Improving Software Engineering Teaching by Introducing Agile Management

Álvaro Soria Marcelo R. Campo* Guillermo Rodríguez*

ISISTAN Research Institute, UNICEN University
Campus Universitario, (B7001BBO) Tandil, Buenos Aires, Argentina.
**Also CONICET-Argentina*

{asoria, mcampo, grodri }@exa.unicen.edu.ar

Abstract. One of the main goals of Software Engineering (SE) courses is to train students to face problems that occur in professional contexts. Thus, software engineering courses have to be continuously reoriented to cater for the demands of the software industry without neglecting academic quality. The widespread use of Scrum, an agile approach to software development, provides SE professors with a suitable option for teaching students good practices of current software development. In the present paper, we introduce a teaching model based on a combination of Scrum and Agile Coaching. This innovative model, which has been contrasted with RUP (Rational Unified Process) and assessed, using CMMI (Capability Maturity Model Integration) as a reference, is a result of an evolutionary process in which several improvements were conducted during the academic period 2008/10. Results show that this agile approach allows students to develop software achieving high levels of CMMI maturity.

Keywords: Software Engineering Education, Scrum, CMMI, Agile Coaching.

1. Introduction

In recent years, software industries have grown rapidly and they are demanding for skilled software engineers in a challenging context in which the increasing complexity of software development, constant changes in system requirements, and mobility of developers take place. Thus, showing students good practices of software development is crucial, so that they are capable of ongoing success in the software engineering field.

As a consequence, professors have to design Software Engineering (SE) courses including several aspects to teach students how to deal with current threats present in large software projects [1]. To do so, we structured a Software Engineering course following CMMI [16]. We utilized CMMI for development version 1.3 (CMMI-DEV 1.3)[1] that is focused on product and service development. CMMI is a framework that covers a set of practices to implement mature and high-quality software development processes. Our initial implementation of CMMI consisted in following the Rational Unified Process (RUP) [15] to support the project software processes. To run a software project, we asked students to follow RUP for achieving the good practices proposed by CMMI.

However, teaching SE to students running a software project following RUP suffers from several drawbacks. As it is a plan-driven development framework, RUP requires the association

[1] CMMI for Development version 1.3. Technical Report. Software Engineering Institute, Carnegie Mellon. November, 2010. http://www.sei.cmu.edu/library/abstracts/reports/10tr033.cfm

of project milestones with specific dates. This makes students focus on reaching deadlines and delivering the agreed milestone, skipping activities of the RUP workflow. In addition, even though RUP encourages the overlapping of phases, it is inevitably for students to fall in a waterfall-like process [24]. Thus, it is difficult for the inexperienced students to detect mistakes made in early stages before the development process reaches last stages. Finally, as in the planning of the scope and the project milestones some students do not take part in; it provokes a lack of commitment in the rest of them. Since SE is a social activity, these aspects are cornerstone of its reality and crucial to professional education and training.

To tackle these problems in an academic environment, Agile Methods (AM) emerge as a much more viable way to implement the main CMMI practices with RUP. AM promote a highly iterative work model with the aim to produce high-quality software and allow quick adaptation to changing requirements [11]. A typical concern among software development companies is the need for strategies which help them to be well positioned in the software market. For this reason, the combination between agile approaches and CMMI seems to be a suitable alternative to develop mature software in a challenging context. Agile values ensure success and quality, making AM ideal partners of CMMI. As a result, companies are able to deliver a high-quality product following a mature process in continuous improvement and optimization [8, 9, 12].

Agile software development has received significant academic attention because of its widespread application in the commercial world [5, 6, 7, 10, 14, 19]. Out of the various agile approaches, Scrum has gained wide acceptance because it concentrates on managing software projects and includes monitoring and feedback activities [3, 19]. These features allow students to acquire skills beyond technical and scientific scenarios, such as teamwork-related abilities. In an educational context, these aspects are welcome because they enable students to get acquainted with agile methods and, at the same time, provide mechanisms for evaluating individual agile concepts.

Along this line, this paper presents a teaching model based on agile practices in a CMMI context. The aim of the model is to maximize the strengths of both discipline and agility to improve software engineering teaching. Two main aspects have been considered to implement AM in a teaching context: the agile process and the Agile Coach. Here, the agile process is supported by Scrum and the Agile Coach role is played by the professor, who is responsible for coaching the teams. During the academic years 2008, 2009 and 2010, we implemented this teaching model in the Software Engineering course of the UNICEN University. In order to measure its effectiveness, we assessed the impact of the performance of students on the coverage level of the CMMI practices. The results have shown that a balance among Scrum, the Agile Coach role and CMMI is more appealing to students so that they can obtain a higher coverage of CMMI practices than when using CMMI with RUP.

The remainder of the paper is organized as follows. Section 2 describes the foundations of our approach. Section 3 presents our agile-based teaching model. Section 4 reports the case-studies and outlines the most important lessons learned and limitations of following the teaching model. Section 5 reviews some related works and section 6 concludes this research and indentifies future lines of work.

2. 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 [15]. 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 [15].

CMMI is often misunderstood [15] as being required massive documentation, many layers of personnel and the use of a rigid waterfall life cycle. However, by following AM it is possible to obtain maturity levels with less overhead and effort [2, 11]. 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 [8, 9]. The value from AM can only be obtained through disciplined use. 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 [7] [18]. Scrum is an agile methodology that organizes projects into small, self-organized and cross-functional teams, called Scrum Teams [30].

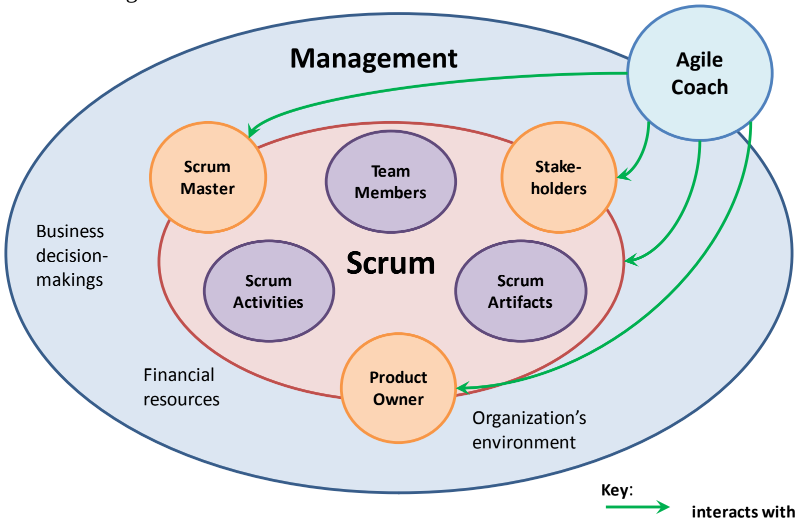
Work in Scrum is organized and prioritized according to the Product Backlog. This is the master list of the desired features in the product. The backlog items are called user stories, which are provided by a domain expert called Product Owner. A user story describes the scenario in which a player wants to log into a virtual world. The user story describes a desired functionality involving role ("As…"), product features ("…I want to…") and the benefit provided to the user ("…so that…"). A sample user story could be the following: "*As a User, when I log out the virtual world I want to save my interaction so that I could log in again and be in the log-out place*".

The user stories in the Product Backlog are prioritized by the Product Owner, who represents the customers' interests, and grouped into short iterations called Sprints. For each sprint, a subset of the user stories in the Product Backlog is selected and organized in a Sprint Backlog. During the Sprint, the Scrum team takes user stories from the Sprint Backlog and develops and tests them. These activities are coordinated by a management representative, called Scrum Master, who enforces the Scrum practices and helps the team make decisions or acquire resources as needed.

Anyhow, not all the aspects required in the agile world are tackled by Scrum. Figure 1 shows Scrum in an agile context of a software organization. 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. For this reason, it is necessary to include the role of an Agile Coach into a software organization. The main goal of the Agile Coach is to enable the team to solve its own problems and come up with its own insights of products [13]. For instance, the Agile Coach is in charge of coaching the team members, enabling them to resolve their own problems, and assisting the Scrum Master in removing organizational impediments.

Regarding a teaching context, Scrum enables the students to have a better teamwork environment and a better communication that results in high-quality products [7]. Along with Scrum, the Agile Coach role is an important aspect that has to be incorporated in a SE course, so that the professor can coach the students in order to help them face the diversity of facets of a product development. Following this line, the next section presents a teaching model that consists of the implementation of CMMI using Scrum complemented with the Agile Coach role.

Figure 1. Scrum in the agile world.



3. Agile-based teaching model

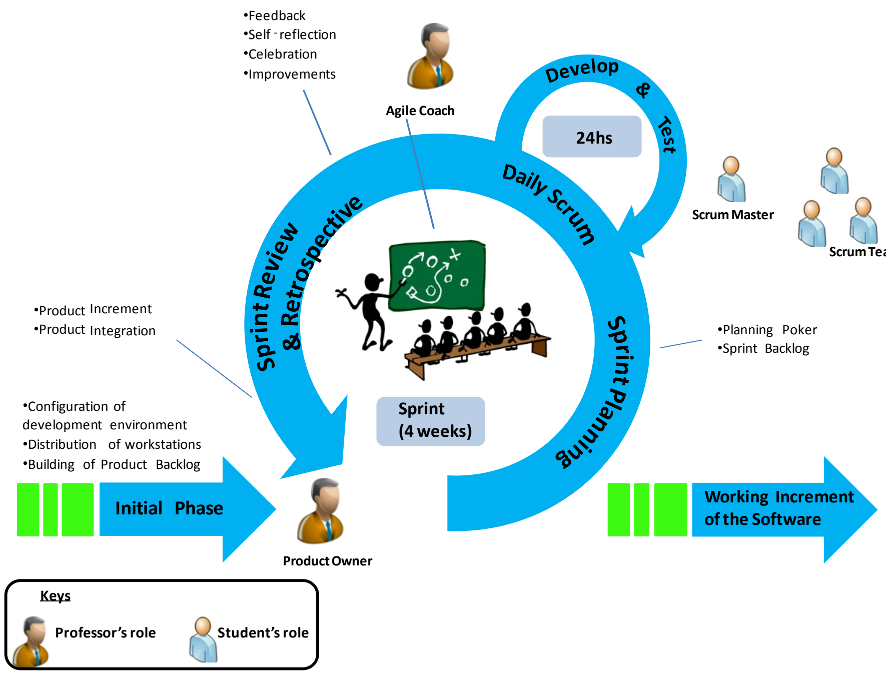
In the current curriculum of System Engineering studies at UNICEN University, Scrum has been included into the Software Engineering course. Before attending this course, students attend the Introduction to Software Engineering course in which students are also trained in understanding the CMMI process areas, implementing these areas with Scrum, and using the development environment. In addition, the professor complements the course with anecdotes and previous experiences in companies in order to emphasize the pedagogical techniques. Simultaneously, the software assets related to each process area are prepared to be used in the next course, which this paper focuses on. An asset is an artifact that relates to describing, implementing, and improving processes (e.g., polices, measurements, process descriptions, documents, and process implementation support tools).

Figure 2 illustrates the teaching model used by the professor to run the course. Our teaching approach is oriented to simulate a software organization. We assume that students have acquired the required knowledge during the previous course. The students play the Scrum Master and the Scrum Team roles, and are responsible for developing and testing the user stories. As professors, we play two roles: Product Owner and Agile Coach simulating a realistic environment [19]. The Product Owner owns the Product Backlog and helps the teams clarify the user story's specifications. Also, she is responsible for validating the final product. The Agile Coach encourages the teams throughout the Scrum process by clearing the team's obstacles and emphasizing the use of tools to maintain the traceability of the user stories. It is important to note that the students playing the role of Scrum Masters are evaluated in how well they (a) protect the Scrum Team, (b) ensure that the Scrum process is followed in terms of values, practices and rules, (c) remove impediments, and (d) bridge the gap between the Product Owner and the Scrum Teams. Instead, the Agile Coach is not involved in the project and is a transitional role until the Scrum Teams grow their own coaching capability. She acts a consultant and a trainer in agile methods.

The model consists of an iterative and incremental life cycle based on Scrum and Agile Coaching. In the Initial Phase, which represents the Sprint 0, all the setup of the development environment is carried out and the students checks that the workstations are working with all the required features. On the other hand, the user stories are defined and loaded into the Product Backlog. This artifact is supervised by the Product Owner, who prioritizes and negotiates the user stories for the Sprint with the team according to risk levels and importance to the project.

The next phase is the Sprint Planning which consists in planning and estimating the work to be done during a Sprint. Before the Sprint Backlog is defined, the estimates of user stories are obtained in order to assure that sum o story points of all user stories in the Sprint Bakclog corresponds to the velocity of the Scrum Teams. Each Scrum Team estimates the complexity of the user stories by using the Planning Poker technique proposed by Cohn [4]. The estimates are constrained to specific predefined values of 0.5, 1, 2, 3, 5, 8, 13, and 20. Once the Sprint Backlog is defined, the selected user stories to be done in the Sprint are decomposed into simpler tasks using the WBS (Work Breakdown Structure) technique. Also, these tasks are estimated too. Table 1 shows an example of the estimation of tasks associated to the sample user story presented in the previous section. Then, the user story has been divided into three tasks: a) log out the virtual world, b) save the user interaction and c) retrieve all the actions. This division occurs because the user story has obtained a high value in the planning game (i.e. 8 or higher). As a consequence, the students of the Scrum Team break the user story down into constituent tasks and organize themselves to perform each one. Based on the estimated complexity, the students estimate the numbers of hours that each task may take. In this phase, the Product Owner works closely with the Scrum Team to provide clarification and approval on user stories. As a result, the practices related to project planning defined by CMMI are accomplished.

Figure 2. Overview of the teaching model based on Scrum and the Agile Coach.



Once the Sprint Planning has finished, the Scrum team is ready to start developing the user stories during 4 weeks. Each user story goes through a miniature process consisting in analyzing, designing, building and testing. The assets generated in these stages should accomplish the CMMI practices related to requirement management, technical solution, software verification and the quality assurance of the process and the product.

In the light of the above, the concept of *done* arises. To consider a user story done, it must go through the miniature process in which the assets of CMMI studied in the previous course are fulfilled. For instance, Table 1 shows when the teams document the user story requirement, the design report, the code and the reports of testing and metrics along the Sprint.

Each day of the Sprint, a Daily Meeting is held to give place to fluent feedback. These meetings enable students to communicate the work done and track the progress. The students answer three questions: What have you done since the last Scrum meeting? What are you planning to do between now and the next Scrum meeting? What got in your way of doing work? These questions allow the students to track the progress of the user stories of the Sprint. As a consequence, the Daily Meetings allow the students to accomplish the practices related to project monitoring and control, risk management and peer review.

Twice a week, a Weekly Meeting is held between the professor, who plays the role of the Agile Coach, and each Scrum Team. The purpose of the meeting is monitoring the students' performance in each stage of the miniature process. During the Weekly Meetings, the Agile Coach encourages students to show architectural designs, user story specifications and other relevant documentation to give them feedback and lead them in the right way. As input to the meeting, the Scrum Master along with the Scrum Team fulfills a template containing the activities done, problems and impediments found, and the team commitment for the next weekly meeting. Based on the template information, the Agile Coach gives feedback to reinforce the student coaching. In particular, the Agile Coach provides support and assistance to the teams and the Scrum Masters. With this approach, the students receive feedback soon and more commitment from them is obtained.

Some examples of the Agile Coach's assistance are shown in the bottom row of Table 1. This kind of assistance should not interfere with the self-organization of the team. For instance, the Agile Coach may suggest "Revise the avatar's configuration because something is missing" (column week 1 and row Agile Coach's assistance in Table 1). This suggestion gives feedback about a possible problem without pointing out the specific solution to the problem. To follow the suggestion, the Agile Coach, who should not interrupt the process to correct deviations, let the students the responsibility for getting more information and clarification from the Product Owner. If students get poor information, the requirement analysis stage will be weak. As a consequence, this will strongly affect the next stages of the miniature process. If the team does not realize the underlying problem, the Agile Coach will teach possible corrective actions to the problems during the Sprint Retrospective Meeting.

At the end of a Sprint, each team has to integrate and deliver a single product increment covering CMMI practices related to the integrated product management. In this phase, the Product Owner is responsible for validating the product and giving feedback to the students. In this scenario, the Sprint Review practice of Scrum is carried out. If there are user stories undone when finishing the Sprint, they are re-estimated to be performed during the next Sprint. After the integration of the teams' products, a new meeting is held in the Sprint Retrospective phase. In this meeting, the Agile Coach informs feedback on the quality of products, self-reflections on team performance and comparison of estimated and adjusted efforts. For instance, the values obtained in the "adjusted estimate" field in Table 1 are reviewed and each team reflects and learns from the past experience to improve itself in the next Sprints.

Table 1. Example of the development of a user story by following the teaching model

User Story	Tasks	Initial Estimate	Adjusted Estimate	Week 1	Week 2	Week 3	Week 4
US 1	-Log out the virtual world -Save the user interaction -Retrieve all the actions	30 20 45	48 35 60	• Sprint Backlog Maintenance • Specification and Validation of the user stories	• Flow charts • Workflow diagrams • High-level architecture design	• Low-level design • Implementation • Code documentation • Test case design	• Test case performance • Bug report • End-user testing • Metrics and audits
Agile Coach's assistance				Revise the storage configuration because something is missing	I cannot understand your design. More details are needed	Some bugs may not be considered by the test cases	I am not able to measure the team performance

During the meeting, the Agile Coach identifies corrective actions to solve a particular problem in the miniature process. For dealing with the problems, the Agile Coach makes suggestions to the students so that they can accomplish the software engineering practices. Following the example in Table 1, the mistake was that the students did not consider the storage of the avatar's configuration, which is crucial to the functionality of the system. The suggestion aimed at teaching the students to both improve the communication with the Product Owner and to apply elicitation requirements' techniques that they have learnt in the previous course. At the end of the meeting, each team, coached by the Agile Coach, implements the identified actions for the next iterations.

At end of the course, the teams show the final integrated product to the Product Owner. The final product is the result of integrating each team product. Upon approval of the Prodcut Owner, the Agile Coach carries out the assessment of the software assets that complement the delivered product.

4. Case-studies

In order to evaluate the effectiveness of our agile teaching model, we carried out three experiments between 2008 and 2010 in the context of the Software Engineering course of the Systems Engineering BSc program at the Faculty of Exact Sciences (Department of Computer Science - UNICEN, Argentine). In 2008, 63 students took part of this experience and they were asked to follow RUP. In 2009, we replaced RUP by Scrum, which was run by 56 students. Finally, 61 students were enrolled in 2010 and we decided to incorporate the role of the Agile Coach in the teaching model so as to reinforce the Scrum implementation in 2009. In total, 160 students were enrolled; of whom 136 were men (85%) and 24 were women (15%). The students attending each course were divided into groups of 7±2 members. To simulate a real work environment, the students were randomly organized so that it is possible to find incompatible personalities. Each group was asked to follow the corresponding teaching model to complete the assigned requirements for a given project along the course.

The software project consisted in a virtual world game of the UNICEN called *Universidad3D*[2]. *Universidad3D* allows users to navigate the campus facilities and interactively learn about academic offerings. The core of the system is a Java 3D engine with features for

[2] http://isistan.exa.unicen.edu.ar/u3d/

scene definition, animation and navigation. *Universidad3D* is designed as a multi-tiered client-server architecture supporting chat, e-mail and forum mechanisms for communication between players. The baseline of the system implementation consisted of 190 Java classes (approximately 13 KLOC). For the experiments, a set of requirements with similar complexity were given to each team.

As development environment, the students interacted with a set of open source and academic-licensed tools. All of the teams received training in the use of these tools. The main tool selected for developing the user stories was the Integrated Development Environment (IDE) called Eclipse[3], in which most of the development tools integrate with. As regards communication, the teams used Google Groups, chats and face-to-face meetings. To deal with source code management, SVN[4] was given to the students. For automatic building and continuous integration the teams used Hudson[5]. To test the source code, the students used JUnit[6] as the testing framework. The issue tracker Mantis was used to manage the project. Finally, the PAL, which contains all of the organization assets, was stored in a XWiki[7].

The experiments aimed to assess the quality of the development processes at the end of each course. This quality is determined by the coverage of the software practices defined by CMMI. The coverage of practices was measured as follow: for each user requirement, a CMMI practice was considered covered if there was at least an asset in the Process Assets Library (PAL) evidencing that the practice has been accomplished by practices proposed in our teaching approach. Also, partially covered practices were taken into account so as to consider the work done by the students. In this context, a practice is followed by the students, but it does not follow a formal method as required by the CMMI.

To smooth the progress of the comparison between RUP and Scrum, we have established a mapping between CMMI covered by RUP and the Scrum practices. The mapping stems from proposals in the works of [7, 12, 22, 27]. For instance, [7] shows an empirical mapping in the context of Project Planning, Project Monitoring and Control, and Requirement Management. In [12] a general mapping between CMMI level 2 and 3, and Scrum is presented. The work presented in [22] shows how Scrum allows the achievement of practices related to Project Planning, Project Monitoring and Control, Integrated Project Management and Risk Management. Finally, in [27] a mapping between Scrum and practices related to Requirement Management, Engineering process areas and Project Planning is presented. Table 2 shows the empirical mapping used to perform the CMMI assessment and the adaptation of the software assets to a Scrum context.

4.1 The Agile-Based Approach Performance

In this section we analyze the students' performance in each variant of the teaching model across three case-studies. Figure 3 summarizes the coverage metric for the evaluation of 3159 software assets corresponding to the three courses. Overall, the results show that students reached the highest coverage of CMMI practices of the experiments with the inclusion of the role of Agile Coach in the teaching model. In the light of those results, it can be stated that this role helps students meet deadlines with high-quality processes and internalize the concept of an

[3] http://eclipse.org/
[4] http://subversion.tigris.org/
[5] http://java.net/projects/hudson/
[6] http://www.junit.org/
[7] http://www.xwiki.org

agile team. As we have stated, the RUP model consists in a strong establishment of a plan and definitions of deliverables. This characteristic is represented by the high coverage of the practice related to the establishment and maintenance of the estimates of the project (P1) for the RUP experience as shown in Figure 3. As a downside, we found a decrease in the coverage of practices related to establishment and maintenance of the commitment to the plan (P2 and P3). However, Scrum by itself was unable to deal with this problem, because the students misunderstood some of the Scrum principles assuming that it was not necessary to do planning in an agile context. Several students had misconceptions concerning Scrum, namely “planning is a waste of time”, “documentation is not necessary” and “design is too hard to achieve”. This led to a weak coverage of the P1 practice. When incorporating the role of Agile Coach in 2010, the professor asked the students for documents and other evidence to accomplish the practices related to planning and oversight activities. Thus, the coverage of the practice P1 in 2010 increased to reach almost the P1 coverage in 2008.

Even though RUP encourages the overlapping of phases; in the 2008’s study, it led students to a waterfall-like process. Thus, a delay in the early stages of the process was inevitable. As a consequence, we found a weak coverage in the practices related to the design and implementation (P4), verification (P7), integration (P10) and deployment (P11) of the product as it is shown in Figure 3. This evidence is consistent with our hypothesis that a plan-driven model makes students focus on reaching deadlines instead of following the activities of the development process. Most of these practices showed improvements when Scrum was implemented in 2009. The reason for this improvement was that the students exercised all the aspects of software development during a Sprint. It is worth noting that the increase of coverage of these practices after the incorporation of the Agile Coach in 2010 stems from the coaching of the Scrum Masters by periodically observing the issue tracker and presenting the Scrum practices uncovered, partially covered and fully covered during the weekly meetings.

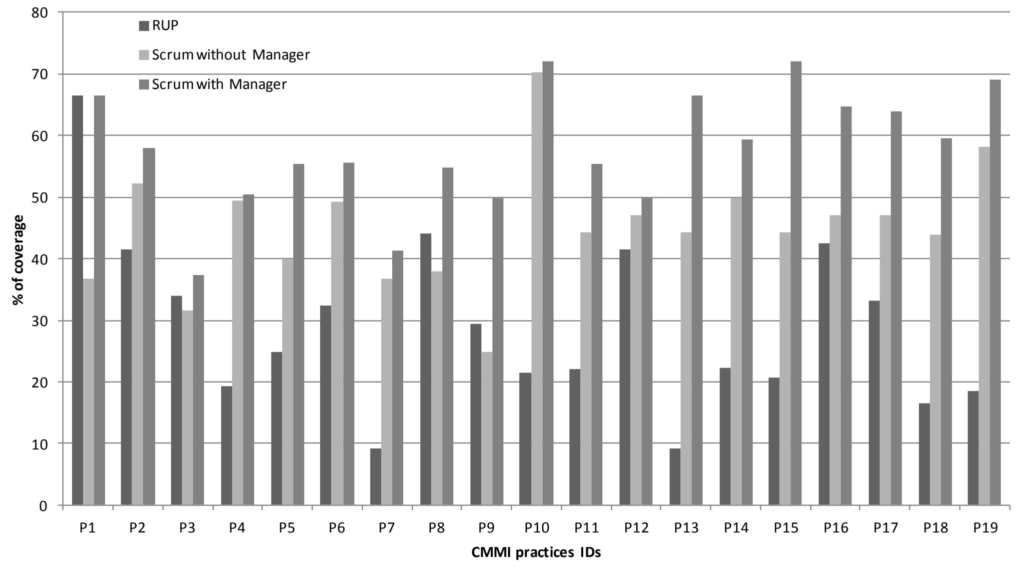
On the other hand, insufficient communication with Product Owner and loose habit of documentation resulted in a weak coverage of the practices related with the validation of the product (P8 and P9) in 2008 and 2009. Regarding the preparation for validation (P8), its coverage hardly overcomes 40% in 2008. Surprisingly, we found even less than 40% of this practice during the 2009’s study. This low coverage stems from the weak communication on the user stories’ evolution between the Product Owner and team members in the 2009’s study. As a consequence, we noticed that the Scrum Masters needed to be coached in playing their role and in the interaction with the Product Owner. This interaction is crucial to identify inconsistencies and impediments in the development of the software product. Similar to P8, the coverage of the practice P9 reached its highest value in the 2010 experience. As a side-effect, we found that the improved interaction with the Product Owner produced an increase in the coverage related to practices of project tracking and risk management (P12-P17) in the 2010’s study.

During the implementation of RUP, we found that the lack of a definition of the criteria to consider a user requirement done and the delay in early stages of the process resulted in a weak coverage of the practices related to quality assurance and noncompliance communication (P18 and P19) as it is shown in Figure 3. By including Scrum, the coverage of these practices improved because of the iterative life cycle that aimed to work on all the aspects of software development during a Sprint. However, some students still misunderstood the done criteria by assuming that a user story was done without the test-cases. The role of the Agile Coach allowed the professors to ask the students the test cases and the Product Owner’s approval for the user stories.

Table 2. Mapping between Scrum and main CMMI practices.

ID	CMMI Practices	Scrum Practices
P1	Establish and maintain the estimates of Project Planning Parameters	Establish Scrum pre-game phase and perform planning poker
P2	Establish and Maintain a Project Plan as the basis for managing the project	Establish the Vision Define and maintain the Product backlog
P3	Establish and Maintain the Commitment to the Project Plan	Perform the face-to-face planning meeting
P4	Select product or product-component solutions from alternative solutions	Develop based on an incremental and iterative life cycle
P5	Develop the product or product-component designs.	
P6	Conduct the preparation for verification.	Establish "done criteria" Perform sprint review meetings
P7	Verify Selected work products against their specified requirements.	Perform "done criteria" Hold sprint review meetings
P8	Conduct the Preparation for validation.	Conduct the stakeholder involvement
P9	Validate the product or product components to ensure that they are suitable for use in their intended operating environment.	Conduct Product Owner and Scrum Master roles
P10	Make the product-component interfaces, both internal and external compatible.	Perform daily meetings, Assemble scrum of scrum in case of larger teams Hold retrospective meeting.
P11	Integrate and assemble product components, and deliver verified and validated product.	Perform incremental product delivery
P12	Conduct the preparation for risk management.	Define the Product Backlog Indentify epics
P13	Identify and analyze risks to determine their relative importance.	Perform daily meetings
P14	Mitigate Risks	Perform daily meetings Identify impediments
P15	Manage requirements and identify inconsistencies with the project plans and work products.	Establish Scrum pre-game phase and perform planning poker Perform the face-to-face planning meeting Hold sprint review meetings Manage user stories in the Sprint Backlog
P16	Monitor actual performance and progress of the project against the project plan.	Perform daily meetings Hold retrospective meeting
P17	Manage corrective actions to closure when the project's performance or results deviate significantly from the plan.	Hold review meetings Perform daily meetings
P18	Evaluate objectively adherence of the performed process and associated work products and services to applicable process descriptions, standards, and procedures.	Hold retrospective meeting
P19	Track and communicate noncompliance issues objectively , and ensure theirs resolution	

Figure 5. Results of the assessments of assets.



4.2. Lessons learned and Limitations

In this section we summarize the lessons learned of evolving our teaching model during three years of a software engineering course. The approach does require some discipline to implement, but the resulting effect can be rewarding. The major challenges faced by the students were: inability to make accurate estimates of workload, resistance to expand a design beyond the immediate requirements, use of an effective and standard testing framework, and ability to discard code in appropriate situations. Each group of students can tackle much more sophisticated and interesting project features comparing to the first course based on the RUP framework.

The main drawback of RUP is that students focus on reaching deadlines falling in a waterfall process. Unlike Scrum, not all team members participate in the planning phase; for this reason, it is hard to obtain commitment from all students at the moment of delivering a product deliverable. Furthermore, regarding testing, students design the required test cases but they cannot be run because the students are not able to finish the implementation of all the user requirements.

The teaching model presented in this work and its findings from the experiments seem to be applicable to other case-studies, under the assumption that the students who attended the course have been trained in the proposed combination of Scrum and Agile Coaching. One interesting finding is that the variations of the median and standard deviation of the coverage percentage throughout the three case-studies denotes the evolution of our teaching model. These metrics indicate a progressive increment in the median of the percentage of the practice coverage from 30.5% to 58.6% as the teaching model evolves. The incorporation of the Agile Coach's role allows us to make adjustments from case to case to both improve the students' learning process and gain their commitment to follow the teaching model. Remarkably, the standard deviation decreased from 19.45% to 5.82% during the period 2008-2010. The teaching model revealed that this decrease was a consequence of compliance with the done criteria, carefully guidance performed by the Agile Coach and improvements in project tracking which resulted in a homogeneous accomplishment of the SE practices.

However, a perceived drawback of the Scrum framework in the academic context is that a Scrum Team requires members with significant experience in software development in order to be effective. Here, both the Scrum Masters and team members were non-experienced students. Most of students have contact with the software development process for the first time so that this is a strong constraint. To tackle this problem, we introduce the Agile Coach's role as a vehicle for coaching and guiding Scrum Teams formed by non-experience members to ensure the delivery of a high-quality product. Thus, an Agile Coach is responsible for gaining commitment and motivation from students, assisting in the identification and implementation of improvements and encouraging the communication with the Product Owner to define and negotiate the working products to be delivered. It is important to note that the Agile Coach holds a Weekly Meeting to discuss the problems found in the development platform, bad practices with tools and lack of documentation, but without interfering with the self-organizing characteristic of agile teams.

Regarding the generalization of our findings, we now discuss the issues that may bias the results of our experiment. First, we simulate an industrial environment in which professors do their best to replicate problems occurring in professional contexts. However, the case-studies were highly influenced by the characteristics of the academic context namely students' motivation, Product Owner' pressure, and the lack of both students' full-time attention and a physical Team Room for all the teams. Regarding this last issue, we had to deal with students who were not able to fully advocate to the course because of other courses, mandatory final exams, and external links with companies. Furthermore, the students were geographically distributed so that the physical Team Room differs among teams. Second, we carried out the experiments with students from the UNICEN University. Participants with other backgrounds, domain knowledge, or levels of expertise might have behaved differently. Finally, the experience of the Agile Coach acquired along the courses is considered another limitation since it may impact in the running of the teaching model. The coaching strongly depends on the ability of the Agile Coach to deal with group management, resource allocation and leadership. Her common sense and perception play a vital role in the model since she is responsible for guiding the students in the right way.

5. Related work

Agile software development has received significant academic attention because of its widespread application in the commercial world [23, 25]. Thus, teaching and learning strategies had to be reoriented towards the software industry demands without neglecting academic quality. Over the past few years, there have been several approaches focused on teaching agile methodologies. Several studies in master's degree software engineering courses were performed to adopt agile methods in the curricula [17, 28]. Coupal and Boechler [5] reported an experience comparing a capstone project developed following an agile approach to their previous projects developed in a traditional way. Devedzic and Milenkovic [6] 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 by introducing practices such as refactoring and pair-programming. In addition, the authors found the Scrum roles, Daily Meetings and Sprint Retrospective appropriated for the process development. Hedin et al. [10] reported the use of Extreme Programming to large group of students and found this methodology highly suitable for introducing them to software engineering. Our work in this paper differs from the Hedin's one since the teaching model is evaluated by assessing the coverage of software engineering practices proposed by CMMI. Koster [14] worked on a SE course in which he introduced AM, particularly pairs programming, to make better software in a more enjoyable scenario. He performed improvements over previous years. However, his work is particularly focused on programming practices.

Mahnic [21] discussed the achievement of teaching goals and provided empirical evaluation of students' progress in estimation and planning skills using Scrum. Also, he observed the behavior of students using Scrum for the first time [20]. However, the inclusion of the Agile Coach is not discussed in these works. Furthermore, our approach considers the teaching of the CMMI practices by accomplishing Scrum practices.

Regarding teamwork, a pedagogical approach was addressed by Chua-Hoo Tan et al. in 2008. They discussed a hybrid agile methodology developed for giving a course of Information Systems (IS). In this course, they focused on team-based guidance rather than on traditional lecture-based teaching. Also, they highlighted the importance of providing working and integrated software, adopting a progressive and flexible method of software development, and adapting to changes in system requirements [32]. Our work differs by providing a concrete comparison between using an agile approach in combination with the coaching of an Agile Coach.

In the light of the above, Alfonso and Botia [1] have subscribed to this idea and added that teachers can act as a project manager with the purpose of planning, monitoring and controlling the learning process effectively. They proposed an iterative and agile process model in a SE undergraduate course. This model served both as an educational technique for teachers and as a subject of learning for students. However, the impact of the manager on the teaching model is not described in terms of the quality of software practices and processes in order to know the benefits of including the manager in the approach.

Finally, the combination between CMMI and AM in software development has been tackled by several authors. They have indicated that AM are useful to reach CMMI maturity levels [2, 26, 31, 33]. For example, Paulk [26] suggests that the use of stories, on site customer and continuous integration of XP fulfill the goals of the CMMI requirement management. Sutherland et al. [31] stated that using CMMI and Scrum with Lean development significantly improved the software process performance positioning the company in a CMMI level 5.

6. Conclusions

This work presented a teaching model based on a balance between Scrum and the Agile Coach's role. We 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 good software practices and maintaining the quality of software processes.

Teaching Scrum software development seems to be effective if students are involved in the development of a project rather than in traditional of-the-book classes. Facing the software engineering problems in a controlled environment gives students the required skills to work in professional contexts. This teaching strategy may help students integrate with the software industry in a better way.

In this paper we have also shown the weakness of a rigid software process based on RUP.

This limitation can be tackled by teaching software engineering practices with Scrum, as it was shown in the second case-study. However, several misconceptions about how to work with the agile framework arose. To tackle this problem, the third case-study consisted in incorporating the role of an Agile Coach to coach the students. Following this line, the assessment of the CMMI practices has also revealed the importance of using a balanced approach between discipline and agility which can help teams institutionalize Scrum more consistently.

As future work, we will focus on applying a teaching tool to allow students to setup the physical development environment through a virtual world in spite of being physically distributed. This tool bases on teaching and integrating teamwork-oriented skills in a real software development environment based on Scrum [29]. We are planning to incorporate assistance to students according to their problems observed during the running of the teaching model. A set of suggestions and corrective actions will be added to the tool in order to provide students with permanent feedback taking into account the way in which the students learn.

To sum up, teaching Scrum complemented with the presence of an Agile Coach is effective for improving communication among students and encouraging their social integration. Beneficially, Scrum leads students to accomplish several CMMI practices with less overhead in terms of documentation and bureaucracy. In addition, using Scrum increases the coverage of the CMMI practices in comparison to the RUP implementation.

References

1. Alfonso, M. I. and Botia, A. An iterative and agile process model for teaching software engineering. In 18th Conference on Software Engineering Education and Training. 2005.
2. Boehm, B. and Turner, R. Balancing Agility and Discipline: A Guide for the Perplexed. Addison Wesley, 2008.
3. Cano, M. D. Students' involvement in continuous assessment methodologies: A case study for a distributed information systems course. IEEE Transactions on Education, 54(3):442 –451, 2011.
4. Cohn, M. Agile Estimating and Planning. Prentice Hall, 2006.
5. Coupal, C. and Boechler, K. Introducing agile into a software development capstone project. In Proceedings Agile Conference, pages 289 – 297, 2005.
6. Devedzic, V. and Milenkovic, S. R. Teaching agile software development: A case study. IEEE Transactions on Education, 54, 2011.
7. Diaz, J., Garbajosa, J. and Calvo-Manzano, J. A. Mapping cmmi level 2 to scrum practices: An experience report. Software Process Improvement, volume 42 of Communications in Computer and Information Science, pages 93–104. Springer Berlin Heidelberg, 2009.
8. Glazer, H. et al. CMMI or Agile: Why not embrace both! Technical Note, CMU/SEI-2008-TN-003, Software Engineering Process Management, Carnegie Mellon, 2008.
9. Glazer, H. et al. Love and Marriage: CMMI and Agile need each other. IEEE Software, 2010.
10. Hedin, G., Bendix, L. and Magnusson, B. Teaching extreme programming to large groups of students. Journal of Systems and Software, 74(2):133–146, 2005.
11. Hurtado Alegría, J. A. and Bastarrica, M. C. Implementing CMMI using a combination of Agile Methods. CLEI Electronic Journal, 1:1, 2006.
12. Keil, P. and Fritzsche, M. Agile methods and CMMI: Compatibility or conflict? e-Informatica Software Engineering Journal, 1, 2007.
13. Kniberg, H. The manager's role in scrum. World Wide Web electronic publication, http://www.scrumalliance.org/articles/103-the-managers-role-in-agile.
14. Koster, B. Agile methods fix software engineering course. J. Comput. Small Coll., 22:131–137, December 2006.
15. Kruchten, P. The Rational Unified Process: An Introduction. Addison-Wesley. 2003.
16. Kulpa, M. and Johnson, K. Interpreting the CMMI. CRC Press, 2008.

17. Laplante, P. A. An agile, graduate, software studio course. IEEE Transactions on Education, 49(4):417 –419, nov. 2006.
18. Maher, P. Weaving agile software development techniques into a traditional computer science curriculum. Third International Conference on Information Technology: New Generations., 0:1687–1688, 2009.
19. Mahnic, V. Teaching Scrum through Team-Project Work: Student's Perceptions and Teacher's Observations. The International Journal of Engineering Education. 2010.
20. Mahnic, V. A capstone course on agile software development using scrum. IEEE Transactions on Education, PP(99):1, 2011.
21. Mahnic, V. A case study on agile estimating and planning using scrum. Electronics and Electrical Engineering, 5, 2011.
22. Marcal, A., Freitas, B., Soares, F., Furtado, M., Maciel, T. and Belchior, A. Blending Scrum practices and CMMI project management process areas. Innovations in Systems and Software Engineering, 4:17–29, 2008.
23. Mathiassen, L. and Pries-Heje, J. Business agility and diffusion of information technology. Eur. J. Inf. Syst., 15:116–119, April 2006.
24. Osorio, J. A. et. al. Moving from Waterfall to Iterative Development – An Empirical Evaluation of Advantages, Disadvantages and Risks of RUP. In Proceedings of 37th EUROMICRO Conference on Software Engineering and Advanced Applications, pages 453-460. IEEE Society, Finland, 2011.
25. Nerur, S. and Balijepally, V. Theoretical reflections on agile development methodologies. Commun. ACM, 50:79–83, March 2007.
26. Paulk, M. C. Extreme programming from a CMM perspective. IEEE Software, 18(26):19–26, 2001.
27. Pikkarainen, M. and Mantyniemi, A. An approach for using cmmi in agile software development assessments: Experiences from three casestudies. In SPICE 2006 conference, Luxemburg., 2006.
28. Rico, D. F. and Sayano, H. H. Use of agile methods in software engineering education. In Proc. Agile 2009 Conf., Chicago, IL, 2009, pp. 174-179., 2009.
29. Rodríguez, G., Soria, Á. and Campo, M. Teaching Scrum to Software Engineering Students with Virtual Reality Support. Lecture Notes in Computer Science (Advances in New Technologies, Interactive Interfaces and Communicability - ADNTIIC 2011). In press. Springer-Verlag. 2011. ISSN 0302-9743.
30. Schwaber, K. and Beedle, M. Agile Software Development with Scrum. Prentice Hall, 2002.
31. Sutherland, J. et al. Scrum and CMMI Level 5: The Magic Potion for Code Warrior. Proceedings of the 41st Annual Hawaii International Conference on System Sciences, 2008.
32. Tan, C., Tan, W. and Teo, H. Training students to be agile information systems developers: a pedagogical approach. In Proceedings of the 2008 ACM SIGMIS CPR, pages 88–96, New York, USA, 2008.
33. Turner, R. and Jain, A. Agile meets CMMI: Culture clash or common cause? In Don Wells and Laurie Williams, editors, Extreme Programming and Agile Methods-XP/Agile Universe 2002, volume 2418 of Lecture Notes in Computer Science, pages 153–165. Springer Berlin / Heidelberg, 2002.