Overcoming Challenges in Software Engineering Education:

Delivering Non-Technical Knowledge and Skills

Liguo Yu
Indiana University South Bend, USA

ENGINEERING SCIENCE REFERENCE
An Imprint of IGI Global

Chapter 13
Experiences in Software Engineering Education: Using Scrum, Agile Coaching, and Virtual Reality

Ezequiel Scott
ISISTAN (UNICEN-CONICET) Research Institute, Argentina

Guillermo Rodríguez
ISISTAN (UNICEN-CONICET) Research Institute, Argentina

Álvaro Soria
ISISTAN (UNICEN-CONICET) Research Institute, Argentina

Marcelo Campo
ISISTAN (UNICEN-CONICET) Research Institute, Argentina

ABSTRACT

Software Engineering courses aim to train students to succeed in meeting the challenges within competitive and ever-changing professional contexts. Thus, undergraduate courses require continual revision and updating so as to cater for the demands of the software industry and guarantee academic quality. In this context, Scrum results in both a suitable and a flexible framework to train students in the implementation of professional software engineering practices. However, current approaches fail to provide guidance and assistance in applying Scrum, or a platform to address limitations in time, scope, and facilities within university premises. In this chapter, the authors present a software engineering training model based on the integration of the Agile Coach role and a virtual-reality platform called Virtual Scrum. The findings highlight the benefits of integrating this innovative model in a capstone course. Not only does this approach strengthen the acquisition of current software engineering practices but also opens new possibilities in the design of training courses.

DOI: 10.4018/978-1-4666-5800-4.ch013

INTRODUCTION

The increasing complexity of the software industry, constant changes in system requirements, distributed environments and professional mobility are ordinary challenges to software developers, who are required to be competent and proactive by rapidly expanding software industries. In order to prepare undergraduate students for ongoing success in this context, Software Engineering (SE) courses must provide effective training in the application of software development practices typically implemented in large projects and organizations (Alfonso & Botia, 2005; Mahnic 2010).

In the light of the above, we structured a SE course based on Capability Maturity Model Integration (CMMI), since this model includes professional and reliable SE practices (Kulpa, 2008). In particular, we used the framework CMMI for development version 1.3 (CMMI-DEV 1.3), which aims to provide software organizations with superior project performance, client satisfaction, and quality of products and services. To support CMMI, we designed an initial version of the course based on the Rational Unified Process (RUP) (Kruchten, 2003) to develop a capstone project (Coupal & Boechler, 2005; Devedzic & Milenkovic, 2011).

Unfortunately, teaching SE to students running a software project following RUP presented several drawbacks. As it is a plan-driven development framework, RUP requires the association of project milestones with specific dates. This made students focus on meeting deadlines and delivering the agreed milestone, skipping activities in the RUP framework. Additionally, even though RUP encourages the overlapping of phases, students inevitably slid into a waterfall-like process (Osorio, 2011). Thus, it is difficult for inexperienced students to detect mistakes made in early stages until the development process reaches the final stages. Furthermore, the prescriptive nature of RUP and the definition of specific roles (i.e. project manager and business process analyst, IBM, n.d.) to perform planning and estimating activities fail to both promote actively participation of all team members, and encourage them to make commitments regarding those activities. Since SE involves people and social interaction, these aspects are cornerstone of its reality and crucial to professional training and development. As a consequence, we moved our RUP implementation of CMMI to Scrum.

The selection of Scrum is founded on the fact that it has increased in the last few years; in 2012, 57% of respondents used Scrum or its variants, and nowadays, more than 50% of the surveyed companies are utilizing Scrum as an effective path towards agility (Versionone, n.d.). First, this agile methodology concentrates on project management practices and includes monitoring and feedback activities that focus on transparency, team effort, effective time management and personal interactions. Second, Scrum facilitates the process of satisfying requirements, for while all the requirements appear in the backlog, only a limited set is fulfilled in each Sprint. Third, the retrospective meetings and continual customer contact allow early detection of impediments and rearrangements of working plans.

In line with the popularity and effectiveness of Scrum in professional contexts, some scholars have engaged in integrating Scrum in undergraduate software engineering courses. In (Alfonso & Botia, 2005) the authors compared the results of teaching Scrum practices in an undergraduate software course to those obtained in an experience using a waterfall-like rigid process. From the experience, the authors founded that the use of Scrum resulted in lower risks and process overhead (Diaz, 2009). Chuan-Hoo Tan et al. introduced Extreme Programming (XP), Scrum and Feature-Driven Development (FDD) to initiate future software professionals in the importance of agility, flexibility and adaptability in professional contexts. The research consisted in training students in developing and delivering large-scale systems under time constraints and shifting deadlines.

The students had to carry out a project-work by following an iterative and incremental teaching approach (Tan, Tan, & Teo, 2008). Mahnic taught Scrum through a capstone project[1] and described the course details, students' perceptions and teachers' observations after the course (Mahnic, 2012a). The author concluded that the students were overwhelmingly positive about the Scrum-based course, indicating that the course fully met or even exceeded their expectations.

Although those experiences proved to be effective in the classroom, there are still challenges to be faced by professors so as to facilitate students' integration in professional contexts. On the one hand, a typical problem of current approaches is the lack of assistance to unskilled students in performing the agile practices (Dubinsky & Hazzan, 2003). Even when learning in an agile context, the students tend to write enormous documents of requirements, fall in waterfall-like issues, follow a plan instead of responding to change, and focus on delivery dates instead of product quality. On the other hand, the aforementioned approaches fail to address teaching constrains in university courses such as large classes, multiple groups working at a time as well as limited space, and tutors. Using a room for multiple teams may jeopardize the effective implementation of the Scrum process, since each team may require customized configurations of the room. What is more, accessing to the required teaching material or physical space for a personalized class may prove impracticable.

In this context, the aim of this chapter is two-fold. First, we have introduced Scrum in an undergraduate software engineering course and have complemented the Scrum roles with the Agile Coach role, which is played by the professor (Soria, Rodriguez, & Campo, 2012). This role arises as an alternative to scaffold students' progress, remove impediments, and advise on leadership and management strategies as needed to ensure success of the methodology (Hedin, Bendix, & Magnusson, 2005). Here, the Agile Coach is focused on the optimization of the teams and is responsible for clarifying and facilitating the Scrum activities and artifacts, assessing the team's health, and providing teams with needed learning opportunities to address difficulties such as decision-making and meeting deadlines (Appelo, 2010). Second, we have presented *Virtual Scrum* (Rodríguez, Soria, & Campo, 2012a) as a solution to the challenges of implementing Scrum within university premises. *Virtual Scrum* is a prototype tool that aims to help Scrum students set up a virtual working environment in which the visual metaphors required by the methodology are effectively displayed. The tool exploits the richness of the 3D metaphor by using virtual worlds and provides a virtual working environment equipped with different visual management aids for accessing team performance and a room for Daily Meetings.

The lessons learned from our experience in an undergraduate SE course show that Scrum together with the Agile Coach role allows professors to generate learning scenarios in which students can explore and acquire non-technical skills, namely communication, negotiation, interaction with customer, adherence to a process, awareness of software quality, among others. We evidenced an increase in the coverage of the professional and reliable SE practices and enhancement in Scrum perception, resulting in high-quality delivered products. Additionally, this chapter reports on the effectiveness of *Virtual Scrum* to teach up-to-date software engineering procedures and training students in the use of valuable skills to work in professional contexts through a virtual world. Along this line, we found that *Virtual Scrum* is a viable tool to implement the different elements in a Scrum team room and to perform activities throughout the Scrum process. Finally, this virtual hands-on experience may result intrinsically motivating for students, who participate in collaborative work, and enhance their comprehension of Scrum.

The rest of the chapter is organized as follows. Section 2 describes the main concepts of Scrum and current trends in its teaching on software

engineering courses. Section 3 presents our innovative training model and the support provided by *Virtual Scrum*. Section 4 describes the results of our experience and recommendations to introduce this model in different contexts. Section 5 reports the future research directions. Finally, section 6 outlines the conclusions of this chapter.

BACKGROUND

CMMI is a framework which consists of a set of best practices that address the development and maintenance of products and services. These practices cover the product life cycle from conception through delivery and maintenance (Kruchten, 2003). CMM is neither a recipe nor guarantee for success, i.e., CMMI refers to "what to do" rather than "how to do it." CMMI is organized in process areas. A process area is a group of related activities performed collectively to achieve a set of goals. Some goals and practices are specific to the process area; others are generic and apply across all process areas (Kruchten, 2003).

Introducing CMMI into the classroom allows software engineering students to deliver reliable, evolvable and quality products (Barbosa & Maldonado, 2006). The idea behind the inclusion of CMMI in the curricula is fourfold, namely to broaden the coverage of important topics in a software engineering course, to deepen the students' understanding of software engineering practices, to provide quality and process standards at organization level, and to define the key elements of an effective process and outlines how to improve suboptimal processes (Lutteroth et al., 2007). However, CMMI is often misunderstood as being required massive documentation, many layers of personnel and the use of a rigid waterfall life cycle. By following Agile Methods (AM) it is possible to obtain maturity levels with less overhead and effort (Boehm & Tuner, 2008; Alegría & Bastarrica, 2006). That is, the use of a combination between AM and CMMI results in benefits to the business performance by exploiting the synergies of both approaches (Glazer, 2008; Glazer, 2010). The value from AM can only be obtained through disciplined use. It is worth mentioning that we utilized CMMI as a reference model for teaching software engineering practices by means of Scrum; on the other hand, the use of CMMI for a single project is attributed to time and schedule constraints within university premises. Although CMMI areas, such as the ones related to quantitative project management, causal analysis and management of product suppliers, among others, fail to be widespread explored, students explore reliable software practices such as adherence to a process and compliance with quality criteria, requirement management, and configuration management, among others, which are also trustworthy practices defined in CMMI. Particularly, we mainly emphasizes on practices of maturity level 3 associated with process definition and process improvement, quality assurance, validation and verification, requirement development, software design, configuration management, and project monitoring and control, among others, which are essential to prepare students for success in professional contexts.

The emergence of the Manifesto for Agile Software Development (Agile Manifesto, n.d.) brought changes in the software development community. The Manifesto essentially defines a new focus on the software development based on agility, flexibility, communication abilities and the capacity for offering new products and services with high value to the marketplace in short periods of time (Highsmith, 2004). Most companies are adopting Scrum to become agile smoothly and reduce overhead and bureaucracy progressively, without losing sight of the quality of the software product (Diaz, 2009; Maher, 2009).

Scrum is an iterative and incremental methodology that organizes projects to make them manageable for small, self-organized and cross-functional teams (Schwaber & Beedle, 2002), and also systematizes software projects, pursu-

ing successful software development practices by emphasizing teamwork interaction. Scrum defines three roles: Product Owner, Scrum Team and Scrum Master. The Product Owner represents the customer and owns the Product Backlog, and works closely with the Scrum Team to provide clarification and approval on User Stories. The Scrum Team is responsible for completing the work and developing User Stories. The Scrum Master is responsible for facilitating the process, assisting the Product Owner in planning the releases and helping the Scrum Team progress by removing impediments and making resources available.

The widespread use of Scrum has evidenced a mismatch between academic instructions and software industry requirements. Beyond being focused mainly on trainings and certifications, the teaching of Scrum has become a cornerstone of undergraduate software engineering courses. However, although Scrum has been considerably used as a teaching strategy to introduce software engineering practices to students, there is insufficient educational material covering the procedures, methods and personal interactional patterns within the software industry, which would highly facilitate students' integration in professional contexts. In order to address this issue, this chapter presents guidelines and teaching strategies to successfully integrate Scrum into undergraduate and postgraduate courses as a further step to reduce the gap between software industry and academia.

Teaching Scrum

Given that the adoption of the Scrum by industry is becoming mainstream (Mathiassen & Pries-Heje, 2006; Nerur & Balijepally, 2007), it is important to consider why agile software development should be taught at university level. The iterative and incremental nature of Scrum, together with daily and retrospective meetings, allow students to receive immediate feedback on how agile practices are performed; furthermore they allow timely identification of problems in the understanding of agile practices, and to define corrective actions to be applied by the students in subsequent Sprints. In contrast to RUP, the students receive this feedback without waiting for the end of the project. On the other hand, student teams may lack the skills to manage a project, particularly when Scrum is implemented for the first time. Taking the recommendations from the Software Engineering 2004 Curriculum (CCSE, n.d.), there are skills, such as teamwork-related ones, that students should acquire, since agile software development is based on teamwork. Moreover, since agile methods support learning processes, we endorse the idea that teaching agile software development might reduce the cognitive complexity of software development processes and foster the acquisition of skills by making Scrum more comprehensible and suitable to undergraduate students.

Recently, the use of capstone projects based on Scrum has been adopted as a vehicle for teaching the basic concepts in software engineering (Mahnic, 2012a). This kind of project is developed in the classroom and supervised by professors. This strategy aims to increase student's participation in the learning process and address not only common problems found in the development of software systems but also several values proposed by the Agile Manifesto (Cano, 2011; Maher, 2009). In (Coupal & Boechler, 2005) an experience comparing a capstone project developed following an agile approach to their previous projects developed in a traditional way has been reported. In (Devedzic & Milenkovic, 2011), the authors described their eight years of experiences in teaching agile software methodologies to various groups of students at different universities. Based on the experience acquired, they recommended how to overcome potential problems in teaching agile software development. Out of the various agile approaches, Scrum has shown to be the most dominating AM in use (Kniberg, 2007; Mahnic, 2010). In (Mahnic, 2011) the achievement of teaching goals and the empirical evaluation of student's progress in

estimation and planning skills using Scrum have been discussed. Also, the behavior of students using Scrum for the first time has been observed (Mahnic, 2012a). One of the benefits of Scrum is that it enables students to have a better teamwork environment and a better communication that results in high-quality products (Diaz, 2009).

All the aforementioned approaches share that, beyond the scope of the Scrum Team, there are management responsibilities such as management of financial resources, business decision-makings and management of the organization's environment (Kniberg, 2008). Following this line, we argue that augmenting the Scrum roles with the agile coach is vital in learning agile principles, particularly when students implement the agile method for the first time.

Agile Coaching

A typical problem in teaching contexts is the lack of coaching in Scrum teams. Augmenting the Scrum roles with the Agile Coach allows professors to train students in the diversity of aspects of software development, to deepen in the understanding of a Scrum-based process, and to acquire skills related to coaching and leadership (Hedin, Bendix, & Magnusson, 2005). It would be beneficial to integrate coaching in a Scrum course built on a simulation of a professional scenario, in order to provide students with an opportunity to experience the use of Scrum within a safe and controlled context. Despite, the acknowledged benefits of coaching, it has been proved challenging to implement effective agile coaching strategies even within academia.

An Agile Coach takes into account critical problems inside and outside students' teams (Dubinsky & Hazzan, 2003), solve impediments and encourage professors, team members and Scrum Masters to apply agile values. Although all the academic works mentioned above have recognized that agile coaching is a key factor, there has been insufficient analysis on the impact of coaching on the performance of software engineering students. The Agile Coach is meant to support and scaffold students' development by acting as a consultant and trainer in practices of software engineering, maintaining an appropriate balance between coaching and self-organization. It is also worth clarifying that the Agile Coach should not be involved in the project and is responsible for maintaining an appropriate balance between coaching and self-organization.

On Using Virtual Aids

In order to make Agile Coaching a reality, we agree with the idea that tools for planning, performing meetings, playing the role of customer, monitoring how students follow the Scrum process, visualizing information and managing the traceability of requirements are crucial to support Scrum practices in a teaching context. At the beginning, several 2D tools were considered for teaching Scrum; however, they fail to represent the physical development environment so as to carry out a realistic simulation of the Scrum process, which would be beneficial in a distributed context. A virtual world is an adequate solution to the problem of teaching the Scrum framework, through a capstone project, within the university premises when lack of physical space and resources prevent the creation of Scrum team rooms. Recent developments have seen Virtual Learning Environments rapidly become an integral part of teaching and learning provision in further and higher education (Callaghan, 2009). This evolution progresses strongly as educators do their best to adopt and adapt new technologies in the provision of more interactive teaching materials and learning environments which allow students the ability not only to view content but also to interact and organize it to suit their personal needs. Video games and virtual worlds are moving into the mainstream as traditional media industries struggle to keep with up digital natives and their desire for information, technology and connectivity. Researchers in the field are convinced that these technologies are

potentially profitable to education (Anderson, 2008; Rodríguez, Soria, & Campo, 2012b).

In the light of the above, a 3D virtual environment allows developers to know about tasks performed by their peers, and even hold meetings regardless their physical location. Also, a 3D environment can be used to provide a physical topology of both a software project and process. This characteristic allows for a faster access to information than a 2D tool plus videoconference system to carry out meetings within a distributed team (Whitehead, 2007). This benefit may be attributed to the possibility of integrating isolated tools in a single development environment, leading to a context in which it is viable to share software artifacts (Herbsleb, 2007).

Using a virtual world has many advantages in comparison with 2D tools. For instance, students use avatars to communicate through gesture, sound, icons, text, among others (Herbsleb, 2007); and manipulate elements in the modeled room going through a 3D experience of Scrum. Additionally, the sense of immersion and the simulated team room, equipped with the Scrum artifacts and rules, are crucial to engage users in a hands-on experience of using Scrum, participate in collaborative work, and improve their comprehension of this agile method.

To sum up, the rest of this chapter explores the hypothesis that introducing the role of Agile Coach into a software engineering course and supporting the Scrum process with *Virtual Scrum* are effective teaching strategies for training students in the skills that are currently required by the software industry. In contrast to an e-learning platform, we aim to simulate a quasi-real software organization by means of our training model running on a capstone course, in which students face current challenges that occur in the software industry. On the one hand, the Agile Coach is essential to train and guide students along the course; on the other hand, *Virtual Scrum* attempts to setup a Scrum-based team room despite the limitations within university premises.

TEACHING SCRUM WITH AGILE COACHING AND VIRTUAL REALITY

The training model proposed in this chapter is the results of many years of teaching software engineering practices in the context of the Software Engineering course within the Systems Engineering BSc program at the Faculty of Exact Sciences (Universidad Nacional del Centro de la Provincia de Buenos Aires, i.e. UNICEN). The aim of the course is to engage students to develop a capstone project so as to experience a context as professional as possible. The students attending the course have been trained in software system design, object-oriented programming, operating systems and networks, and database management; additionally, they were given a previous course in which they have learnt basic concepts of software engineering. The capstone project has been designed to be completed within a course usually lasting 16 weeks and taken by the students in the ninth semester, side-by-side with other courses. The capstone project is organized and prioritized according to the Product Backlog (the master list of the desired features in the product, which are called User Stories and are grouped into short iterations called Sprints) divided into three Sprints (students are expected to work for about 2 hours a day).

Following Scrum, we assign the roles of Scrum Team and Scrum Master to students and the Product Owner role to a professor. The professor, playing the role of Product Owner, is responsible for defining and clarifying the requirements of the software product, establishing requirement priorities and validating the results obtained from each Sprint. The students, in the role of the Scrum Team, are subdivided into teams that are responsible for developing a set of user stories. The Scrum Master role is taken by different team members in each Sprint (Werner, Arcamone, & Ross, 2012). The reason for this is preserving the self-organizing principle with the aim of promoting students' experiential learning about management activities such as bridging the gap

between the Product Owner and the team, cleaning the team obstacles and ensuring that the Scrum process is followed in terms of values, practices and rules. Furthermore, charging students with these responsibilities leads to optimize team dynamics and productivity, leaving students to reach their full potential (Davies, 2009).

Given that the students who take the course are in contact with Scrum for the first time, they need to be guided and oriented by professors in the use of Scrum so as to learn how to balance discipline and agility. For this reason, we complement the aforementioned Scrum roles with the Agile Coach role, played by another professor. Currently, our training model is designed to support two professors: one playing the role of Product Owner and another playing the role of Agile Coach, but our model can be easily extended to multiple professors. In case the course is given by only one professor, s/he should be able to exchange the Product Owner's hat by the Agile Coach's hat or vice versa with no conflicts. It is worth mentioning that s/he should leave no room for doubt concerning the role played in a certain moment.

As the Sprints progress, the Agile Coach, who is responsible for all the Scrum teams, should try different coaching styles. At the beginning of the course, the Agile Coach focuses on teaching the Scrum rules to be followed by the students. During the first sprint, which lasts 4 weeks, the students learn the basics of Scrum and engage with the idea of producing real value. In this context, the Agile Coach should complement the instructions with anecdotes and experiences in the software industry. During the second sprint, which lasts 3 weeks, if the Agile Coach notices that the students show signs of being trained in Scrum, s/he should suggest students to figure out possible improvements to enhance their performance; however, they need to be guided to complete the recommended practices. During the third sprint, which lasts 2 weeks, the Agile Coach assumes the students have acquired the skills expected at the end of the course, so s/he offers assistance only when necessary and advises students to make some decisions. The second and third Sprints aim to prepare students to be extremely adaptable and foster them to think of solutions that may be away from their experience so far.

To effectively work in an agile context, each group of students in a course should be able to set up an individual Scrum-like team room, with different visual management strategies to perform the Scrum process activities (Diaz, 2009; Sutherland, Jakobsen, & Johnson, 2008): planning cards for estimating user stories, whiteboards for organizing user stories in Product and Sprint Backlog, burn-down charts for accessing team performance and room for performing Daily Meetings. In order to help students within limitations in the university premises, we propose to exploit the richness of the 3D metaphor by exploring virtual worlds. This kind of 3D environments may provide a physical topology of the organization of teamwork along a software project, which would facilitate the following of the Scrum-based process and allow accessing the artifacts, generated during the implementation of Scrum regardless the synchrony of place and time of the team. To ensure reliability, virtual worlds must be designed to be as real-life-like as possible; in this context, *Virtual Scrum* allows students to have information readily available through whiteboards that display the Product and Sprint Backlogs, and trace requirements easily through task boards that show team progress and performance metrics into the virtual world. Indeed, *Virtual Scrum* arises as an alternative to help each team of Scrum students set up a virtual working environment in which the visual metaphors required by Scrum are effectively displayed.

The communication among team members within *Virtual Scrum* is supported by chat and videoconference system in order to tackle collaborative and distributed issues. However, these mechanisms may hinder communication among team members, since they cannot experience the distinct feelings that can be sensed involving

people who are together in the same place. In this context, weekly meetings arise as a face-to-face alternative in order to reinforce the relationship not only among team members, but also between team members and professors, in the role of Product Owner and Agile Coach.

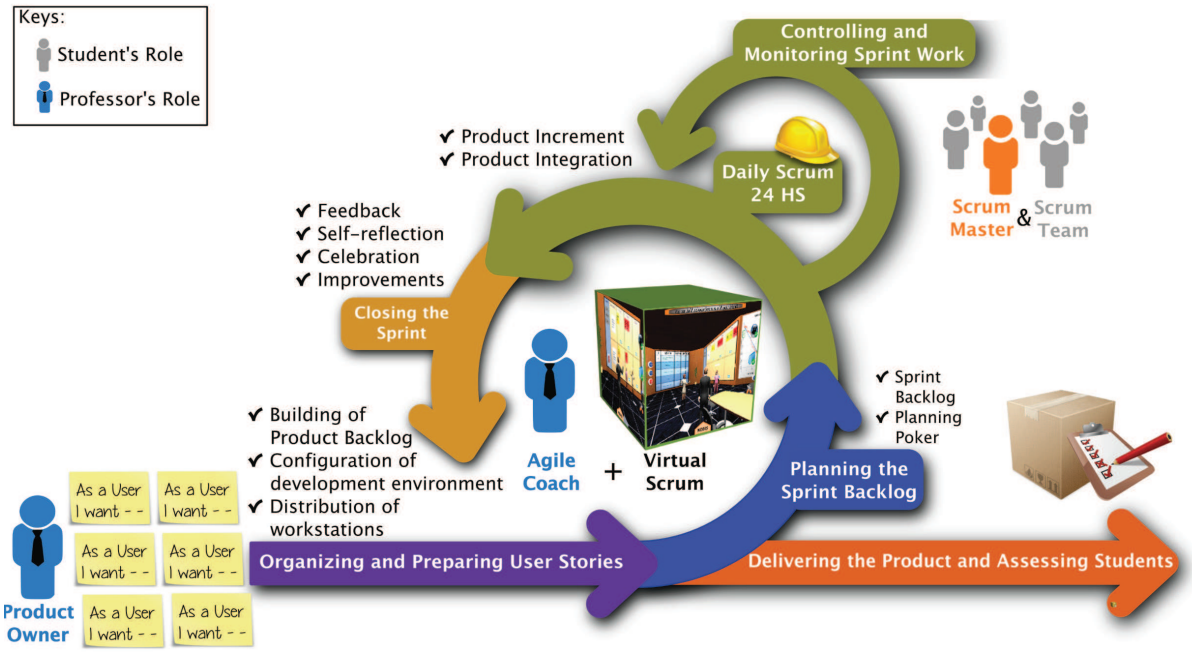
Figure 1 shows our training model, in which each Sprint is organized in 4 phases, namely Organizing and Preparing the User Stories, Planning of the Product Backlog, Controlling and Monitoring the Sprint Work and Closing the Sprint. The first phase consists in building the Product Backlog, customizing the development environment and workstations to students. The second phase is focused on defining the Sprint Backlog and estimating the User Stories to be done during the sprint by using Planning Poker. During the third phase, the students develop User Stories, generate a product increment and integrate it to the working product. The fourth phase consists of feedback provided by the Product Owner, suggestions made by the Agile Coach and celebration within teams. The last Sprint also includes a phase called Delivering the Product and Assessing Students, in which the students deliver the final product obtained from the capstone project and the professors assess the performance of the students along the course.

Organizing and Preparing User Stories

The first phase of the training model is the setup, which consists of two parts. First, the professors give formal lectures on professional software engineering practices recommended by CMMI and show how these practices can be supported with Scrum. Second, the professors provide students with *Virtual Scrum* and teach them how the tool deals with each of the phases. In addition, *Virtual Scrum* can also interact with tools for versioning code, testing classes and documenting artifacts.

After the setup, the building and prioritization of the User Stories into the Product Backlog take place. The Product Backlog is the master list of the desired product features and the professor, in the role of Product Owner, loads the User Stories

Figure 1. The training model



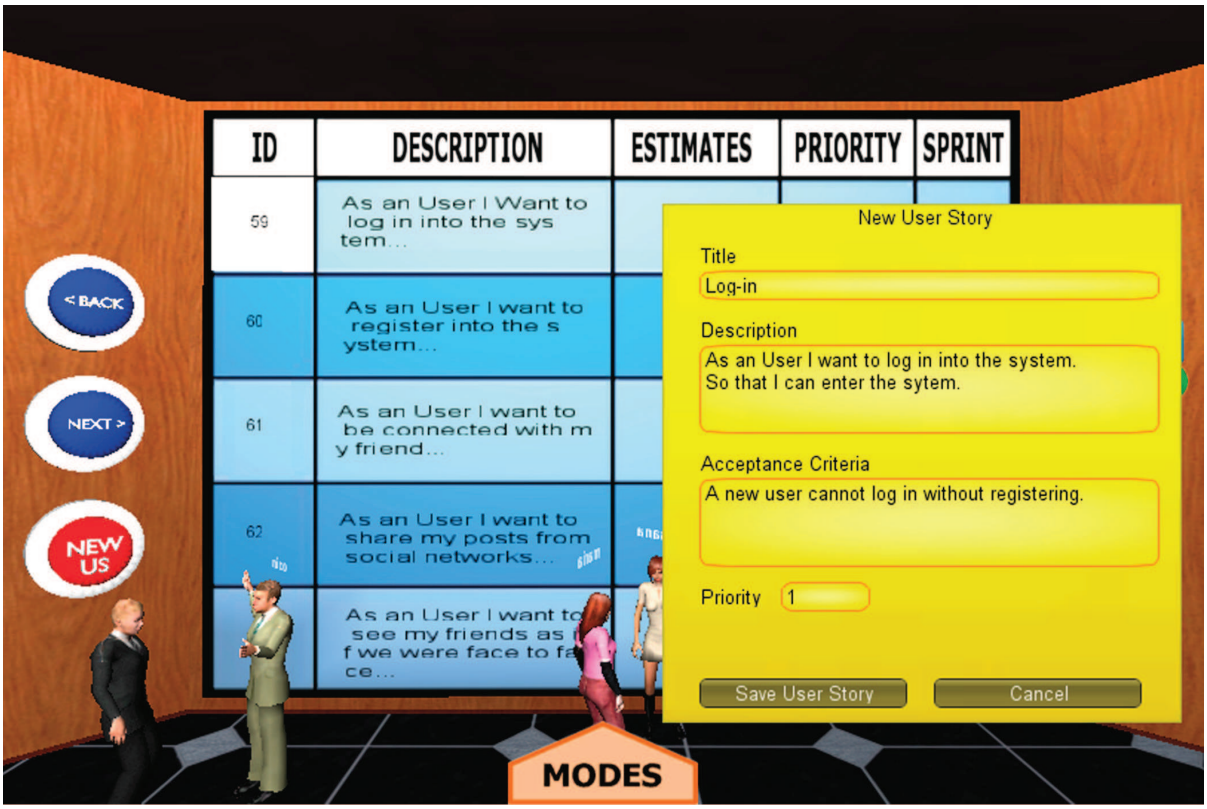
necessary to develop the capstone project. Figure 2 shows an example of the Product Backlog loaded with the user stories for *Universidad3D* (n.d.). *Universidad3D* is a virtual world under the Unity game engine designed as multi-tiered client-server architecture. The virtual world allows users to navigate the facilities of the UNICEN University, play thematic games and attend virtual courses, utilizing chat, e-mail and forum mechanisms for communication.

Universidad3D allow professors to generate different scenarios namely, building a piece of software from scratch, enhancing an already existing solution, or even reengineering an existing legacy system. In all mentioned cases, distributed Scrum teams have been given comparable user stories of similar complexity. For example, a Scrum team is in charge of incorporating features of social networks during the first Sprint, improving quality of 3D models during the second Sprint, and using Web Services to reuse and integrate heterogeneous platforms during the third Sprint.

The Product Backlog is supported by *Virtual Scrum* as a spreadsheet that contains the backlog items. The reason to use a spreadsheet is that it is a suitable tool to create a bit of history and make the work transparent. Also, this document can be easily shared and comprehended by the Scrum Team and the Product Owner because of its simplicity and accessibility (Tan, Tan, & Teo, 2008).

On the spreadsheet, the professor, wearing the Product Owner hat, clarifies and prioritizes (column "Priority" in Figure 2) the User Stories, and the Scrum Team asks the Product Owner for the necessary information by using questionnaires, interviews, and prototypes, among others. Each row of the spreadsheet represents a User Story. A User Story (US), which is identified by an ID,

Figure 2. Product backlog in Virtual Scrum



describes a desired functionality involving role ("As…"), product features ("…I want to…") and the benefit provided to the user ("…so that…"). Row 1 represents the User Story "As a User, I want to log in into the system so that I can enter the system" (ID 59).

As long as students prepare and organize the Product Backlog with the Product Owner, the Agile Coach works on fostering human relationships within the teams so each team member becomes acquainted with the strengths and weaknesses of other partners. Another important issue that the Agile Coach works on is the importance of comprehending the User Stories and the vision of the project by using the above mentioned Product Backlog. For dealing with the User Stories, the Agile Coach should help teams to refine the epics (i.e. large User Stories that rarely can be broken down into smaller ones), introduce non-functional requirements and specify acceptance criteria. For example, the Agile Coach could highlight the non-functional requirement of availability: "As a User I want to the virtual world is available 99.99% of the time I try to access it, so that I do not get frustrated." The "Acceptance Criteria" confirms when the User Story is completed; in the example, the US is tested against a user that tries to log-in into the virtual world without being registered. On the other hand, being aware of the vision is vital and derives some benefits for the team. The students gain momentum if they all work towards the same goal, enjoy working if they understand how they are contributing, and make better decisions and take responsibility if they can visualize what they are able to do along the capstone project.

At the end of this phase, the Scrum Team is ready to select the User Stories that will be developed during each Sprint (column "Sprint" in Figure 2). Here, the Scrum Team moves to the next phase to fill the column "Estimates" in Figure 2 for each of the selected User Stories, which are the intended ones to be delivered at the end of the Sprint.

Planning the Sprint Backlog

The Sprint initiates with the specification of achievable and expected outcomes of that Sprint. The Scrum Team holds a planning meeting to estimate a set of User Stories and reach a decision to start the activities of the Sprint. Considering the recommendations from the literature review (Cohn, 2004; Cohn, 2005), we selected the Planning Poker technique (Grenning, 2002; Mahnic, 2012b) since it increases individual commitment, i.e., those responsible for performing a task are the same who estimate how long this task may take (Brenner, 1996). Another motivation for our decision is that each team member must argue to support estimations (Jorgensen, 2004).

During the Planning Poker session, the team members vote the candidate estimates for the user stories using cards, which contain the story points used to estimate the US. A story point is considered to represent a working day consisting of 2 hours of uninterrupted work. As story point values, we decided to utilize the following Fibonacci sequence 0.5, 1, 2, 3, 5, 8, 13, and 20. The use of Fibonacci sequence to establish orders of magnitude is based on the idea that the ability of developers to accurately discriminate size decreases as the difference between the story points becomes larger (Miranda, 2001).

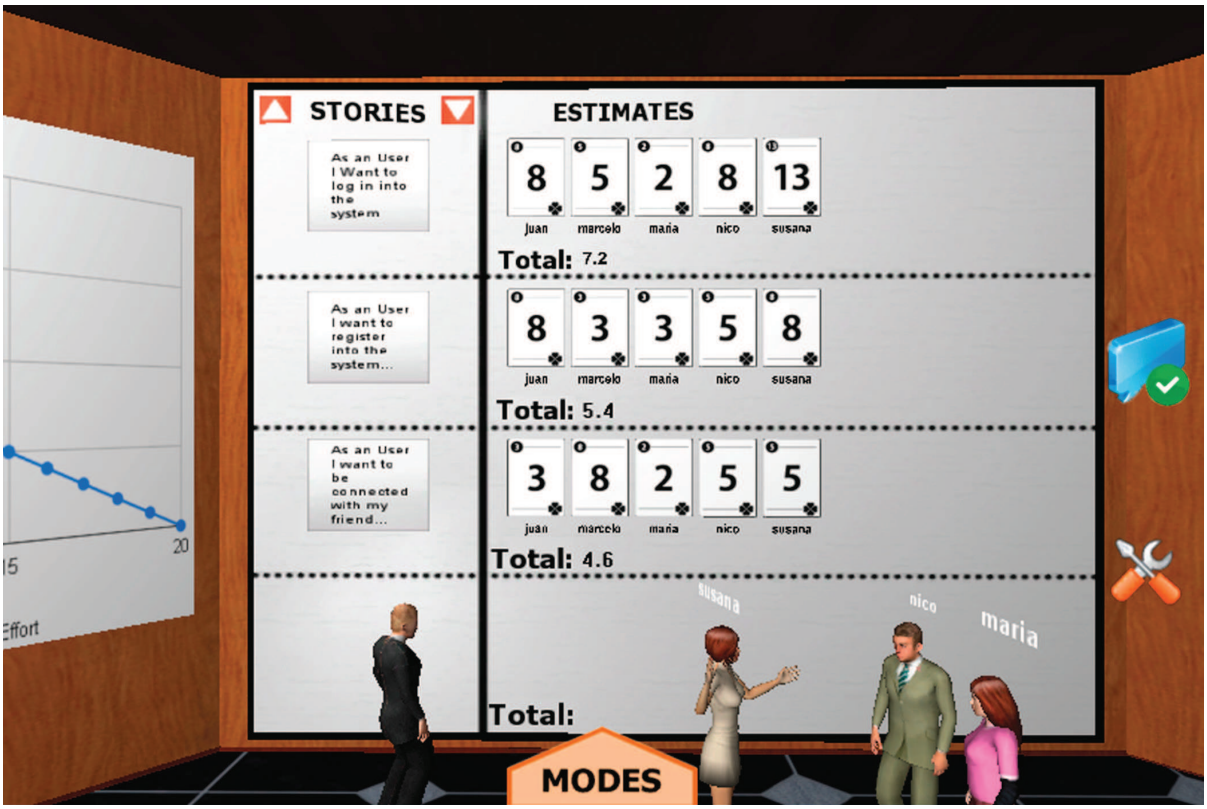
Figure 3 shows the Scrum Team estimating three User Stories from the Product Backlog displayed in Figure 2 by using the Planning Poker Component of *Virtual Scrum*. For each US, team members, using the Planning Poker View, select their cards simultaneously and those members with high and low story points have to justify their estimates by means of either chat or video-conference inside *Virtual Scrum*. Then, all team members vote again until a consensus is reached or the Scrum Master decides to finish the process after three iterations, deciding the average of story points as the estimate. For example, the resulting estimate for the US related to *log-in* was a value of

7.2 story points (Figure 3, Row 1), which meant 14.4 hours of uninterrupted work for this US.

In this context, the Agile Coach should help teams select the User Stories whose estimates are lower or equals than 13, and postponing the others for later where they can be disaggregated. Along the planning meeting, the Agile Coach assists teams to estimate the effort and communicate with the Product Owner. Regarding estimation, the Agile Coach uses *Virtual Scrum* to visualize the estimation process of each team and thus, monitor their status and skills. When it is necessary, the Agile Coach suggests teams to disaggregate complex User Stories into more simply ones to facilitate the estimation of a particular User Story. For instance, a possible suggestion could be "Be careful! You are underestimating the effort to develop that User Story. Maybe it requires to be divided into simpler User Stories."

After performing the planning session, each team has the User Stories that will be developed and load them into the Sprint Backlog. Then, each team disaggregates the User Stories into smaller tasks and the tasks are assigned among team members. Here, the Agile Coach encourages teams to elicit information from the Product Owner so as to avoid teams discarding tasks that can be vital for the development of the User Stories. At last, the team starts the development of those User Stories, which cannot be modified along the Sprint.

Figure 3. Planning Poker in Virtual Scrum

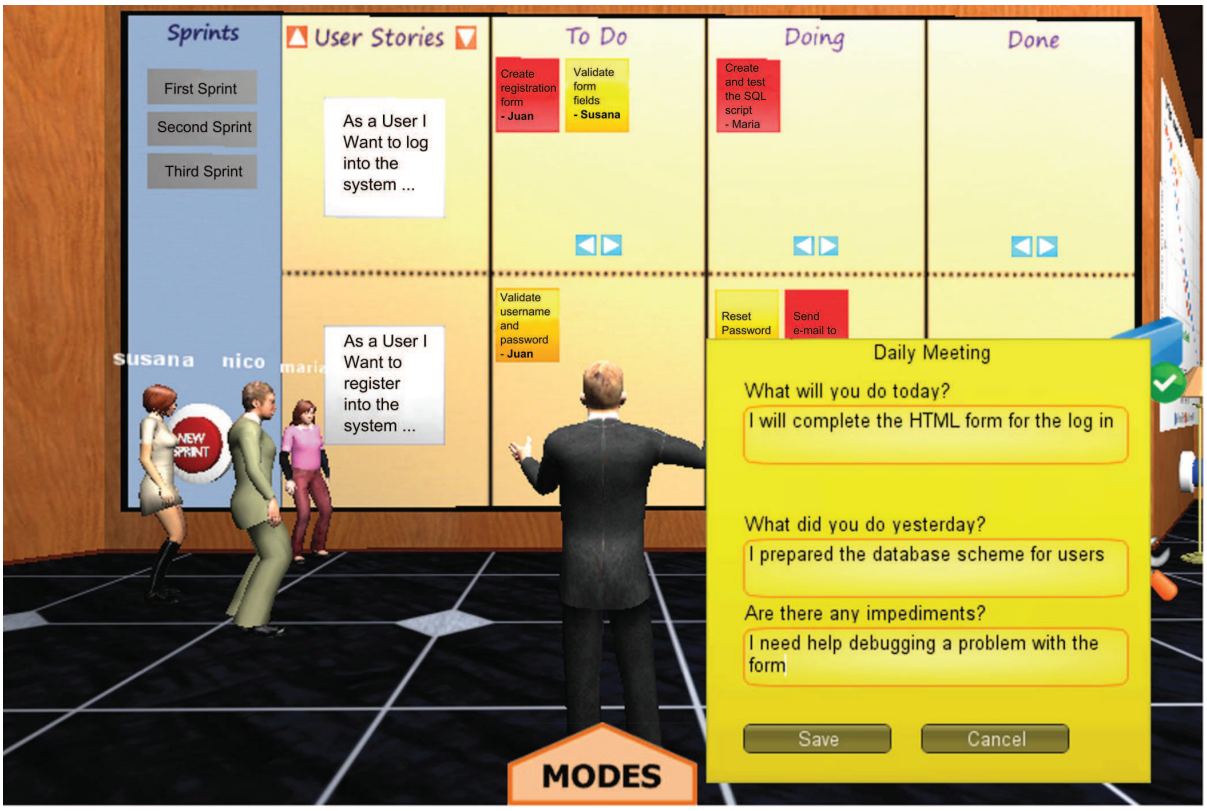


Controlling and Monitoring Sprint Work

During this phase, the Scrum Teams works on developing the tasks associated with the User Stories estimated in the previous phase. Every day of the Sprint at the same time, the team enters *Virtual Scrum* and meets in a Daily Meeting (Figure 4) to review and discuss the progress of the tasks. During a 15-minute's time-boxing period, from his/her working place at different locations each team member answers three questions through the Daily Meeting View artifact: What will you do today?, What did you do yesterday?, and Are there any impediments?. For example, a possible scenario of a Daily Meeting from a team member standpoint can be the following: *What will you do today? I will complete the HTML form for the log-in module. What did you do yesterday? I prepared the database schema for users. Are there any impediments? I need help to debug a problem with the form.* At this point, team members utilize the chat mechanism for communicating each other and can both surf the Web through a Web browser inside *Virtual Scrum* and access to a viewer of pdf documents so as to display any documents necessary to support their answers.

Remarkably, during Daily Meetings *Virtual Scrum* provides Scrum all team members together in the same place with a sense of immersion that allows them to touch a user story and talk about it, which are difficult activities to achieve by using 2D tools in combination with videoconference. Complementary to *Virtual Scrum*, there are open-source tools, such as USVN (n.d.), XWiki (n.d.), and Jira (n.d.), provided by the professors that support software engineering practices, such as configuration management,

Figure 4. Daily meeting in Virtual Scrum



process monitoring, and project tracking, respectively. As these tools can be accessed through the Web Browser, *Virtual Scrum* acts as a project repository along with its virtual artifacts.

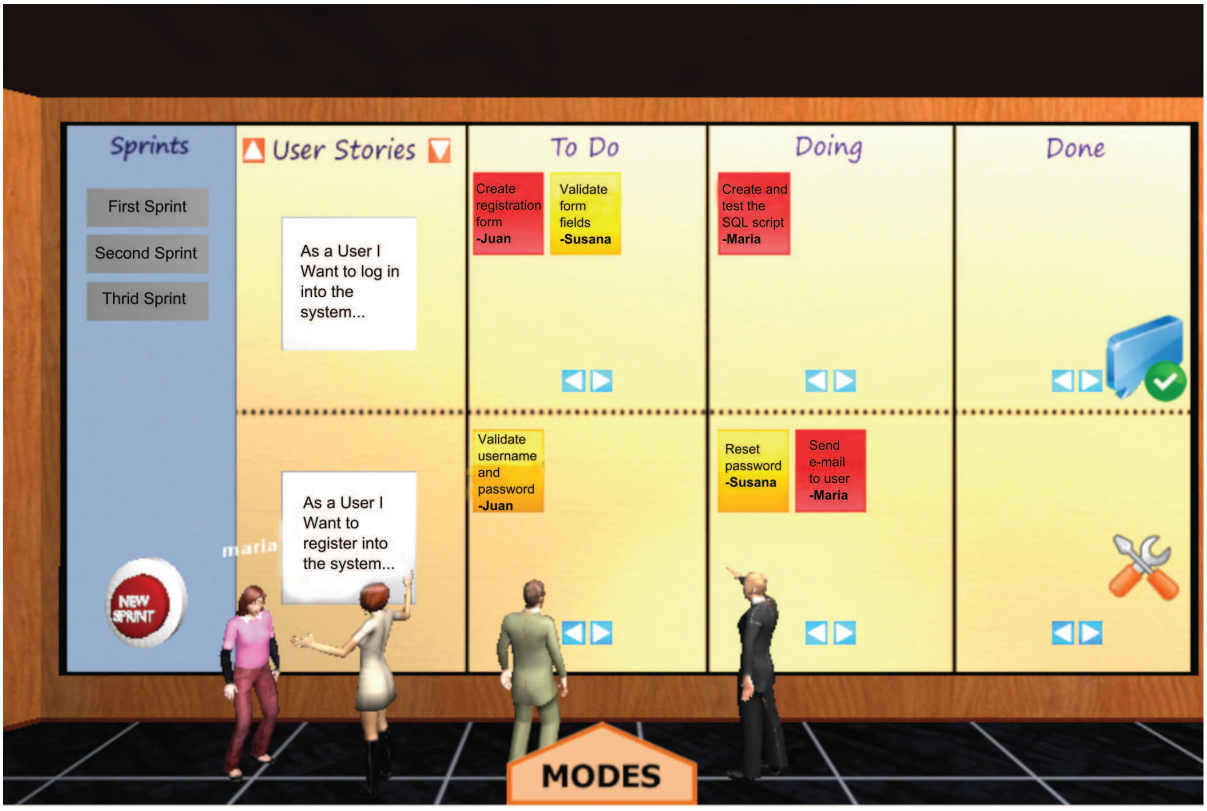
After answering the three questions, the Scrum Team updates the figures of the tasks in the Virtual Task Board as shown in Figure 5.

As the development of the tasks evolves, the tasks move along the DOING list on the Task Board and team members are able to load hours worked into the assigned tasks. When a task is completed by a team member and reviewed by the Scrum Master, this task moves to the DONE list. In order to promote self-organization, when a team member finishes the tasks assigned, s/he selects a new task, usually chosen when answering "What will you do today?"

Notice that the Task Board is a very powerful tool that creates communication among team members easily, since they can glance at it and review the work in progress, completed, or pending. The Task Board both helps team members visualize tasks, and shows what they have accomplished, encouraging them to move on to new tasks. The Task Board is a useful aid to embrace the agile values such as transparency, collaboration, focus, and self-organization, among others. On the other hand, supporting a Virtual Task Board enables each team to setup a specific configuration of the work, regardless the physical location. This is a considerable issue that may jeopardize the learning process within university premises, since students are unable to fully advocate to the course due to other courses, examinations, and external links with companies.

Twice a week, the Agile Coach and each Scrum Team participate in a meeting called Weekly Meeting (up to 15 minutes long), which is face-to-face and fixed by the professor. The Agile Coach monitors students' performance in each stage of

Figure 5. The Task Board in Virtual Scrum



the miniature process by showing *Virtual Scrum* artifacts through a projector in order not to switch to another different Scrum environment for each team. Also, the Agile Coach guides teams in the use of prototypes to validate the user stories and obtain fruitful information to develop them. On Mondays, the students and the Agile Coach revise and analyze estimates and planning decisions; and on Fridays, the students highlight a contrast between planned tasks and executed tasks along the week. During the Weekly Meetings, the Agile Coach stimulates students to show architectural designs, user story specifications and other relevant documentation. During this assistance, *Virtual Scrum* is a crucial tool since it allows the Agile Coach to analyze how teams utilize the Task Board and how they progress on the tasks, without moving from one side to another.

Ideally, this kind of assistance should not interfere with the self-organization of the team. The Agile Coach should remain in silence waiting for the right moment to intervene. For example, the User Stories fail to be well-specified, the Burndown chart shows ups and downs, or the design document is deprecated. Either of these situations is suitable for interfering so as to discuss with the team about how to correct the course of action and address the remaining work.

It is worth clarifying that these meetings do not become Retrospective meetings. On one hand, the Agile Coach should take note of the issues observed so as to design the Retrospective meeting instead of pointing out a specific solution to the problem. On the other hand, the Agile Coach should detect competences and skills that arise spontaneously within each team. This allows the Agile Coach to be aware of strengths and weaknesses of team members; however, s/he should be cautious in emphasizing the merits in order to avoid discouraging the student effectiveness.

Closing the Sprint

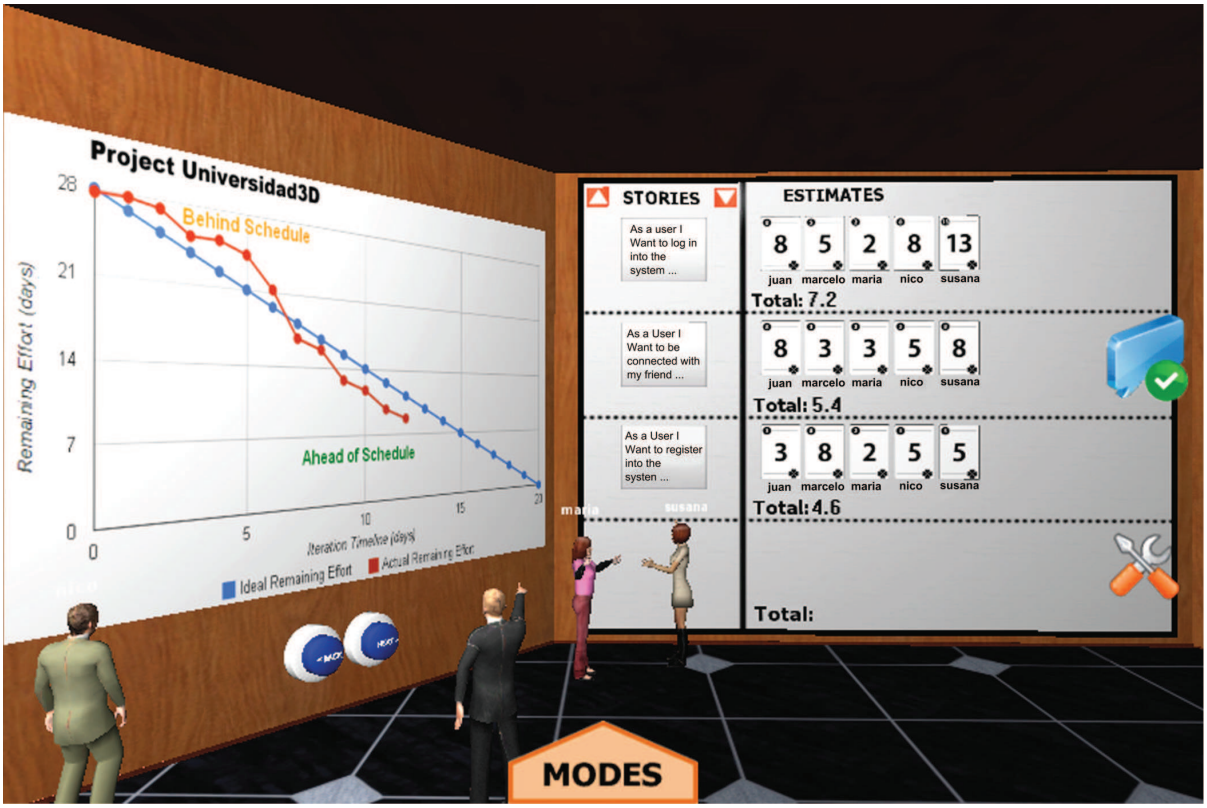
The Sprint concludes with a review of the deliverables to be evaluated by the Product Owner in a Sprint Review meeting, and a Sprint Retrospective meeting to assess achievement of initial goals, review risks and carefully define the process aspects to be improved. In the Sprint Review, the Scrum Team displays the user stories completed during the Sprint by a demo. The objective of the demo is to examine the work done and get feedback from the Product Owner and other stakeholders. During the demo, the team shows how the developed features pass the acceptance tests and the Product Owner may want new features or improvements to the features, after interacting with the real software. In this context, the Agile Coach observes the demo and notes all the points to reinforce later, and identifies corrective actions to solve a particular problem in the miniature process. It is worth mentioning that in our training model, we decided to perform the demo in a face-to-face meeting. The reason for this decision stems from the necessity of augmenting *Virtual Scrum* with a desktop sharing tool to present the software product increment. The simultaneous use of both tools results in a high overhead and technological cost as well as (Oliveria & Lima, 2011) complex interaction that decreases team members' sense of presence and commitment (Paassivaara, 2009). A further reason for utilizing face-to-face meetings is that the distributed agile software development may lead to breakdowns in communication, misunderstandings among team members (Shrivastava & Date, 2010) and mistrust between the students and the professor.

After the Sprint Review, the team holds the Sprint Retrospective. The Agile Coach addresses the meeting by considering the issues observed during the previous phase. In this meeting, the

Agile Coach helps each member learns from the other in the best way; the methodology consists in pointing out a specific solution to a problem if necessary by considering the competences and skills within each team that were detected before. This meeting represents an opportunity to reflect on what occurred during the iteration, and identify the problems that may prevent the team from improving their productivity. To do so, each team member mentions the problems that need to be addressed, and the Scrum Master is in charge of monitoring the effective adoption of those solutions so as to reduce the list of impediments. The Agile Coach helps teams to reflect on few priority points to work on, and encourage team members to agree with the idea of making a list of improvements to be implemented. Following the example of the User Story "As a User, I want to log in into the system so that I can enter the system," a mistake detected by the Agile Coach was that the students failed to consider storage of avatar' configurations, which is crucial to the load of the scene within the virtual world. The suggestion aimed at teaching the students to both improve the communication with the Product Owner and to apply elicitation requirements' techniques that they had learnt in a previous course.

To augment the monitoring process with performance indicators and help team members to adjust their estimates, *Virtual Scrum* supports a Burn-down chart, a graphic that shows the number of story points burnt during the Sprint. By using the Burn-down chart, the team knows both when the pending user stories in the Sprint Backlog should be completed and when the team deviates from the ideal effort at a specific moment. Figure 6 shows the Burn-down chart that represents the estimated story points ("Ideal Tasks Remaining") and the burnt story points ("Actual Tasks Remaining") during a Sprint in the *Universidad3D* project.

Figure 6. Burn-down chart in Virtual Scrum



Delivering the Product and Assessing Students

At the end of the course, the teams present the final integrated product to the Product Owner, who is in charge of assessing the way students followed Scrum during the development of the user stories. In this context, there is no formal final exam, and the students' grades are calculated by the average between two criteria. Firstly, we consider that the individual mark is the same that the team mark but individual, which is determined by the number of user stories accomplished in the Product Backlog, the quality of the software and documentation developed, the fulfillment of releases and Sprint plans, and the professor's evaluations on students' cooperative work, maintenance of the Sprint Backlog, and meeting of deadlines. Secondly, we determined the individual mark by the compliance with the process, interaction with tools, artifacts fulfilled, lines of code implemented, worked hours and number of bugs solved, among others. In order to avoid subjectivity, we revise students' pieces of evidence associated to each task and assess whether each piece is satisfied or unsatisfied.

As regards the students' compliance with the Scrum process, we assess the accomplishment of the recommended and reliable software engineering practices performed during the course, by utilizing the Capability Maturity Model Integration (CMMI) framework. Particularly, we used CMMI for development version 1.3 (CMMI-DEV 1.3), which aims at providing software organizations with superior project performance, client satisfaction, and quality of products and services. For example, for each CMMI practice we consider it as satisfied if there is significant evidence, such piece of code, document, screenshot and e-mail, among others, that Scrum practices performed show that the CMMI practice is accomplished. To perform the assessment, in a previous work (Soria, Rodriguez, & Campo, 2012) we designed a mapping between CMMI and Scrum practices that stems from the proposals in the works of (Diaz, 2009; Fritzsche & Keil, 2007; Marcal, 2008; Pikkarainen & Mantyniemi, 2006).

For instance, (Diaz, 2009) shows an empirical mapping in the context of Project Planning, Project Monitoring and Control, and Requirement Management. A general mapping between Scrum and CMMI level 2 and 3 is presented in (Fritzsche & Keil, 2007). The work presented in (Marcal, 2008) shows how Scrum allows achieving goals related to Project Planning, Project Monitoring and Control, Integrated Project Management and Risk Management. Finally, a mapping between Scrum and practices related to Requirement Management, Engineering process areas and Project Planning is presented in (Pikkarainen & Mantyniemi, 2006). Based on the aforementioned works, the coverage of the Scrum practices has been defined as the evaluation criterion for assessing students' performance. For each User Story, a practice is considered covered if there is at least a software artifact, i.e. an artifact defined in the development process designed for the capstone project, in the development repositories that evidences the practice had been completed.

To assist the professor in the students' assessment, *Virtual Scrum* provides the Task Board to evaluate the progress of the User Stories, Planning Poker View to evaluate students' estimates, Burndown chart and Daily Meeting View to evaluate students' performance and meeting of deadlines, and the Sprint Retrospective artifacts to evaluate the quality of products and documents completed by the students. In the light of the above, we believe that *Virtual Scrum* will help students improve their learning of Scrum practices and get a concrete outlook of how to setup a suitable Scrum-based team room to develop a capstone project.

Lessons Learned and Recommendations for Similar Courses

After three years of teaching software engineering to undergraduate students through capstone projects, we have learned a number of lessons

about how to efficiently teach Scrum to students organized in agile teams. Our experience indicates students could effectively both acquire certain non-technical skills and perform practices related to quality assurance, monitoring and control, and estimation of user stories by means of the inclusion of both the Agile Coach and *Virtual Scrum* in the training model. In this light, it can be stated that the Agile Coach helps the students meet deadlines with high-quality processes and internalize the concept of an agile team (Soria, Rodriguez, & Campo, 2012). Furthermore, *Virtual Scrum* is suitable for teaching students how to implement different artifacts in a Scrum Team Room, and to effectively perform the practices throughout the Scrum process (Rodríguez, Soria, & Campo, 2012b). In both cases, we carried out controlled experiments with undergraduate software engineering students that were described in the works. Additionally, we found that the tool provides effective communication among developers and coordination mechanism among Scrum artifacts optimizing the synchronization of the project along the course. In the following paragraphs, different recommendations are discussed.

Avoiding Falling in Waterfall-Like Issues

In our first implementation of the course based on RUP, students usually ends up falling in a waterfall-like process, even though RUP encourages the overlapping of phases. As RUP concentrates students' efforts on eliciting all the requirements during the Inception and Elaboration phases, this resulted in inevitable delays in the early stages of the process. These delays prevented students from having enough time to design and run test cases before delivering the product, as they focus on correcting the mistakes made in early phases of the RUP process. As a consequence, the test cases are rarely run and many requirements fail to be implemented, which results in a weak coverage of the practices related to design and implementation, verification, integration and deployment of the product.

By moving to Scrum, most of these aforementioned practices showed improvements because of the iterative life cycle that aims to work on all the aspects of software development during a Sprint. However, some students still misunderstood the concept behind the "done criteria" by assuming that a User Story is completed without fulfilling the testing stage. The concept of "done criteria" establishes that a User Story is tagged "DOING" until all the pre-established test cases have been not implemented (Williams et al., 2011).

To help students tackle this issue, the Agile Coach should take the responsibility of enforcing the concept of "done criteria," which is essential for ensuring that the User Stories developed are robust enough to handle alternative flows and/or error handling, and preventing students from misunderstanding Scrum. That is to say, the Agile Coach prompts students to complete the testing process before considering a User Story as "DONE." Rather than tracking what percentage of a new User Story is completed by a team member, the Agile Coach uses a binary "all or nothing" tag for tracking User Stories completion. In this line, we found that the Agile Coach enforces the concept of "done criteria" by making use of *Virtual Scrum* to facilitate transparency and visualization of progress within the teams, aiding students to keep the traceability of the User Stories across the Task Board.

Avoiding Following a Plan Instead of Responding to Change

As RUP relies on a rigid definition of plans, students, who usually lack the skills to describe all activities they need to undertake upfront, have to design and then follow a purely devised plan to develop user requirements. Given that the plan

diverges from actual execution, a low coverage of the practices related to establishment and commitment is achieved in our initial version of the training model. In this context, obtaining commitment from all team members in this context is an issue as only some of them take part in the planning session.

Even after moving to Scrum, these problems remained; the students misunderstood how to respond to changes and thought that planning is unnecessary for agile development. The Agile Coach is central in tackling this hindrance by clarifying misconceptions, as well as guiding and monitoring achievement plan commitment and goals. One of the techniques that foster commitment is Planning Poker, in which each student must participate in the estimation of the work to be done. Most the students agreed or somewhat agreed with the idea that *Virtual Scrum* facilitates the implementation of the Planning Poker technique. This opinion may be attributed to the idea that students worked in a distributed environment and found the tool viable to estimate user stories. It is worth clarifying that students were familiar with the technique and we aimed to evaluate the impact of *Virtual Scrum* on the performance of Planning Poker.

To help students keep the plan up-to-date, the Agile Coach should encourage them to analyze the Burn-down chart at the end of each Sprint, so that they can revise and adjust the backlog estimates based on the team velocity. In this context, *Virtual Scrum* makes both individual and group work permanently visible, which allows students to trace the progress of the team, and the Scrum Master to follow each member' progress. It is possible to observe that the display of team's metrics in the 3D metaphor resulted in a useful aid for self-reflection.

Avoiding Focusing on Dates Instead of Quality

The plan-driven property of RUP emphasizes the definition of deliverables, which students misunderstand as a need to meet deadlines, sometimes disregarding the RUP process. In this context, students tend to develop User Stories as fast as possible and skip recommended software engineering practices, resulting in low quality products. Consequently, during subsequent Sprints, most the time is devoted to fixing faulty User Stories rather than developing new ones, which eventually cause delays in the releases.

When RUP was replaced by Scrum, this situation failed to improve since practices such as building plans and designing software were underestimated by students. Due to a lack of assistance in a disciplined use of Scrum, most of the students misconceived essential practices as evident in assertions such as "documentation is not necessary," "design is too hard to achieve" and "planning is a waste of time." Documenting, designing and planning are mandatory in software engineering regardless the methodology.

To balance discipline and agility along the tracking of the capstone project, the Agile Coach should encourage students to focus on developing the User Stories by performing recommended and reliable practices. Thus, the Agile Coach should prevent accepting to subsequent Sprints User Stories that fail to be implemented with the required quality. To guarantee this, we incorporate these high-quality practices into the aforementioned "done criteria" as acceptance conditions for the delivered functionality.

In this context, in order to facilitate tracking of practices, *Virtual Scrum* guarantees the availability and accessibility of software artifacts in

each meeting. Moreover, *Virtual Scrum* simplifies Daily Meetings among geographically distributed students. We found *Virtual Scrum* suitable to help both students, enhancing their comprehension of Scrum, and the Agile Coach in monitoring the agile teams. From a conceptual point of view, the students stated that *Virtual Scrum* allowed them to optimize understanding software engineering concepts and problem-solving skills necessary to succeed in professional contexts.

Avoiding Writing Enormous Documents of Requirements

When students run our first version of the model based on RUP, we found that they used to write down what the professor required, instead of writing what the professor needed as a customer. The students spent a lot of time generating detailed documentation of requirements, resulting in a weak coverage of the practices related with the verification and validation of the product at the end of the project. As a consequence, the training model yielded as a result low coverage of the practices related to the identification and analysis of risks, since regular reviews of processes were rarely performed. Unsurprisingly, this situation led to weak communication among the Product Owner and team members resulted in a low coverage in the validation of the product.

In this context, an iterative and incremental life-cycle leads student to start with only the User Stories planned to be implemented within a Sprint. Focusing on a limited set of User Stories, the Agile Coach encourages students to communicate with the Product Owner and, thus, the specifications of those User Stories are complemented with the information resulting from that conversation. The Agile Coach should periodically observe the working progress and requires test cases and design documentation. Moreover, Holding Daily Meetings allowed increasing the coverage of the aforementioned practices. The Weekly Meetings held with the Agile Coach helped teams to track and communicate noncompliance issues objectively, and ensure theirs prompt resolution. These factors resulted in more student commitment to run functional and non-functional test cases associated with a User Story, meeting successfully the "done criteria." Along this line, we found that these kind of check-point meetings facilitated internalization of the Scrum concepts, faster solutions of impediments, and guidance from the Agile Coach.

Furthermore, since *Virtual Scrum* supports the Task Board to display the User Stories, the professor, playing the role of Product Owner, can enter the virtual world, check the progress of the User Stories and answer team questions. Another advantage of using *Virtual Scrum*, over creating Scrum rooms within the university premises, is that students can also access data they need for each Sprint and exchange valuable information, vital in agile teams, without having to be in the same physical context.

To bring this section to an end, we summarize the main points. One interesting finding after our recent experience with our teaching model is a progressive increment in the coverage of all the CMMI practices by students. This increment was due to compliance with the done criteria, carefully guidance performed by the Agile Coach and improvements in project tracking. As a result, we have obtained a more homogeneous accomplishment of the software engineering practices and higher commitment among the students. Another important finding of the experience is that *Virtual Scrum* effectively supports the Scrum practices performed by both students, in the role of Scrum Team, and professors, taking the roles of Product Owner and Agile Coach.

FUTURE RESEARCH DIRECTIONS

Nowadays, there are several approaches that allow introducing agile practices within software engineering curricula. Our experience aligns with the idea of using capstone projects, which reduce

the gap between professional and training contexts (Mahnic, 2010; Mahnic, 2012a; Devedzic, 2011; Schroeder, 2012). The aim of some approaches is to maximize student learning experience by using games (Lynch, 2011) or 3D interfaces (Parsons, 2010) to teach agile concepts. Other approaches propose the use of teaching models, such as agile teaching models (Hon, 2004) or frameworks that facilitate the teaching of software development processes (Hazzan & Dubinsky, 2006). Our training model takes elements of the aforementioned projects and introduces the Agile Coach role and *Virtual Scrum* support to provide tutoring and training to undergraduate students. However, students learn in many ways, and thus, professors should know the students' profile, their strengths and weaknesses in order to help them to perform agile practices. As a result, student would be able to maximize their learning experience.

From the experience of our aforementioned training model, we believe that personalization of teaching is crucial. As professors, we are planning to concentrate on teaching Scrum, complemented to XP and Kanban, catering for individual learning characteristics of students. A personalized learning approach focuses on knowing the leaner in order to offer learning scenarios that result relevant and motivating. Then, the personalization tailors these scenarios to students' needs by harnessing technology and data from the context (Ally, 2004). However, it is necessary to identify only student generated information relevant for such personalization and include irrelevant anecdotic data (Antunes, 2010). To address this issue, we will need to know the student talents and difficulties to learn of agile practices (Dick, Carey, & Carey, 2005). This knowledge will allow us to adapt their teaching strategies to student preferences (Felder & Silverman, 1988).

In this context, a line of research worth pursuing further is the study of relationships between students' performance and their learning style within heterogeneous undergraduate courses. Taking into account different learning styles within heterogeneous undergraduate courses may have an impact on learning by fostering engagement, motivation and attentiveness (Coffield, 2004; Felder, & Spurlin, 1988; Hawk & Shah, 2007). For example, Layman et al (Layman, Cornwell, & Williams, 2006) proposes to explore personality types and learning styles. Other approaches (Limongelli, 2008; Popescu, 2009; Zaina, 2011) focus on addressing various learning styles to personalize Web-based teaching contexts. A first attempt to shed some light on personalizing the teaching of Scrum on a software engineering course was carried out in (Scott, Rodriguez, Soria, & Campo, 2013) as a step towards improving the teaching of software engineering practices. In this research, we evidenced relationships between the Felder-Silverman learning styles (Felder & Silverman, 1988) of students and how they perform Scrum practices. For instance, we found a considerable influence of the processing style on the way students estimate User Stories and follow the Scrum process, and the amount of time that the students spend on the development of tasks. These findings will serve as a cornerstone for further studies to determine how the learning styles of students can be used to build a customized environment for learning Scrum.

CONCLUSION

This chapter has presented a teaching model based on a combination between Scrum, Agile Coach's role and Virtual Reality. We have discussed the design and implementation of the teaching model for introducing agile software development in a software project, focusing on both improving the learning of professional software practices and maintaining the quality of software processes. We found that teaching Scrum software development is effective, if students are involved in the development of a project rather than in traditional of-the-book classes. Moreover, facing the software engineering problems in a controlled environment

gives students the required skills to work in professional contexts. We observed that this teaching strategy may facilitate the student's integration in the software industry.

It is possible to conclude that the introduction of the Agile Coach was an effective teaching strategy that allowed students to achieve higher coverage of the software practices. Furthermore, the iterative and incremental nature of the training model allowed students to experience all the aspects of software development along a Sprint. During this period, students had the opportunity to learn from their mistakes, adapt to the context, and concentrate on only the user stories for a specific Sprint. Additionally, the students revealed the benefits of using the Scrum model augmented with agile coaching, namely facilitated internalization of the Scrum concepts, faster solutions of impediments, and guidance by means of check-point meetings.

Besides, this chapter showed that *Virtual Scrum* is an alternative able to face challenges of implementing Scrum within university premises. This tool proved to be effective in enriching teaching strategies for training students in the use of valuable professional skills to work in professional contexts through a virtual world. We found that using *Virtual Scrum* was helpful in improving students' comprehension of the fundamentals of agile practices and principles of developing software with Scrum. It is worth noting that the tool outperformed students' expectations with regard to support for planning meetings, which increased students' commitment; and follow-up metrics, which allowed students to self-reflect on their performance in the Sprint Retrospective meetings. The students also provided constructive feedback on user interactions and traceability of the User Stories through *Virtual Scrum*. Based on this feedback, we intend to improve user interactions by upgrading the support of media aids, specially the avatar integration with current social networks. As for traceability of User Stories, we found that students preferred using 2D tools for dealing with configuration management rather than a 3D representation of the artifacts.

In the light of the above, as future work, we are planning to complement *Virtual Scrum* with conventional team tools and emphasize on holding software meetings, and displaying software metrics and coverage of the Scrum practices. Another line of future work will concentrate on applying profiling techniques to provide personalized assistance to students according to their learning style, and thus, improve students' learning experience within teaching contexts.

REFERENCES

Agile Manifesto. (n.d.). Retrieved from http://www.agilemanifesto.org/

Alegría, J. A. H., & Bastarrica, M. C. (2006). Implementing CMMI using a combination of agile methods. *CLEI Electronic Journal, 1*(1).

Alfonso, M. I., & Botia, A. (2005). An iterative and agile process model for teaching software engineering. In *Proceedings of 18th Conference on Software Engineering Education and Training*. Academic Press.

Ally, M. (2004). Foundations of educational theory for online learning. In *Theory and practice of online learning* (pp. 3–31). Athabasca University, Canada's Open University.

Anderson, T. (Ed.). (2008). *The theory and practice of online learning*. Athabasca University Press.

Antunes, C. (2010). Anticipating student's failure as soon as possible. In C. Romero et al. (Eds.), *Handbook of educational data mining*. CRC Press. doi:10.1201/b10274-28

Apelo, J. (2010). *Management 3.0: Leading agile developers, developing agile developers*. Addison-Wesley.

Atlassian. (n.d.). Retrieved from https://www.atlassian.com/software/jira

Barbosa, E. F., & Maldonado, J. C. (2006). Towards the establishment of a standard process for developing educational modules. In *Proceedings of the 36th ASEE/IEEE Frontiers in Education Conference*. San Diego, CA: ASEE/IEEE.

Boehm, B., & Turner, R. (2008). *Balancing agility and discipline: A guide for the perplexed*. Reading, MA: Addison-Wesley.

Brenner, L. A., Koehler, D. J., & Tversky, A. (1996). On the evaluation of one-sided evidence. *Journal of Behavioral Decision Making*, *9*(1), 59–70. doi:10.1002/(SICI)1099-0771(199603)9:1<59::AID-BDM216>3.0.CO;2-V

Callaghan, M. J., McCusker, K., Losada, J. L., Harkin, J. G., & Wilson, S. (2009). Teaching engineering education using virtual worlds and virtual learning environment. In *Proceedings of 2009 Conference on Advances in Computing, Control and Telecommunications Technologies* (pp. 295–299). Kerela, India: Academic Press.

Cano, M. D. (2011). Students' involvement in continuous assessment methodologies: A case study for a distributed information systems course. *IEEE Transactions on Education*, *54*(3), 442–451. doi:10.1109/TE.2010.2073708

CCSE. (n.d.). Retrieved from http://sites.computer.org/ccse/SE2004Volume.pdf

Coffield, F., Moseley, D., Hall, E., & Ecclestone, K. (2004). *Should we be using learning styles?* Learning and Skills Research Centre.

Cohn, M. (2004). *User stories applied for agile software development*. Reading, MA: Addison-Wesley.

Cohn, M. (2005). *Agile estimating and planning*. Upper Saddle River, NJ: Prentice Hall.

Coupal, C., & Boechler, K. (2005). Introducing agile into a software development capstone project. In *Proceedings of Agile Conference* (pp. 289–297). Academic Press.

Davies, S. (2009). Appointing team leads for student software development projects. *Journal of Computing Sciences in Colleges*, *25*(2), 92–99.

Devedzic, V., & Milenkovic, S. R. (2011). Teaching agile software development: A case study. *IEEE Transactions on Education*, 54.

Diaz, J., Garbajosa, J., & Calvo-Manzano, J. A. (2009). Mapping CMMI level 2 to scrum practices: An experience report. *Software Process Improvements*, *42*, 93–104. doi:10.1007/978-3-642-04133-4_8

Dick, W. O., Carey, L., & Carey, J. O. (2005). *The systematic design of instruction*. Pearson/Allyn & Bacon.

Dubinsky, Y., & Hazzan, O. (2003). Extreme programming as a framework for student-project coaching in computer science capstone courses. In *Proceedings of IEEE International Conference on Software: Science, Technology and Engineering* (pp. 53–59). IEEE.

Felder, R. M., & Silverman, L. K. (1988). Learning and teaching styles in engineering education. *English Education*, *78*(7), 674–681.

Felder, R. M., & Spurlin, J. E. (2005). Applications, reliability, and validity of the index of learning styles. *International Journal of Engineering Education*, *21*(1), 103–112.

Fritzsche, M., & Keil, P. (2007). Agile methods and CMMI: Compatibility or conflict? *e-Informatica. Software Engineering Journal*, *1*(1), 9–26.

Glazer, H. (2008). *CMMI or agile: Why not embrace both! (Technical Note, CMU/SEI-2008-TN-003)*. Pittsburgh, PA: Software Engineering Process Management, Carnegie Mellon.

Glazer, H. (2010). Love and marriage: CMMI and agile need each other. *CrossTalk*, *23*(1), 29–34.

Grenning, J. (2002). *Planning poker or how to avoid analysis paralysis while release planning*. Retrieved from http://renaissancesoftware.net/files/articles/PlanningPoker-v1.1.pdf

Hawk, T. F., & Shah, A. J. (2007). Using learning style instruments to enhance student learning. *Decision Sciences Journal of Innovative Education*, *5*(1), 1–19. doi:10.1111/j.1540-4609.2007.00125.x

Hazzan, O., & Dubinsky, Y. (2006). Teaching framework for software development methods. In *Proceedings of the 28th International Conference on Software Engineering* (pp. 703–706). ACM.

Hedin, G., Bendix, L., & Magnusson, B. (2005). Teaching extreme programming to large groups of students. *Journal of Systems and Software*, *74*(2), 133–146. doi:10.1016/j.jss.2003.09.026

Herbsleb, J. D. (2007). Global software engineering: The future of socio-technical coordination. In *Proceedings of Future of Software Engineering* (pp. 188–198). IEEE. doi:10.1109/FOSE.2007.11

Highsmith, J. (2004). *Agile project management-creating innovative products*. Boston: Addison-Wesley.

Hon, A., & Chun, W. (2004). Teaching agile teaching/learning methodology and its e-learning platform. *LNCS-Advances in Web-Based Learning*, *3143*, 11–18.

IBM. (n.d.). *Developerworks.* Retrieved from http://www.ibm.com/developerworks/rational/library/apr05/crain/

Isistan. (n.d.). Retrieved from http://isistan.exa.unicen.edu.ar/u3d/

Jorgensen, M. (2004). A review of studies on expert estimation of software development effort. *Journal of Systems and Software*, *70*, 37–60. doi:10.1016/S0164-1212(02)00156-5

Khan, F. A., Graf, S., Weippl, E. R., & Tjoa, A. M. (2010). Implementation of affective states and learning styles tactics in web-based learning management systems. In *Proceedings of the 10th IEEE International Conference on Advanced Learning Technologies* (pp. 734–735). IEEE.

Kniberg, H. (2008). *The manager's role in scrum*. Retrieved from http://www.scrumalliance.org/resources/293

Kruchten, P. (2003). *The rational unified process: An introduction*. Reading, MA: Addison-Wesley.

Kulpa, M., & Johnson, K. (2003). *Interpreting the CMMI*. CRC Press. doi:10.1201/9780203504611

Layman, L., Cornwell, T., & Williams, L. (2006). Personality types, learning styles, and an agile approach to software engineering education. *ACM SIGCSE Bulletin*, *38*(1), 428–432. doi:10.1145/1124706.1121474

Limongelli, C., Sciarrone, F., & Vaste, G. (2008). Ls-plan: An effective combination of dynamic courseware generation and learning styles in web-based education. In *Proceedings of the 5th International Conference on Adaptive Hypermedia and Adaptive Web-Based Systems* (pp. 133–142). Springer-Verlag.

Lutteroth, C., Luxton-Reilly, A., Dobbie, G., & Hamer, J. (2007). A maturity model for computing education. In *Proceedings of the Ninth Australasian Conference on Computing Education* (vol. 6, pp. 107–114). Australian Computer Society.

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 *Proceedings of Frontiers in Education Conference* (pp. F1H-1). IEEE.

Maher, P. (2009). Weaving agile software development techniques into a traditional computer science curriculum. In *Proceeding on 6th Conference on Information Technology: New Generations*. Academic Press.

Mahnic, V. (2010). Teaching scrum through team-project work: Students' perceptions and teachers' observations. *The International Journal of Engineering Education*.

Mahnic, V. (2011). A case study on agile estimating and planning using scrum. *Electronics and Electrical Engineering, 5*.

Mahnic, V. (2012a). A capstone course on agile software development using scrum. *IEEE Transactions on Education*, *5*(2), 99–106. doi:10.1109/TE.2011.2142311

Mahnic, V., & Hovelja, T. (2012b). On using planning poker for estimating user stories. *Journal of Systems and Software*, *85*(9), 2086–2095. doi:10.1016/j.jss.2012.04.005

Marçal, A. S. C., de Freitas, B. C. C., Soares, F. S. F., Furtado, M. E. S., Maciel, T. M., & Belchior, A. D. (2008). Blending scrum practices and CMMI project management process areas. *Innovations in Systems and Software Engineering*, *4*(1), 17–29. doi:10.1007/s11334-007-0040-1

Mathiassen, L., & Pries-Heje, J. (2006). Business agility and diffusion of information technology. *European Journal of Information Systems*, 116–119. doi:10.1057/palgrave.ejis.3000610

Miranda, E. (2001). Improving subjective estimates using paired comparisons. *IEEE Software*, *18*(1), 87–91. doi:10.1109/52.903173

Nerur, S., & Balijepally, V. (2007). Theoretical reflections on agile development methodologies. *Communications of the ACM*, *50*, 79–83. doi:10.1145/1226736.1226739

Oliveira, E., & Lima, R. (2011). State of the art on the use of scrum in distributed development software. *Revista de Sistemas e Computação*, *1*(2), 106–119.

Osorio, J. A., Chaudron, M. R., & Heijstek, W. (2011). Moving from waterfall to iterative development – An empical evaluation of advantages, disadvantages and risks of RUP. In *Proceedings of 37th Conference on Software Engineering and Advanced Application* (pp. 453–460). IEEE.

Paasivaara, M., Durasiewicz, S., & Lassenius, C. (2009). Using scrum in distributed agile development: A multiple case study. In *Proceedings of the 4th IEEE International Conference on Global Software Engineering* (pp. 195–204). IEEE.

Parsons, D., & Stockdale, R. (2010). Cloud as context: Virtual world learning with open wonderland. In *Proceedings of the 9th World Conference on Mobile and Contextual Learning* (mLearn 2010). Valetta, Malta: mLearn.

Pikkarainen, M., & Mantyniemi, A. (2006). *An approach for using CMMI in agile software development assessments: Experiences from three case studies*. University of Limerick Institutional Repository.

Popescu, E. (2009). Evaluating the impact of adaptation to learning styles in a web-based educational system. In *Proceedings of the 8th International Conference on Advances in Web Based Learning* (pp. 343–352). Berlin: Springer-Verlag.

Rodríguez, G., Soria, A., & Campo, M. (2012a). Teaching scrum to software engineering students with virtual reality support. In *Advances in new technologies, interactive interfaces and communicability* (pp. 140–150). Berlin: Springer. doi:10.1007/978-3-642-34010-9_14

Rodríguez, G., Soria, A., & Campo, M. (2012b). Supporting virtual meetings in distributed scrum teams. *IEEE Latin America Transactions*, *10*(6), 2316–2323. doi:10.1109/TLA.2012.6418138

Schroeder, A., & Klarl, A. (2012). Teaching agile software development through lab courses. In *Proceedings of IEEE Global Engineering Education Conference* (EDUCON) (pp. 1177–1186). IEEE.

Schwaber, K., & Beedle, M. (2002). *Agile software development wit scrum*. Upper Saddle River, NJ: Prentice Hall.

Scott, E., Rodriguez, G., Soria, A., & Campo, M. (2013). El rol del estilo de aprendizaje en la enseñanza de prácticas de scrum: Un enfoque estadístico. In *Proceedings of the 14th Argentine Symposium on Software Engineering*. Academic Press.

Shrivastava, S. V., & Date, H. (2010). Distributed agile software development: A review. *Journal of Computer Science and Engineering*, *1*(1), 10–16.

Soria, A., Rodriguez, G., & Campo, M. (2012). Improving software engineering teaching by introducing agile management. In *Proceedings of the 13th Argentine Symposium on Software Engineering* (pp. 215–229). Academic Press.

Sutherland, J., Ruseng Jakobsen, C., & Johnson, K. (2008). Scrum and CMMI level 5: The magic potion for code warriors. In *Proceedings of Hawaii International Conference on System Sciences, Proceedings of the 41st Annual* (pp. 466–466). IEEE.

Tan, C. H., Tan, W. K., & Teo, H. H. (2008). Training students to be agile information systems developers: A pedagogical approach. In *Proceedings of the 2008 ACM SIGMIS CPR Conference on Computer Personnel Doctoral Consortium and Research* (pp. 88–96). ACM.

Unity 3D. (n.d.). Retrieved from http://unity3d.com/

USVN. (n.d.). Retrieved from http://www.usvn.info/

Versionone. (n.d.). *State of agile development survey results.* Retrieved from http://www.versionone.com/pdf/2012_State_of_Agile_Development_Survey_Results.pdf

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.

Whitehead, J. (2007). *Collaboration in software engineering: A roadmap*. Santa Cruz, CA: University of California.

Williams, L., Brown, G., Meltzer, A., & Nagappan, N. (2011). Scrum + engineering practices: Experiences of three Microsoft teams. In *Proceedings of International Symposium on Empirical Software Engineering and Measurement* (pp. 463–471). IEEE Society.

Xwiki. (n.d.). Retrieved from http://www.xwiki.org/

Zaina, L. A. M., Bressan, G., Rodrigues, J. F., & Cardieri, M. A. C. A. (2011). Learning profile identification based on the analysis of the user's context of interaction. *IEEE Latin America Transactions*, *9*(5), 845–850. doi:10.1109/TLA.2011.6030999

KEY TERMS AND DEFINITIONS

Agile Coach: A person who provides directions to the project and is responsible for removing any process impediments.

Capability Maturity Model Integration (CMMI): A framework which consists of a set of best practices that address the development and maintenance of products and services.

Capstone Project: A cooperative assignment in which undergraduates are expected to integrate the course contents in a quasi-real world experience.

Learning/Teaching Strategies: Recommended practices to improve learning and teaching experiences.

Rational Unified Process (RUP): It is a software engineering process, aimed at guiding software development organizations.

Scrum: An iterative and incremental methodology that organizes projects to make them manageable for small, self-organized and cross-functional teams.

Software Engineering Education: A set of university courses to prepare students to be professional software engineers.

Virtual Reality: A computer-simulated environment that can simulate physical presence in places within real or imagined worlds.

ENDNOTES

1. A capstone project is a cooperative assignment in which undergraduates are expected to integrate the course contents in a quasi-real world experience (Mahnic, 2012a).