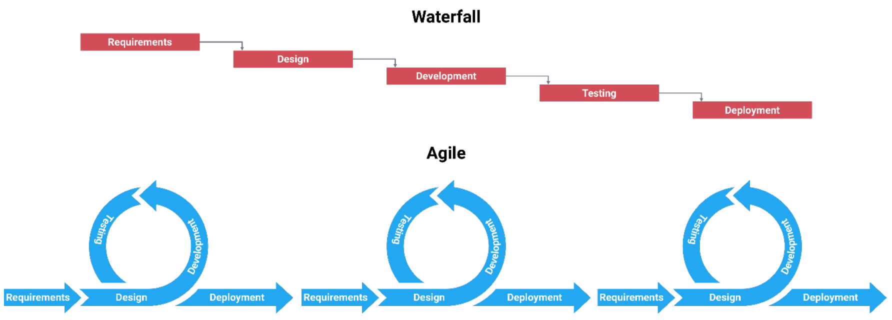

Fig. 1. The waterfall and Agile lifecycles.

In an Agile lifecycle, shown in Figure 1, there is not a strict sequence of events to follow as the classic waterfall model has. The phases are flexible and always evolving, and many of them can even happen in parallel:

1. *requirements analysis:* involves many meetings with the managers, stakeholders, and users to identify the business requirements. The team collects quantifiable, relevant, and detailed information, i.e., who will use the product, and how;
2. *planning:* once the idea becomes viable and feasible, the team splits it into smaller pieces of work, prioritizing and assigning them to different iterations;
3. *design:* the team looks for a solution for the requirements, together with a test strategy;
4. *development:* the features are implemented;
5. *testing:* the produced code is tested against the requirements to make sure the software is solving the customer needs;
6. *deployment:* at this point, the product is delivered to the customers to be used, but this is not the end of the project. It can be only a partial delivery, and new requirements could come.

Agile has given birth to several methodologies, all following its philosophy (Abrahamsson, Salo, Ronkainen, & Warsta, 2002; Dingsøyr, Nerur, Balijepally, & Moe, 2012; Dybå & Dingsøyr, 2008). Table 1 shows the main employed ones.

Table 1. Agile methodologies.

Methodology	Description
Adaptive Software Development (ASD)	It focuses mainly on the problems in developing complex and large systems. The method strongly encourages incremental, iterative development, with constant prototyping. Its aim is to provide a framework with enough guidance to prevent projects from falling into chaos, but not too much, which could suppress emergence and creativity (Abrahamsson et al., 2002).
Crystal Methods	A family of methods for co-located teams of different sizes and criticality: Clear, Yellow, Orange, Red, and Blue. Crystal Clear, the most used one, focuses on communication in small teams developing software that is not life-critical. Clear development has seven characteristics: frequent delivery, reflective improvement, osmotic communication, personal safety, focus, easy access to expert users, and requirements for technical environment (Cockburn, 2004; Dybå & Dingsøyr, 2008).
Dynamic Software Development Method (DSDM)	It divides projects into three phases: pre-project, project life-cycle, and post project. Nine principles are present: user involvement, empowering the project team, frequent delivery, addressing current business needs, iterative and incremental development, allowing reversing changes, high-level scope being fixed before project starts, testing throughout the lifecycle, and efficient and effective communication (Dybå & Dingsøyr, 2008; Stapleton, 2003).
eXtreme Programming (XP)	It focuses on best practices for development: the planning game, small releases, metaphor, simple design, testing, refactoring, Pair Programming, collective ownership, Continuous Integration, 40-hour workweek, on-site customers, and coding standards (Beck, 1999; Dybå & Dingsøyr, 2008).
Feature-Driven Development (FDD)	It combines model-driven and Agile development with emphasis on the initial object model, a division of work in features, and iterative design for each feature. FDD claims to be suitable for the development of critical systems. An iteration of a feature consists of two phases: design and development (Dybå & Dingsøyr, 2008; Palmer & Felsing, 2001).
Kanban	It is a scheduling method developed by Toyota for Lean production. It was designed as a system for scheduling to facilitate the production and inventory control. Using Kanban, work teams can achieve a Just-In-Time (JIT) manufacturing, reducing flow times within production system and response times from suppliers and to customers (Sugimori, Kusunoki, Cho, & Uchikawa, 1977).

Scrum	It focuses on project management in situations where it is difficult to plan, with mechanisms for “empirical process control”. The feedback loops constitute the core element. The software is developed by a self-organizing team in increments called “sprints”, starting with planning and ending with a final review. Then, the product owner decides which backlog items should be developed in the next sprint. Team members coordinate their work in a daily stand-up meeting. One team member, the Scrum master, is in charge of solving problems that might stop the team from work efficiently (Dybå & Dingsøyr, 2008; Schwaber & Beedle, 2002; Schwaber & Sutherland, 2011).

The table shows only a few of the available Agile methodologies. Indeed, industries do not necessarily follow their rules in a strict way and this produced many variants, or hybrids, e.g., Scrumban (Ladas, 2009). In the following sections, an overview of Agile values and principles adaptated to the classroom environment is first given. Then, the focus is shifted to eXtreme Programming and Scrum since they represent the most relevant applications in the field of education.

3.1 Agile in Education

Many researchers had the intuition of tailoring Agile methodologies to the education environment (Dewi & Muniandy, 2014). Stewart et al. presented a first review of the literature aimed at showing how agile methods were applied to education (Stewart, DeCusatis, Kidder, Massi, & Anne, 2009). Moreover, they provided a mapping between values and principles of the Agile Manifesto to specific educational methods and activities.

Table 2 shows the values defined in the Agile Manifesto and the applied mapping, which consists of the translation of software development figures and roles to the education environment (Stewart et al., 2009).

Table 2. Mapping Agile values to the classroom environment (Stewart et al., 2009).

Value	Agile Manifesto	Agile Manifesto in Education
1	Individuals and interactions, over process and tools.	Students over traditional processes and tools.
2	Working software, over comprehensive documentation.	Working projects over comprehensive documentation.
3	Customer collaboration, over contract negotiation.	Student and instructor collaboration over rigid course syllabi.
4	Responding to change, over following a plan.	Responding to feedback rather than following a plan.

As it is shown in the table, the first value favors student-centric environments as the most effective method of learning. In the Agile school, the old-fashioned lecture-driven environment is considered as surpassed. The students participate actively in the learning process through activities and group-based components aiming at reinforcing concepts and allowing for exploration.

As with software in Agile, the second value in education favors the production of working projects from the beginning, without waiting for the end of a project-based course. The students work in an iterative environment with a strong deliverable component, leading to higher immersion in the project, more learning, and final deliverables of better quality.

The third value allows having an environment focused on what the students are doing and which pedagogical methods can facilitate the learning. In traditional courses, the syllabus is outlined as a strict contract between the student and the instructor. Within the Agile school, the students and instructor can establish a more flexible and collaborative relationship, similar to the way that developers and customers collaborate following the Agile approach.

Finally, with the fourth value, agility is applied so that different learning approaches can be adopted and delivery methods changed if the current methods are not producing the expected results.

Table 3 shows the original principles of the Agile Manifesto and the mapping that Stewart et al. applied to the context of the educational environment. In line with the values described above, the 12 principles highlight that the highest priority of Agile instructors is to satisfy the needs of the students, together with their families who become an active part in the learning process, through the early and continuous delivery of meaningful learning. Every change in this direction, even late in the learning cycle, is always welcome.

The students are motivated individuals, working in an environment based on the complete trust from the instructors about getting the job done. Indeed, the old-fashioned face-to-face conversation is assumed superior over technology, even though the latter is considered of undeniable importance and usefulness.

In this context, working deliverables constitute the most tangible measure of student progress. For this reason, meaningful and project-based learning is primarily encouraged with continuous attention to technical excellence and good design but promoting the simplicity of solutions. The students are self-organized in teams, all collaborating with the same effort and reflecting at regular intervals on how to become more effective, tuning and adjusting their behavior accordingly.

Table 3. Mapping Agile principles to the classroom environment (Stewart et al., 2009).

Principle	Agile Manifesto	Agile Manifesto in Education
1	High priority to the customer satisfaction, early and continuously delivering valuable software.	High priority to prepare the student to be self-organized, continuously delivering course components that reflect competence.
2	Requirements can change at any time for the customer's competitive advantage.	The instructor and students can adapt to changes at any time to facilitate learning and better develop marketable skills.
3	Deliver working software frequently with a preference to the shorter timescale.	Working deliverables from the students over short time periods allowing for frequent feedback.
4	Business people and developers must work together daily.	Iterative interaction between the instructor and students (or student groups).
5	Build projects around motivated individuals and support them in a proper environment.	Give students the environment and support necessary to be successful.
6	Prefer face-to-face conversation.	Allow for direct face-to-face interaction with students or student groups.
7	Working software is the primary measure of progress.	Working deliverables (e.g., models, software, project deliverables, presentations) are the primary measure of student progress.
8	The sponsors, developers, and users should be able to maintain a constant pace indefinitely.	The cooperative learning environment is the basis for teaching the skills needed for life-long learning.
9	Continuous attention to technical excellence and good design enhances agility.	Continuous attention to technical excellence and good design enhances learning.
10	Simplicity is essential.	Understanding the problem and solving it simply and clearly is essential.
11	The best architectures, requirements, and designs emerge from self-organizing teams.	Student groups and teams should self-organize, but all should participate equally in the effort.
12	At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.	At regular intervals, the students and instructor reflect and offer feedback on how to be more effective, then the stakeholders adjust accordingly to be more efficient.

Agile has been effectively used in the academy to teach software engineering, mostly with a project-based learning approach where the final learning product is a software produced simulating the Agile practices in small groups (Alfonso & Botia, 2005; Bruegge, Reiss, & Schiller, 2009; Hanks, 2007; Kessler & Dykman, 2007; Lu & DeClue, 2011; Paez, 2017; Rico & Sayani, 2009). As an example, Alfonso and Botia described the use of an iterative and Agile process model in a software engineering course for undergraduate students (Alfonso & Botia, 2005). It served both as an educational technique for teachers and a subject of learning for students.

The model allowed to the expected knowledge to be provided efficiently, acquired gradually, evaluated, and the learning process to be effectively driven.

The use of Agile in K-12 environments is also documented (Fronza, Ioini, & Corral, 2017; Kastl, Kiesmüller, & Romeike, 2016; Meerbaum-Salant & Hazzan, 2010; Romeike & Göttel, 2012). As a relevant example, Fronza et al. described the design and implementation of an educational framework using animations and Scratch to teach computational thinking skills based on Agile practices (Fronza et al., 2017). The framework covers 60 hours, 4 hours per week. The students start from brainstorming and produce storyboards and mindmaps. In each Agile iteration, they draw and program, checking the conformance with the requirements. During the retrospective phase, they analyze the whole project and plan future activities.

Some works report the effectiveness of applying Agile methodologies also in online courses (Ashraf, Shamail, & Rana, 2012; Noguera, Guerrero-Roldán, & Masó, 2018; Vivian, Falkner, & Falkner, 2013). For instance, Noguera et al. proposed an approach for implementing the Agile method into online academic education context (Noguera et al., 2018). The results revealed that Agile strategies incorporated in project-based learning facilitated both the team regulation and project management. The teacher acted as a supervisor and facilitator, helping students improve their learning process iteratively through the development of the projects.

Agile was also employed to teach other subjects. Seman et al. reported a study about applying Agile to two project-based learning courses in electrical engineering (Seman, Hausmann, & Bezerra, 2018). The results showed the importance of the humanization feature in learning, automatically given by applying Agile, as a fundamental part of the education process in electrical engineering.

Finally, the literature reports many cases in which Agile was exploited for its useful tools, not necessarily applying a full methodology. An example is the Kanban board, especially in project-based learning (Ahmad, Liukkunen, & Markkula, 2014; Bacea, Ciupe, & Meza, 2017; Heikkilä, Paasivaara, & Lassenius, 2016). Widely used in industry, Kanban boards are a physical or electronic visualization tool for the management of work in teams, to improve their delivery of products and services in terms of predictability, quality and just-in-time performance. Figure 2 depicts a typical Kanban board, structured on multiple columns to visualize the workflow process.

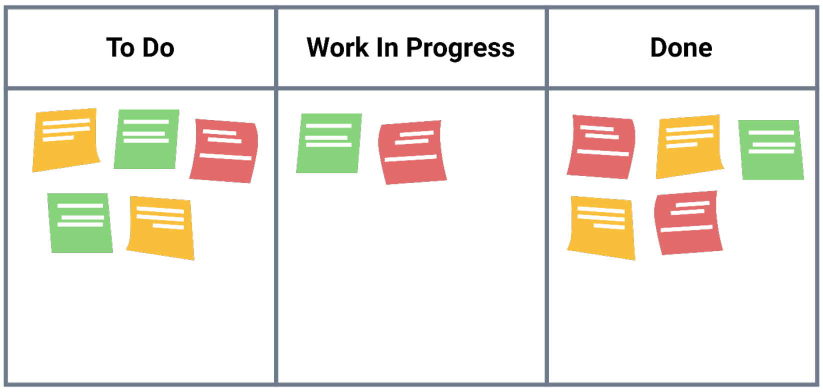

Fig. 3. A typical Kanban board.

Every task in a project, represented by a card, will follow a path:

1. from the "to do" status, where the tasks are discussed within the team and then assigned to specific members;
2. passing to the "in progress" status when the tasks are being executed;
3. ending in "done" when completed.

4 eXtreme Programming

The origin of eXtreme Programming (XP) started in 1996 from Kent Beck, one of the 17 Agile practitioners that signed the Agile Manifesto. At that time, Beck was handling the Chrysler Comprehensive Compensation System (C3) project in the Chrysler Corporation aiming at replacing several payroll applications with a single system. In 1999, he collected all the experience at Chrysler in a book called "eXtreme Programming Explained: Embrace Change" (Beck, 1999).

XP is a development methodology that is intended to improve software quality and responsiveness to change in customer requirements. Figure 3 shows the XP lifecycle. XP allows frequent releases in short development cycles, improving the productivity of the team, and at the end of which new customer requirements can be adopted. XP employs user stories and associates acceptance tests to them that need to be successfully passed before the stories can be considered as done. Moreover, the programmer is expected to write tests for the individual tasks that contribute to a story. Indeed, XP puts tests before code, and each piece of code is expected to be associated with a test, or it should not be integrated.

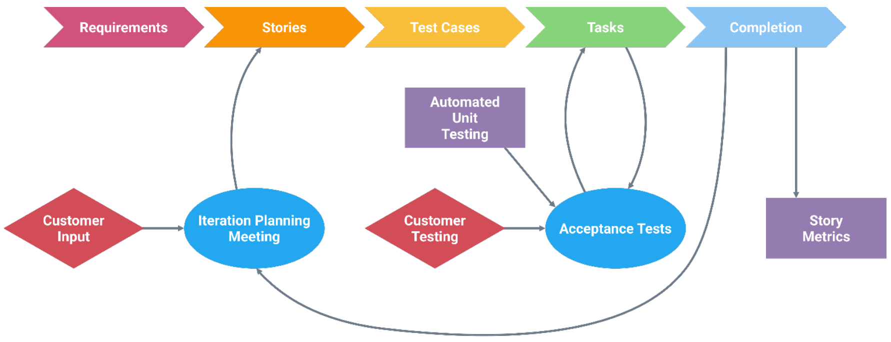

Fig. 3. The eXtreme Programming lifecycle.

XP is based on 5 values:

- *simplicity:* simple solutions are considered as cheaper and faster to be implemented than more complex solutions;
- *communication:* the documentation is always after the direct communication. All the team members should intensively communicate with each other through personal dialogue, aiming at avoiding misunderstandings;
- *feedback:* the customer is interviewed using very short feedback loops. The development progress is frequently shown, and mistakes can be thus avoided. The feedback also comes from the tests;
- *respect:* every member of the team is valuable, even if it is just enthusiasm;
- *courage:* using the XP values requires a great deal of courage. It is important to tell the truth about progress and estimates. Also, XP does not allow fear of refactoring old code or throwing it away if needed for changes.

A typical XP team includes 6 roles:

- *customer:* the person responsible for writing the user stories, setting priorities and specifying the functional tests;
- *programmer:* an ordinary developer who produces the code and performs the whole amount of expected project tasks;
- *coach:* watches and manages the teamwork, teaching its members how to implement the most effective practices;
- *tracker:* the person who monitors the progress of the software development and detects possible issues;
- *tester:* responsible for product testing;
- *doomsayer:* tracks the project risks and warns the team about them.

Although there are a total of 28 rules and practices of XP, they can be compacted into 12 simple rules, many of which originated some of the most used and modern Agile components and methodologies, e.g., Test-Driven Development, Pair Programming and Continuous Integration:

1. *user stories (planning):* a smaller version of use cases. The customer defines as briefly as possible the specification of the new application regarding features, value, and priority. The stories serve as a base to cost and effort estimation of the project;
2. *small releases (building blocks):* each newly added requirements will be instantly incorporated, and the system is re-released;
3. *metaphor (standardized naming schemes):* the developers adhere to standards on names, e.g., class and method names;
4. *collective ownership:* there is not any individual property of the code. All the code can be reviewed and updated by everyone;
5. *coding standard:* developers also adhere to styles and formats of coding to favor the compatibility between team members;
6. *simple design:* the most straightforward implementation of solutions is the most welcome if it meets all the required functionalities;
7. *refactoring:* the application should be continually adjusted and improved by all the team members;
8. *Test-Driven Development:* every small release must pass tests before becoming a new release;
9. *Pair Programming:* the programmers work in pairs in front of a single machine. Essentially, all the code is reviewed as it is written;
10. *Continuous Integration:* the code is continuously integrated and released avoiding the work fragmentation, no more than a few hours;
11. *40-hour workweek:* no one works more than what the body can handle;
12. *on-site customer:* the customer is an integral part of the project. S/he has to be available at all the times to ensure the right track for the project.

4.1 XP in Education

The characteristics of XP also aroused the interest of researchers in education. In particular, Lembo and Vacca proposed an instructional design methodology that exploits XP and project-based learning (Lembo & Vacca, 2012). They found the XP paradigm as particularly well suited for teaching since everything continuously changes in an educational environment. The students are human, and thus their learning response to teaching is never completely predictable, changing with the time and from student to student. The instructional design process is considered as constituted by roles, e.g., the students, teachers, leadership teams, families, each of them performing some activities, e.g., solving problems, personal study, lecturing.

In the same way as the Agile Manifesto in Education (see Table 3), Lembo and Vacca adapted the values of Agile to the educational environment to apply the XP methodology:

1. the collaboration between students and teachers over processes and tools;

2. the collaboration between students, parents, and teachers over educational agreements;
3. exciting activities over instructional design documentation;
4. the design, problem-solving and task performing over notions and knowledge;
5. responding to feedback over following plans.

The highest priority is to satisfy the students and their parents through the continuous production of real projects, and the achievement of results. A collaboration between teachers, students, and parents takes places during each iteration of the project, preferring face-to-face communication. The educational projects must be designed to solve complex real-life problems and be of a short duration, generating at least the level of expected knowledge, skills, and capabilities. Thus, a project must require critical activities, analysis, synthesis, problem-solving to be applied individually or cooperatively and collaboratively. The project proposals are presented in the form of stories and shared with students and parents, who represent the stakeholders.

The literature presents some works regarding the teaching of XP methodology in software engineering courses (Melnik & Maurer, 2003, 2005; Stapel, Lübke, & Knauss, 2008). Stapel et al. presented the experiences and best practices gained in three different XP labs conducted during three years of a software engineering course (Stapel et al., 2008). They organized the labs to practically experience most of the XP practices. Besides being entertaining, XP resulted in a worthwhile experience to improve the overall programming and social skills. Melnik and Maurer conducted a study over five different academic levels of agile practices, in particular using the XP method, showing that all the students accept and like them (Melnik & Maurer, 2005).

Many works involve the practical use of Pair Programming (PP), one of the most influential practices derived from XP (Anderson & Gegg-Harrison, 2012; Lui, Litts, Widman, Walker, & Kafai, 2016; Umapathy & Ritzhaupt, 2017). Other than numerous experiences in computer science, PP was also used to teach other subjects. For instance, Lui et al. reported the findings from an e-textile workshop in which high school students worked in pairs, observing positive results regarding role distribution, and partner communication strategies (Lui et al., 2016). Furthermore, Anderson and Gegg-Harrison explored pair teaching, where the pairing concept is applied to the teachers instead of the students (Anderson & Gegg-Harrison, 2012). Pair teaching increased the interaction between teachers and individual students, improved the quality of course materials, and gave the students two potential role models.

Moreover, Test-Driven Development (TDD) was investigated in the literature. One example is the one involving the use of the Coding Dojo as an environment to teach TDD (Heinonen, Hirvikoski, Luukkainen, & Vihavainen, 2013; Lee, Marepalli, & Yang, 2017; Luz, Neto, & Noronha, 2013). Coding Dojo is a dynamic and collaborative activity where people can practice programming, in particular by applying techniques related to XP practices such as TDD and PP. Even if Coding

Dojos are usually themed by programming challenges, the environment is non-competitive, collaborative, and fun. The participants were able to learn and practice TDD with a relaxed feeling.

5 Scrum

Scrum is one of the most employed process frameworks implementing Agile values and principles. It was created in 1993 and presented to the scientific community in 1995 by Schwaber and Sutherland, two of the authors of the Agile Manifesto. Their work was then collected in a public document named “The Scrum Guide”, trying to define the framework officially (Schwaber & Sutherland, 2011). The term was borrowed from a 1986 article by Takeuchi and Nonaka in the context of product development, published in the Harvard Business Review (Takeuchi & Nonaka, 1986). They found an analogy between high-performing, cross-functional teams and the scrum formation used by rugby teams.

In Scrum, the software is developed by following an iterative model used to manage people and complex projects. Figure 4 shows the Scrum lifecycle. It follows a set of roles, responsibilities, and meeting that never change. With Scrum, the products are built in a series of fixed-length iterations, short and on a regular cadence, defined as “sprints”. During these intervals, teams have the time to develop the software and at the end of each sprint, i.e., the “milestone”, the progress is tangible. By using short iterations, the importance of a good estimation and a fast feedback from tests are reinforced.

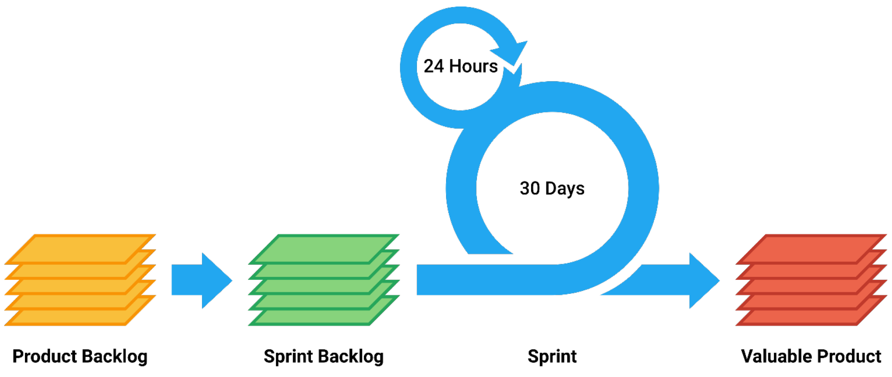

Fig 4. The Scrum lifecycle.

Everyone in Scrum plays a specific role:

- *product owner:* serves as an interface between the development team and its customers. S/he is responsible for checking that the expectations, in the form of a prioritized wish list called “product backlog”, are respected;

- *Scrum master:* a facilitator within the team members. S/he is responsible for ensuring that the Scrum best practices are respected, and the project moves forward;
- *Scrum development team:* comprehends the developers that work together to create and test incremental releases of the product.

Scrum defines 4 events related to a sprint, called "ceremonies":

- *sprint planning:* the team meets and determines what to complete in the coming sprint;
- *daily stand-up (or "daily scrum"):* a 15 minutes short meeting for the team to sync on the project progress;
- *sprint demo:* the team shows what has been produced during sprint;
- *sprint retrospective:* a review of the work done and not done, defining the actions to make the next sprint better.

The development team writes down a list of tasks in what is called the "scrum backlog". These are divided into value-driven "user stories", a quick way of handling customer requirements without having to create formalized documents.

5.1 Scrum in Education

Many researchers worked on ways to adapt Scrum to the educational context. One relevant attempt is given from "eduScrum" (Delhij, van Solingen, & Wijnands, 2015), a guide that translates the Scrum process, roles and responsibilities in pedagogic terms and that can potentially be applied to teach any subject at any education level.

The teacher assumes the role of product owner, who decides what needs to be learned, monitors, processes, and evaluates the students. His/her main goal is delivering the highest value, in terms of both discipline specific learning outcomes and soft skills such as organization, planning, collaboration and teamwork.

The student team is self-organized and aims at acquiring (delivering) learning results iteratively and incrementally. An eduScrum master, who is chosen by the product owner or the class, acts as a coaching leader and helps the team to perform optimally.

Even the sprints are mapped into the education context. The tasks are considered as time-boxed events with a maximum duration and designed to allow critical transparency and inspection. Thus, the sprints are collections of tasks, coherently organized to achieve the learning goals, and usually have a duration of 2 months or less. The expected ceremonies in eduScrum are:

- a planning meeting at the beginning of the sprint, to define team formation, learning goals, and work planning;

- the stand-ups at the beginning of every class, with a duration of 5 minutes to synchronize activities and make plans for the next meeting;
- a review of the past activities of the last sprint, to display what the members have learned;
- a retrospective to create a plan for improvement and preparing for the future sprint.

As for Scrum used to teach software engineering, the literature reports many experiences where it was successfully employed (Bruegge, Krusche, & Wagner, 2012; Mahnic, 2012; Scharf & Koch, 2013; Smith, Cooper, & Longstreet, 2011; Werner, Arcamone, & Ross, 2012; Zorzo, de Ponte, & Lucredio, 2013). Scrum is adapted in the context of the development of academic projects, both in undergraduate and graduate courses. The students are organized into small teams and execute the projects following the rules of Scrum. The instructor usually takes the role of product owner and, for each team, one of the members acts as the Scrum master.

Missiroli et al. presented a case study of software development teaching for K-12 education using Scrum (Missiroli, Russo, & Ciancarini, 2017). They designed an experiment in 7 classes in different schools, assigning the same software project but realized by 2 teams of the same class using different methodologies, i.e., the classic waterfall and Scrum. For Scrum, the product owner is not a peer but the teacher. Considering the young age and experience of participants, the authors suggested striking a compromise between the two methodologies, planning and structure along with creativity and reactivity.

Scrum was also successfully introduced in universities by means of serious games (Fernandes & Sousa, 2010; Lynch et al., 2011; Paasivaara, Heikkilä, Lassenius, & Toivola, 2014; Steghöfer, Burden, Alahyari, & Haneberg, 2017; von Wangenheim, Savi, & Borgatto, 2013). One example is the one related to LEGO-based games (Lynch et al., 2011; Paasivaara et al., 2014; Steghöfer et al., 2017). In the games, student teams learn the Scrum roles, events, and concepts in practice by simulating several development sprints, incrementally planning and building a product of LEGO blocks. Another example is SCRUMIA, which involves a different setup where teams of students are asked to develop different artifacts with the use only of pencil and paper over multiple sprints (von Wangenheim et al., 2013).

Moreover, Scrum was effectively employed as an educational and management method for interdisciplinary education (Cubric, 2013; da Silva Coelho et al., 2014; Gestwicki & McNely, 2016; Ramos et al., 2013). Gestwicky and McNely managed together programmers, artists, and user interface designers to produce 6 different educational game projects (Gestwicki & McNely, 2016). The students came from a variety of degree programs, working in collaboration with one or more faculty mentors and community partners from outside the university.

Finally, Scrum was used to teach other subjects (Duvall et al., 2017; Grimheden, 2013; Mäkiö, Mäkiö-Marusik, & Yablochnikov, 2016; Pinto et al., 2009; Ringert, Rumpe, Schulze, & Wortmann, 2017; Scharff & Verma, 2010). Duvall et al. implemented some Scrum-based classroom management techniques with the aim of getting students to take more responsibility for their learning in a discrete mathematics course at the university (Duvall et al., 2017). The students, divided into teams, enjoyed the self-management and crafting of their learning process. The teams variously chose lecture-based learning, online video-learning, traditional or interactive online textbook reading. Each of them kept a project management progress board so that the professor could track team progress toward self-selected milestones. Besides the independent work, a few traditional lecture times followed, feeling to the students more like group discussions. Furthermore, Grimheden investigated the use of Scrum for the teaching of mechatronics, which is defined as a synergic integration of electronics, mechanical engineering, control, and software engineering (Grimheden, 2013). They showed that Scrum enables the students to deliver results faster, more reliably and with higher quality than other methodologies.

6 Conclusions

In this chapter, the world of Agile methodologies is explored focusing on their adaptation to the education environment. The philosophy behind Agile and its variations, e.g., eXtreme Programming and Scrum, consisting of values, principles, and best practices, is also maintained in the classroom, where the people factor is particularly important.

The reported experiences suggest that Agile can be effective, especially where active and project-based learning can be applied. Not only can Agile be simulated in software engineering courses, but also it can be applied to teach other subjects. Also Agile tools, e.g., Kanban boards, can be part of the learning process.

Applying Agile methodologies to learning and teaching transforms from knowledge transfer to knowledge generated from rich collaboration and experience. Teachers become facilitators, coaches, and inspirational servant leaders for students that are self-directed learners. The focus is not on rigid plans, rather flexibility is required to take into account students' feedback and their different abilities, interests, difficulties, and experiences, aiming at unlocking their hidden strengths and passions. The emphasis is on delivering the highest value, in terms of both discipline specific learning outcomes and soft skills such as organization, planning, collaboration, and teamwork.

This review of the literature shows that there is a growing interest in studying, but even more, applying Agile learning methodologies to allow students to work together in an energetic, targeted, and effective way. There is also an active effort from researchers in formalizing the agile methodologies in the education context, e.g., eduScrum.

References

Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2002). *Agile Software Development Methods: Review and Analysis*. VTT.

Ahmad, M. O., Liukkunen, K., & Markkula, J. (2014). Student Perceptions and Attitudes Towards the Software Factory as a Learning Environment. In *IEEE Global Engineering Education Conference (EDUCON)* (pp. 422–428).

Alfonso, M. I., & Botia, A. (2005). An Iterative and Agile Process Model for Teaching Software Engineering. In *IEEE International Conference on Software Engineering Education and Training (CSEE&T)* (pp. 9–16).

Anderson, N., & Gegg-Harrison, T. (2012). Pair^2 Learning = Pair Programming x Pair Teaching. In *Western Canadian Conference on Computing Education* (pp. 2–6).

Ashraf, M. A., Shamail, S., & Rana, Z. A. (2012). Agile Model Adaptation for E-Learning Students' Final-Year Project. In *IEEE International Conference on Teaching, Assessment and Learning for Engineering (TALE)* (pp. T1C–18).

Bacea, I. M., Ciupe, A., & Meza, S. N. (2017). Interactive Kanban - Blending Digital and Physical Resources for Collaborative Project Based Learning. In *IEEE International Conference on Advanced Learning Technologies (ICALT)* (pp. 210–211).

Beck, K. (1999). *Extreme Programming Explained: Embrace Change*. Addison-Wesley Professional.

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., … Thomas, D. (2001). *Manifesto for Agile Software Development*. Retrieved from http://agilemanifesto.org

Bruegge, B., Krusche, S., & Wagner, M. (2012). Teaching Tornado: From Communication Models to Releases. In *Educators' Symposium (EduSymp)* (pp. 5–12).

Bruegge, B., Reiss, M., & Schiller, J. (2009). Agile Principles in Academic Education: A Case Study. In *International Conference on Information Technology: New Generations* (pp. 1684–1686).

Cockburn, A. (2004). *Crystal Clear: A Human-Powered Methodology for Small Teams*. Pearson Education.

Cockburn, A., & Highsmith, J. (2001). Agile Software Development, the People Factor. *Computer*, *34*(11), 131–133.

Cubric, M. (2013). An Agile Method for Teaching Agile in Business Schools. *The International Journal of Management Education*, *11*(3), 119–131.

da Silva Coelho, R. A., da Cunha, A. M., Gomes, A. A., Segeti, E. R., Marques, J. C., Vicente, L. M., … Mirachi, S. (2014). Developing a CDS with Scrum in an Interdisciplinary Academic Project. In *IEEE/AIAA Digital Avionics Systems Conference (DASC)* (pp. 5D6–1).

Delhij, A., van Solingen, R., & Wijnands, W. (2015). *The eduScrum Guide* (No. 1.2) (p. 21). Retrieved from http://eduscrum.nl/en/file/CKFiles/The_eduScrum_Guide_EN_1.2.pdf

Dewi, D. A., & Muniandy, M. (2014). The Agility of Agile Methodology for Teaching and Learning Activities. In *Malaysian Software Engineering Conference (MySEC)* (pp. 255–259).

Dingsøyr, T., Nerur, S., Balijepally, V., & Moe, N. B. (2012). A Decade of Agile Methodologies: Towards Explaining Agile Software Development. *Journal of Systems and Software*, *85*(6), 1213–1221.

Duvall, S., Hutchings, D., & Kleckner, M. (2017). Changing Perceptions of Discrete Mathematics Through Scrum-Based Course Management Practices. *Journal of Computing Sciences in Colleges*, *33*(2), 182–189.

Dybå, T., & Dingsøyr, T. (2008). Empirical Studies of Agile Software Development: A Systematic Review. *Information and Software Technology*, *50*(9–10), 833–859.

Fernandes, J. M., & Sousa, S. M. (2010). PlayScrum - A Card Game to Learn the Scrum Agile Method. In *International Conference on Games and Virtual Worlds for Serious Applications* (pp. 52–59).

Fronza, I., Ioini, N. E., & Corral, L. (2017). Teaching Computational Thinking Using Agile Software Engineering Methods: A Framework for Middle Schools. *ACM Transactions on Computing Education*, *17*(4), 1–28.

Gestwicki, P., & McNely, B. (2016). Interdisciplinary Projects in the Academic Studio. *ACM Transactions on Computing Education*, *16*(2), 1–24.

Grimheden, M. E. (2013). Can Agile Methods Enhance Mechatronics Design Education? *Mechatronics*, *23*(8), 967–973.

Haddaway, N. R., Collins, A. M., Coughlin, D., & Kirk, S. (2015). The Role of Google Scholar in Evidence Reviews and Its Applicability to Grey Literature Searching. *PloS One*, *10*(9), e0138237.

Hanks, B. (2007). Becoming Agile Using Service Learning in the Software Engineering Course. In *Agile Conference (AGILE)* (pp. 121–127).

Heikkilä, V. T., Paasivaara, M., & Lassenius, C. (2016). Teaching University Students Kanban with a Collaborative Board Game. In *IEEE/ACM International Conference on Software Engineering: Software Engineering Education and Training Track (ICSE-SEET)* (pp. 471–480).

Heinonen, K., Hirvikoski, K., Luukkainen, M., & Vihavainen, A. (2013). Learning Agile Software Engineering Practices Using Coding Dojo. In *ACM SIGITE Conference on Information Technology Education (SIGITE)* (pp. 97–102).

Kastl, P., Kiesmüller, U., & Romeike, R. (2016). Starting Out with Projects: Experiences with Agile Software Development in High Schools. In *Workshop in Primary and Secondary Computing Education (WiPSCE)* (pp. 60–65).

Kessler, R., & Dykman, N. (2007). Integrating Traditional and Agile Processes in the Classroom. In *ACM Technical Symposium on Computer Science Education (SIGCSE)* (pp. 312–316).

Kitchenham, B., & Charters, S. (2007). *Guidelines for Performing Systematic Literature Reviews in Software Engineering* (EBSE Technical Report). Keele University. Retrieved from https://www.cs.auckland.ac.nz/~norsaremah/2007%20Guidelines%20for%20performing%20SLR%20in%20SE%20v2.3.pdf

Ladas, C. (2009). *Scrumban: Essays on Kanban Systems for Lean Software Development*. Modus Cooperandi Press.

Lee, Y., Marepalli, D. B., & Yang, J. (2017). Teaching Test-Drive Development Using Dojo. *Journal of Computing Sciences in Colleges*, *32*(4), 106–112.

Lembo, D., & Vacca, M. (2012). *Project Based Learning + Agile Instructional Design = EXtreme Programming based Instructional Design Methodology for Collaborative Teaching* (No. 8). Dipartimento di Informatica e Sistemistica "Antonio Ruberti", Sapienza Università di Roma.

Lu, B., & DeClue, T. (2011). Teaching Agile Methodology in a Software Engineering Capstone Course. *Journal of Computing Sciences in Colleges*, *26*(5), 293–299.

Lui, D., Litts, B. K., Widman, S., Walker, J. T., & Kafai, Y. B. (2016). Collaborative Maker Activities in the Classroom: Case Studies of High School Student Pairs' Interactions in Designing Electronic Textiles. In *Annual Conference on Creativity and Fabrication in Education (FabLearn)* (pp. 74–77).

Luz, R. B. da, Neto, A. G. S. S., & Noronha, R. V. (2013). Teaching TDD, the Coding Dojo Style. In *IEEE International Conference on Advanced Learning Technologies (ICALT)* (pp. 371–375).

Lynch, T. D., Herold, M., Bolinger, J., Deshpande, S., Bihari, T., Ramanathan, J., & Ramnath, R. (2011). An Agile Boot Camp: Using a LEGO®-Based Active Game to Ground Agile Development Principles. In *IEEE Frontiers in Education Conference (FIE)* (pp. F1H–1).

Mahnic, V. (2012). A Capstone Course on Agile Software Development Using Scrum. *IEEE Transactions on Education*, *55*(1), 99–106.

Mäkiö, J., Mäkiö-Marusik, E., & Yablochnikov, E. (2016). Task-Centric Holistic Agile Approach on Teaching Cyber Physical Systems Engineering. In *Annual Conference of the IEEE Industrial Electronics Society (IECON)* (pp. 6608–6614).

Meerbaum-Salant, O., & Hazzan, O. (2010). An Agile Constructionist Mentoring Methodology for Software Projects in the High School. *ACM Transactions on Computing Education (TOCE)*, *9*(4), 21.

Melnik, G., & Maurer, F. (2003). Introducing Agile Methods in Learning Environments: Lessons Learned. In *Conference on Extreme Programming and Agile Methods* (pp. 172–184).

Melnik, G., & Maurer, F. (2005). A Cross-Program Investigation of Students' Perceptions of Agile Methods. In *IEEE/ACM International Conference on Software Engineering (ICSE)* (pp. 481–488).

Missiroli, M., Russo, D., & Ciancarini, P. (2017). Agile for Millennials: A Comparative Study. In *IEEE/ACM International Workshop on Software Engineering Curricula for Millennials (SECM)* (pp. 47–53).

Noguera, I., Guerrero-Roldán, A.-E., & Masó, R. (2018). Collaborative Agile Learning in Online Environments: Strategies for Improving Team Regulation and Project Management. *Computers & Education*, *116*, 110–129.

Paasivaara, M., Heikkilä, V., Lassenius, C., & Toivola, T. (2014). Teaching Students Scrum Using LEGO Blocks. In *IEEE/ACM International Conference on Software Engineering: Software Engineering Education and Training Track (ICSE-SEET)* (pp. 382–391).

Paez, N. M. (2017). A Flipped Classroom Experience Teaching Software Engineering. In *IEEE/ACM International Workshop on Software Engineering Curricula for Millenials (SECM)* (pp. 16–20).

Palmer, S. R., & Felsing, M. (2001). *A Practical Guide to Feature-Driven Development*. Pearson Education.

Pinto, L., Rosa, R., Pacheco, C., Xavier, C., Barreto, R., Lucena, V., … Maurçcio, C. (2009). On the Use of Scrum for the Management of Practical Projects in Graduate Courses. In *IEEE Frontiers in Education Conference (FIE)* (pp. 1–6).

Ramos, M. P., Matuck, G. R., Matrigrani, C. F., Mirachi, S., Segeti, E., Leite, M., … Dias, L. A. V. (2013). Applying Interdisciplinarity and Agile Methods in the Development of a Smart Grids System. In *International Conference on Information Technology: New Generations* (pp. 103–110).

Rico, D. F., & Sayani, H. H. (2009). Use of Agile Methods in Software Engineering Education. In *Agile Conference (AGILE)* (pp. 174–179).

Ringert, J. O., Rumpe, B., Schulze, C., & Wortmann, A. (2017). Teaching Agile Model-Driven Engineering for Cyber-Physical Systems. In *IEEE/ACM International Conference on Software Engineering: Software Engineering Education and Training Track (ICSE-SEET)* (pp. 127–136).

Romeike, R., & Göttel, T. (2012). Agile Projects in High School Computing Education: Emphasizing a Learners' Perspective. In *Workshop in Primary and Secondary Computing Education (WiPSCE)* (pp. 48–57).

Scharf, A., & Koch, A. (2013). Scrum in a Software Engineering Course: An in-Depth Praxis Report. In *IEEE International Conference on Software Engineering Education and Training (CSEE&T)* (pp. 159–168).

Scharff, C., & Verma, R. (2010). Scrum to Support Mobile Application Development Projects in a Just-in-Time Learning Context. In *ICSE Workshop on Cooperative and Human Aspects of Software Engineering* (pp. 25–31).

Schwaber, K., & Beedle, M. (2002). *Agile Software Development with Scrum*. Prentice Hall Upper Saddle River.

Schwaber, K., & Sutherland, J. (2011). *The Scrum Guide*. Scrum Alliance.

Seman, L. O., Hausmann, R., & Bezerra, E. A. (2018). On the Students' Perceptions of the Knowledge Formation When Submitted to a Project-Based Learning Environment Using Web Applications. *Computers & Education*, *117*, 16–30.

Smith, T., Cooper, K. M., & Longstreet, C. S. (2011). Software Engineering Senior Design Course: Experiences with Agile Game Development in a Capstone Project. In *International Workshop on Games and Software Engineering (GAS)* (pp. 9–12).

Stapel, K., Lübke, D., & Knauss, E. (2008). Best Practices in Extreme Programming Course Design. In *IEEE/ACM International Conference on Software Engineering (ICSE)* (pp. 769–776).

Stapleton, J. (2003). *DSDM: Business Focused Development*. Pearson Education.

Steghöfer, J.-P., Burden, H., Alahyari, H., & Haneberg, D. (2017). No Silver Brick: Opportunities and Limitations of Teaching Scrum with Lego Workshops. *Journal of Systems and Software*, *131*, 230–247.

Stewart, J. C., DeCusatis, C. S., Kidder, K., Massi, J. R., & Anne, K. M. (2009). Evaluating Agile Principles in Active and Cooperative Learning. In *Student-Faculty Research Day, CSIS, Pace University* (p. B3).

Sugimori, Y., Kusunoki, K., Cho, F., & Uchikawa, S. (1977). Toyota Production System and Kanban System Materialization of Just-in-Time and Respect-for-Human System. *The International Journal of Production Research*, *15*(6), 553–564.

Takeuchi, H. & Nonaka, I. (1986, January). The New New Product Development Game. *Harvard Business Review*. Retrieved from https://hbr.org/1986/01/the-new-new-product-development-game

Umapathy, K., & Ritzhaupt, A. D. (2017). A Meta-Analysis of Pair-Programming in Computer Programming Courses: Implications for Educational Practice. *ACM Transactions on Computing Education*, *17*(4), 1–13.

Vivian, R., Falkner, K., & Falkner, N. (2013). Analysing Computer Science Students' Teamwork Role Adoption in an Online Self-Organised Teamwork Activity. In *Koli Calling International Conference on Computing Education Research (Koli Calling)* (pp. 105–114).

von Wangenheim, C. G., Savi, R., & Borgatto, A. F. (2013). SCRUMIA: An Educational Game for Teaching Scrum in Computing Courses. *Journal of Systems and Software*, *86*(10), 2675–2687.

Werner, L., Arcamone, D., & Ross, B. (2012). Using Scrum in a Quarter-Length Undergraduate Software Engineering Course. *Journal of Computing Sciences in Colleges*, *27*(4), 140–150.

Wohlin, C. (2014). Guidelines for Snowballing in Systematic Literature Studies and a Replication in Software Engineering. In *International Conference on Evaluation and Assessment in Software Engineering (EASE)* (p. 38).

Zorzo, S. D., de Ponte, L., & Lucredio, D. (2013). Using Scrum to Teach Software Engineering: A Case Study. In *IEEE Frontiers in Education Conference (FIE)* (pp. 455–461).