Improving the Experience of Teaching Scrum

Ricardo Pérez-Castillo[1], Ismael Caballero[1] and Moisés Rodríguez[1,2]
[1]Institute of Technology and Information Systems, University of Castilla-La Mancha
Camino de Moledores, 13051 Ciudad Real, Spain
[2]AQCLab Software and Data Quality Laboratory,
Camino de Moledores, 13051 Ciudad Real, Spain
[ricardo.pdelcastillo | ismael.caballero | moises.rodriguez]@uclm.es

***Abstract*—Scrum dramatically shortens the feedback loop between customer and developers as well as between requirement list and functional implementations. Scrum poses to turn small teams into self-managers of their own work. Despite the increasing importance of Scrum in the software development Industry, Academia is not providing a suitable response to meet the demand of professionals. This is because, in many cases, reference curricula do not consider Scrum with a great relevance in knowledge areas nor in time scheduled for it. Even so, Academia should be able to educate and provide software engineers with enough Scrum knowledge and skills. To collaborate to this aim, we present our efforts for improving the teaching/learning experience when dealing with time limitations: we have verified that when using a comparison strategy (regarding traditional software development methodologies like Unified Process, (UP)), students improve their learning process. To validate our results, students were assessed through a twofold assessment: (i) based on Scrum certification exams plus, and with (ii) a questionnaire based on the previously mentioned comparison (Scrum vs UP). The main conclusion raised from this investigation is that students' learning experience is really more satisfactory when Scrum concepts are introduced after a comparison with UP ones.**

Keywords—Software Engineering; Project Management; Scrum.

I. INTRODUCTION

The emergence of the agile concept along with the understanding of the principles contained in Agile Manifesto offered a more revolutionary view of development [1-4]. As the Agile Manifesto has emphasized, agile thinking does not discard earlier ideas and concepts; it adjusts the emphasis given to different aspects of the development process allows for greater adaptability of form. Some process artifacts, such as extensive documentation, are deemphasized, and the people in the process (developers, customers, and other stakeholders) and their interactions are typically given greater priority.

Agile software development depicts a set of principles for software development under which requirements and software solutions evolve in an adaptive way through the collaborative effort of self-organizing cross-functional teams and the own customer [5].

According to a survey concerning agile method [6], 98% of respondents said that their organization has successfully finished projects when using agile techniques. The same survey states that the three most common reasons for adopting agile methods are:

- Accelerate product delivery
- Enhance ability to manage changing priorities
- Increase productivity

Agile software development principles have led to several agile methods in the last decade such as extreme programming (XP), Kanban, Feature-Driven Development, Lean development or Scrum among many other. Scrum or Scrum variants continue being used in more than two-thirds of the projects included in the survey [6].

Scrum is not a prescriptive process; it doesn't describe what to do in every circumstance. Scrum is used for complex work in which it is almost impossible to predict everything that will occur. Accordingly, Scrum simply offers a framework and set of best practices that keep everything visible. This allows Scrum's practitioners to know exactly what is going on and to make ad hoc adjustments to keep the project advancing toward the desired goals [3].

The increasing use of such agile methods for software development, and particularly Scrum, has reoriented education of future computer science and software engineers [7]. Scrum features allow students to gain skills beyond technical and scientific scenarios, such as teamwork-related abilities. These features enable Software Engineering professors to let students have a deeper understanding of agile methods and, at the same time, provide mechanisms for evaluating individual agile concepts [3].

Despite the well-known importance of Scrum and how useful is for students, it is not still well enough recognized by current international reference teaching curricula like SWEBOK [8] or ACM/IEEE SE [9]. In most of mentioned curricula, Scrum and in general agile methods are slightly mentioned but without an own well-recognized area. This fact leads to two major learning problems:

- To be aligned with such reference curricula, most university syllabi do not provide subjects or slots of time specifically focused on Scrum or agile methods.
- Having provided a syllabus considering in somehow Scrum or agile method, such topics are condensed in a short period of time. As a result, most of the times, those topics cannot be successfully taught

Two effects raise as a collide effect to fill this gap:

978-1-5386-2957-4/18/$31.00 ©2018 IEEE 17-20 April, 2018, Santa Cruz de Tenerife, Canary Islands, Spain
2018 IEEE Global Engineering Education Conference (EDUCON)

- universities have begun to offer extended courses under payment outside ordinary syllabi.
- Private companies have begun to offer private Scrum certification courses in detriment of teaching of software engineering courses addressing Scrum at universities.

Considering these effects, this paper deals with the challenge of teaching Scrum effectively in a limited period within the academic term. Main hypothesis of this research is that Scrum method can effectively be taught by comparison with other traditional software engineering methods for which a specific period of time was already planned.

This paper presents an experience concerning teaching agile method and Scrum in our course. This is conducted within 15 weeks - Software Engineering course of 45 hours, from which 4,5 hours (10%) were dedicated for teaching agile methods and Scrum. Having so little time, we posed that students could learn better Scrum concepts when compared to UP concepts previously explained. To check the validity of our hypothesis, we provided a dual assessment for students by means of two different questionnaires: (i) one questionnaire with multiple choice questions coming from Scrum certification exams; and (ii) one questionnaire based on classification of different criteria between Scrum or traditional project management. Marks in a scale of 1 to 10 were on average 6.9 for the first questionnaire, against 9.1 for the second one. Result assessment shows that students will gain a better understanding of Scrum by considering a previous reference method already known like Unified Process (UP).

The main implications of this study are that having a small period for addressing Scrum topics, it can be taught in an effective way by comparison with other traditional methodologies.

The remaining part of the document is organized as follows. Section II provides an overview of the main international curriculum context in which scrum is framed. Section III shows some related teaching experiences. Section IV presents the proposed approach for teaching Scrum to software engineer students. Section V depicts in detail the case study conducted concerning the teaching approach. Finally, Section VI draws conclusions and learned lessons.

II. Scrum in International Software Engineering Curricula

Currently, there are several international curricula focused on the profession of Software Engineering such as SWEBOK, Computing Curricula, ICF-2000 (IFIP / UNESCO) or ISCC, among others. However, SWEBOK [8] and ACM / IEEE SE [9] are probably the most relevant ones. In this section, it is presented how some of these curricula address Scrum and agile methods in general.

A. SWEBOK curriculum

SWEBOK (*Software Engineering Body of Knowledge*), which was designed for the accreditation of university curricula and certification of professionals, identifies a core body of knowledge that characterizes the discipline of Software Engineering divided into 15 knowledge areas. The ninth knowledge area is Software Engineering Models and Methods which impose structure on software engineering with the goal of making that activity systematic, repeatable, and ultimately more success-oriented. This area is divided into four main topics. The last one is Software engineering methods which, in turn, consider agile methods (see Fig. 1).

According to this topic area in SWEBOK [8], Agile methods were born in the 1990s from the need to reduce the apparent large overhead associated with heavyweight, plan-based methods used in large-scale software development projects. Agile methods are considered lightweight methods in that they are characterized by short, iterative development cycles, self-organizing teams, simpler designs, code refactoring, test-driven development, frequent customer involvement, and an emphasis on creating a demonstrable working product with each development cycle. Although there are many agile methods available in the literature, SWEBOK focuses on some of the more popular approaches such as Rapid Application Development (RAD), eXtreme Programming (XP), Feature-Driven Development (FDD) and of course Scrum.

Scrum is summarized by SWEBOK as an agile approach more project management-friendly than the others. The scrum master manages the activities within the project increment. Each increment is called a sprint and lasts no more than 30 days. A Product Backlog item list is developed from which tasks are identified, defined, prioritized, and estimated. A working version of the software is tested and released in each increment. Daily scrum meetings ensure work is managed to plan.

B. ACM/IEEE SE curriculum

The Computing Curricula of ACM / IEEE-CS provides the guidance for curriculum development of careers of computer science and engineering.

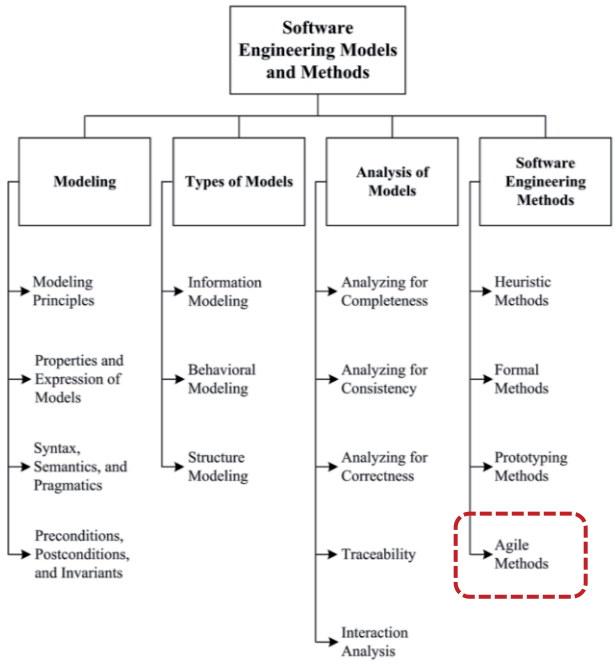

Fig. 1. Knowledge areas of software engineering models and methods in SWEBOK [8].

The Computer Curricula is comprised of several parts: a master volume and additional volumes for specific disciplines. One of these volumes, the SE 2014 [9], is particularly focused on the description of software engineering curriculum. The SE 2014 is divided into 10 knowledge units. One of these knowledge units is directly related to the software processes: the unit PRO entitled Software Evolution (see Fig. 2). This knowledge unit is divided into five subunits and two of them highly related to agile methods: Process Implementation (PRO.imp) and Project planning and tracking (PRO.pp) (see Fig. 3). SE 2014 additionally specifies the certain time that should be dedicated in each unit. In case of these two knowledge subunits, 8+8 hours must be dedicated. Together with desirable time, SE 2014 specifies the attributes using the Bloom's [10] refereeing to knowledge (k), comprehension (c), and application (a); as well as the topic's relevance to the core, which is represented as essential (E) or desirable (D). Agile method concepts are included in PRO.imp2 (c, E) together other life-cycle model characteristics. Additionally, some Scrum concepts like product backlog, priorities dependencies and changes, as well as project tracking metrics (e.g., burndown charts) are considered respectively in PRO.pp1 (a, E) and PRO.pp6 (a, E). It must be noted all these subunits are considered as essential.

C. Analysis

The two mentioned curricula show how the agile methods and Scrum do not receive as much relevance as other knowledge areas and topics studied in Software Engineering. Specifically, both curricula consider such topic within larger knowledge areas such as Software Engineering methods (SWEBOK) or Software Processes (ACM/IEEE SE). This fact reveals the poor ratio of time devoted to agile methods, and even worse, the poor ratio of time specifically devoted to teach Scrum at universities.

Two phenomena are happening in the software engineering academia and industry as a collide effect to fill this gap:

- In academia, most Computer Science School at universities have special courses for teaching software engineering methodologies. However, to really understand and assimilate agile methodologies like Scrum, students should practice it in contexts closer to real Scrum development environments. More It is not possible to run a project using Scrum by only reading a book or listening to someone who is teaching such a methodology. Hence most courses teach those techniques by requiring the students to complete a given project in a team during a given amount of time. As a result, most universities offer special courses for teaching agile software engineering methodologies such as Scrum [11]. However, these are outside of the mainstream, mandatory courses, and therefore, not every student has the possibility to deal with this topic.
- In software development industry, engineers who already know how to apply Scrum has forced the proliferation of a huge demand of Scrum certification programs around this, what has evolved to a large business on certification. The problem of this solution is that engineers can be certified if they pay and pass an exam which does not mean the learning process was effective in all cases.

KA/KU	Title	Hours	KA/KU	Title	Hours
CMP	**Computing essentials**	**152**	**DES**	**Software design**	**48**
CMP.cf	Computer science foundations	120	DES.con	Design concepts	3
CMP.ct	Construction technologies	20	DES.str	Design strategies	6
CMP.tl	Construction tools	12	DES.ar	Architectural design	12
			DES.hci	Human-computer interaction design	10
			DES.dd	Detailed design	14
			DES.ev	Design evaluation	3
FND	**Mathematical and engineering fundamentals**	**80**	**VAV**	**Software verification and validation**	**37**
FND.mf	Mathematical foundations	50	VAV.fnd	V&V terminology and foundations	5
FND.ef	Engineering foundations for software	22	VAV.rev	Reviews and static analysis	9
FND.ec	Engineering economics for software	8	VAV.tst	Testing	18
			VAV.par	Problem analysis and reporting	5
PRF	**Professional practice**	**29**	**PRO**	**Software process**	**33**
PRF.psy	Group dynamics and psychology	8	PRO.con	Process concepts	3
PRF.com	Communications skills (specific to SE)	15	PRO.imp	Process implementation	8
PRF.pr	Professionalism	6	PRO.pp	Project planning and tracking	8
			PRO.cm	Software configuration management	6
			PRO.evo	Evolution processes and activities	8
MAA	**Software modeling and analysis**	**28**	**QUA**	**Software quality**	**10**
MAA.md	Modeling foundations	8	QUA.cc	Software quality concepts and culture	2
MAA.tm	Types of models	12	QUA.pca	Process assurance	4
MAA.af	Analysis fundamentals	8	QUA.pda	Product assurance	4
REQ	**Requirements analysis and specification**	**30**	**SEC**	**Security**	**20**
REQ.rfd	Requirements fundamentals	6	SEC.sfd	Security fundamentals	4
REQ.er	Eliciting requirements	10	SEC.net	Computer and network security	8
REQ.rsd	Requirements specification and documentation	10	SEC.dev	Developing secure software	8
REQ.rv	Requirements validation	4			

Fig. 2. Knowledge units in ACM / IEEE SE curriculum [9].

Reference		k,c,a	E,D	Hours
PRO.imp	Process implementation			8
PRO.imp.1	Levels of process definition (e.g., organization, project, team, and individual)	k	E	
PRO.imp.2	Life-cycle model characteristics (e.g., plan-based, incremental, iterative, and agile)	c	E	
PRO.imp.3	Individual software process (model, definition, measurement, analysis, and improvement)	a	E	
PRO.imp.4	Team process (model, definition, organization, measurement, analysis, and improvement)	a	E	
PRO.imp.5	Software process implementation in the context of systems engineering	k	E	
PRO.imp.6	Process tailoring	k	E	
PRO.imp.7	Effect of external factors (e.g., contract and legal requirements, standards, and acquisition practices) on software process	k	E	
PRO.pp	Project planning and tracking			8
PRO.pp.1	Requirements management (e.g., product backlog, priorities, dependencies, and changes)	a	E	
PRO.pp.2	Effort estimation (e.g., use of historical data and consensus-based estimation techniques)	a	E	
PRO.pp.3	Work breakdown and task scheduling	a	E	
PRO.pp.4	Resource allocation	c	E	
PRO.pp.5	Risk management (e.g., identification, mitigation, remediation, and status tracking)	a	E	
PRO.pp.6	Project tracking metrics and techniques (e.g., earned value, velocity, burndown charts, defect tracking, and management of technical debt)	a	E	
PRO.pp.7	Team self-management (e.g., progress tracking, dynamic workload allocation, and response to emergent issues)	a	E	

Fig. 3. Knowledge units of Process implementation and Project planning and tracking in ACM / IEEE SE curriculum [9].

III. Related Work

In line with the challenges depicted previously there exist in the literature several works aimed at innovating Scrum teaching or/and improving learning process.

A 2005 workshop focused on engineering course projects for undergraduate students deal with Scrum teaching [11]. This workshop detected some of the fundamental characteristics of teaching projects (e.g., team-based, large-scale, long-lived) are difficult to realize within the constraints of a university course in a single semester. This is particularly true when dealing with young students who may lack the perceptions of the real-world. The workshop explored how educators and industry can work together to develop a more rewarding educational experience for all stakeholders involved.

Scharf and Koch [12] present an in-depth praxis report about an Agile Methods undergraduate course. Such course tries to simulate a real world working environment introducing agile methods like Scrum.

To enhance students' exposure to Scrum, Rodriguez et al. [13] developed Virtual Scrum, an educational virtual world that simulates a Scrum-based team room through virtual elements such as blackboards, a Web browser, document viewers, charts, and a calendar.

Alfonso et al. [14] describes the use of an iterative and agile process model in a Software Engineering undergraduate course. According to the authors, the proposed model serves both as an educational technique (for teachers) and as a subject of learning (for students).

Rico and Sayani [15] carried out a teaching experience concerning agile software development methods. This study analyzed how some groups of students completed fully functional e-commerce websites using agile methods. Groups who struck an optimum balance of customer collaboration, use of agile methods, and technical programming ability had better productivity and website quality.

Mahnic [16] conducted an undergraduate capstone course in software engineering, which not only exposed students to agile software development, but also makes it possible to observe the behavior of developers using Scrum for the first time. The course required students to work as Scrum Teams, responsible for the implementation of a set of user stories defined by a project domain expert playing the role of the Product Owner. During that course, data on project management activities were collected to analyze the amount of work completed, compliance with the release and iteration plans, productivity, ability in effort estimation. This work also discussed the achievement of teaching goals and provides empirical evaluation of students' progress.

IV. Teaching Experience

This section explains deeply our teaching experience conducted during last year for assessing the teaching-learning process concerning Scrum. Firstly, the context in which the experience was conducted is introduced. Secondly, some key organizational aspects are explained. Thirdly, the described Scrum teaching experience is depicted by providing some teaching resources. Finally, the assessment method is presented by providing these resources.

A. Context

The teaching experience was conducted during the first semester of the academic term 2017/18 within a subject called 'Software Engineering II'.

The current syllabus, introduced in 2012, for the Computer Science BSc at University of Castilla-La Mancha (UCLM) consists of four years. This syllabus considers two half-year subjects related to software engineering: 'Software Engineering I' and 'Software Engineering II'. First subject is planned in second term, and second subject in the third one. This experience focuses on the second subject which is aimed at presenting advanced software engineering processes like agile method, testing, software maintenance, and so on. 'Software Engineering II' subject is taught in a half year term, in the first semester, and implies 6 ECTS (European Credit Transfer and Accumulation System) [17].

B. Organizational aspects

The subject is scheduled in 15 weeks with three sessions per week; every session lasts one hour and a half (two lecture + one lab session) (see Table I). This makes a total of 65 hours splitted into 45 hours of lecture and 20 lab hours with practical exercises

The Agile method and Scrum was condensed in only 4 hours and a half, 2 lecture sessions plus one lab session. This schedule was done in line with the mentioned international curricula, and this demonstrates the minimum time slot or period devoted to this trending topic. Hence, the necessity for providing effective and efficient teaching-learning experience.

108 students were enrolled in the during the term 2017/18. Students were divided into three groups, one taught in English with 25 students (group A) and two taught in Spanish with 46 and 37 students (groups B and C respectively). Different professors lecture to each group.

TABLE I. 'SOFTWARE ENGINEERING 2' SUBJECT SCHEDULE

Lesson	Title	Distribution (Lecture + Lab sessions)	W1	W2	W3	W4	W5	W6	W7	W8	W9	W10	W11	W12	W13	W14	W15
1	Software Engineering Processes	10 + 4	◊	◊	◊	◊	◊										
1.1	*Software process introduction*	4 + 1	♦	♦													
1.2	*Unified Process (UP)*	4 + 2			♦	♦											
1.3	***Agile Methods - Scrum***	2 + 1					♦										
2	Quality Management	6 + 3						◊	◊	◊							
3	Configuration Management	2 + 1									◊						
4	Testing Management	6 + 3										◊	◊	◊			
5	Maintenance Management	5 + 3													◊	◊	◊

Concerning IT resources, Moodle platform was customized by our university and used for the whole subject by providing all material there and enabling some questionnaires and tasks for assessing students in such platform.

C. Lecture Sessions and Lab Material description

The two lecture sessions (3 hours in total) were covered with master class about agile method and Scrum supported by some slide material. The teaching plan was first to provide an overview of agile method and to present the main concepts (see Session 1 structure).

Session 1

- Agile methods introduction
- Agile Manifesto principles
- Some Agile Methodologies
- User Stories and task decomposition
- Agile Planning and Estimation

Since traditional software processes like UP [18] were taught in previous lessons, authors used a teaching strategy based on comparison. Thereby, agile method concepts were presented in comparison with traditional methods. The following example used in lecture shows illustrates this idea:

> *"Using Scrum, team deliver working software frequently, from a couple of weeks to a month, with a preference to the shorter timescale. While, UP measures the success of the project in terms of the degree in which the progress of the project adheres to the initial plan."*

After Session 1 concerning the overview of agile methods, Session 2 was fully oriented to Scrum. In this case, the comparison strategy was not possible since it had to be simply limited to presents concepts depicted in Scrum methodology.

Session 2

- Scrum introduction
- Product Backlog
- Sprints
- Scrum Roles
  - Product Owner
  - Scrum Master
  - Team
- Scrum Meetings
  - Sprint Planning Meetings
  - Daily Scrum Meetings
  - Sprint Review Meetings
  - Sprint Retrospective Meetings
- Project monitoring with burndown charts

The lab session focuses on estimating story points and task decomposition. For doing that, students first had to define some user stories as example and then decomposed such stories into some fine-grained tasks which were planned for a four-week sprint as example. Then, students were asked (in groups) to estimate effort for every task by using planning poker technique, which is consensus-based technique.

D. Assessment description

The main hypothesis of this teaching experience is that Scrum method can be more effectively taught by establishing comparison to other traditional software engineering methods like UP, previously explained in the subject. To validate this hypothesis, the assessment process was conducted. The assessment consisted of two questionnaires in Moodle platform, Q1 and Q2.

First questionnaire (Q1) has 10 questions extracted from some examples of Scrum certification exams (see Fig. 4). Every question has 5 choices with a single right answer, hence wrong answers penalize with 20%.

Second questionnaire (Q2) consisted of a different kind of questions. Thirteen questions asked about thirteen different criteria regarding software development methodologies. For each criterion, students were requested to classify each of the two answer options as appropriate for Scrum project management or traditional project management (see Table II).

TABLE II. Q2. QUESTIONNAIRE BASED ON SCRUM VS. TRADITIONAL PROJECT MANAGEMENT COMPARISON

Criteria	Scrum	Traditional Project Management
Emphasis is on	People	Processes
Documentation	Minimal—only as required	Comprehensive
Process style	Iterative	Linear
Upfront planning	Low	High
Prioritization of Requirements	Based on business value and regularly updated	Fixed in the Project Plan
Quality assurance	Customer centric	Process centric
Organization	Self-organized	Managed
Management style	Decentralized	Centralized
Change	Updates to Productized Product Backlog	Formal Change Management System
Leadership	Collaborative, Servant Leadership	Command and control
Performance measurement	Business value	Plan conformity
Return on Investment (ROI)	Early/throughout project life	End of project life
Customer involvement	High throughout the project	Varies depending on the project lifecycle

Question 1. What kind of software development projects can be executed by Scrum Project Management Framework?

□ Choice-1: Complete software packages
□ Choice-2: Customer projects
□ Choice-3: Sub-systems, components or parts of bigger systems
■ Choice-4: All kinds of software development projects
□ Choice-5: None of the given answers

Question 2. What does NOT belong to cornerstones of the agile manifesto?

□ Choice-1: Individuals and interactions over processes and tools
□ Choice-2: Working software over comprehensive documentation
■ Choice-3: Processes over people
□ Choice-4: Customer collaboration over contract negotiation
□ Choice-5: Responding to change over following a plan

Question 3. What is defined by the Scrum Framework?
A) Rules & Roles
B) Document guidelines
C) Artifacts and events

□ Choice-1: A
□ Choice-2: B
□ Choice-3: C
□ Choice-4: A, B, C
■ Choice-5: A, C

Question 4. Where are the customer requirements stored?

■ Choice-1: In the Product Backlog
□ Choice-2: In the Sprint Backlog
□ Choice-3: In a database
□ Choice-4: In a Scrum Product Requirement Specification
□ Choice-5: Nowhere. The Scrum Product Owner knows them

Question 5. Which ones of the following main roles are defined by Scrum Framework?
A) Scrum Tester
B) The Scrum Team
C) Scrum Manager
D) Scrum Master
E) Scrum Product Owner

□ Choice-1: A, B, C, D, E
□ Choice-2: B, C, D, E
■ Choice-3: B, D, E
□ Choice-4: A, B, D, E
□ Choice-5: A, B, C, D

Question 6. Which ones of the following main events are defined by Scrum Framework?
A) Sprint Planning Meeting
B) Sprint Retrospective Meeting
C) Sprint Review Meeting
D) Mid-Sprint Status Review Meeting
E) Daily Scrum Meeting

□ Choice-1: A, B, C, D, E
□ Choice-2: A, B, C, D
□ Choice-3: A, C, D, E
■ Choice-4: A, B, C, E
□ Choice-5: A, C, E

Question 7. Which concept is NOT defined in the Scrum Framework?

□ Choice-1: Scrum Master
■ Choice-2: Project Manager
□ Choice-3: Scrum Product Owner
□ Choice-4: Daily Scrum
□ Choice-5: Scrum Product Burndown

Question 8. What is important in all Scrum projects?
A) Self-organization
B) Clear hierarchies in the company
C) Communication
D) Continuous improvement

□ Choice-1: A, B, C, D
■ Choice-2: A, C, D
□ Choice-3: A, D
□ Choice-4: A
□ Choice-5: A, B

Question 9. In software engineering what are the disadvantages of the classical waterfall model?
A) End-Product has to be fully anticipated beforehand.
B) Some requirements are implemented as defined in the beginning of the project, and yet they are not really needed by the customer.
C) Each phase is strictly separated.

□ Choice-1: A
□ Choice-2: B
□ Choice-3: C
□ Choice-4: A, B
■ Choice-5: A, B, C

Question 10. What are the advantages of the Scrum Framework?

□ Choice-1: Fine-grained requirements are only defined when they are really needed.
□ Choice-2: All activities to design, build and test a certain functionality are kept together in one phase.
□ Choice-3: Changes are expected and welcomed by Scrum team.
■ Choice-4: All of the given answers
□ Choice-5: None of the given answers

Fig. 4. Q1. Scrum questionnaire based on questions from certification exams

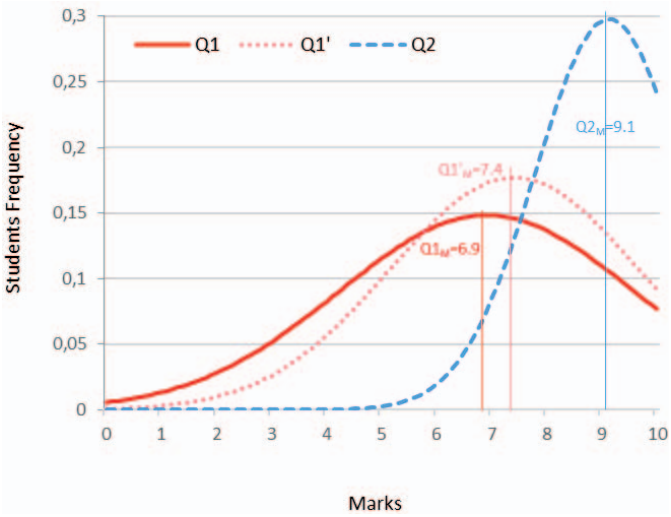

Fig. 5. Distributions of marks for Q1, Q1' and Q2 questionnaires.

V. Result Discussion

After conducting the assessment with the mentioned questionnaires, the obtained results were analyzed. Both questionnaires were scored in a range of 0 to 10. The mean score was $Q1_M$=6.9 and $Q2_M$=9.1 with respective standard deviations of $Q1_{SD}$=2.69 and $Q2_{SD}$=1.34 (see normal distribution representation in Fig. 5). Additionally, in order to limit the effect of the penalization of wrong answer in Q1, results were also analyzed without considering penalization (Q1'). Even so, the mean was $Q1'_M$=7.4 with a standard deviation of $Q1'_{SD}$=2.25 (see Fig. 5).

This result demonstrates that students' scores were better in Q2 than Q1, in other words, the questionnaire about comparison between Scrum and traditional project management seems to be more addressable for students. Additionally, regarding standard deviation it can be observed that marks in Q1 vary more than in Q2. The conclusion is that the learning process was more homogenous using the comparison strategy (Q2).

The standard deviation is not moderated (both around 2) which indicates that students' marks varies around the mentioned means in both questionnaires.

VI. Conclusions

Teaching in Software Engineering should be addressed to enable students to develop their future work in current professional environment, which are demanding tasks and skills that are sometimes not covered within current academia curricula. A large number of engineers are today using Scrum for developing software. Unfortunately, reengineering often receive only bit attention in terms of teaching.

This paper presented a teaching experience concerning Scrum and in general, Agile methods. The main challenge of this experience was that the teaching of this topic must be addressed in a short period of time. Despite this drawback, the expectations were achieving the maximum effectiveness in the learning process due to the importance of Scrum for the future software engineers.

To fulfill both, time constraints and maximization of Scrum concepts and skills acquisition, the teaching proposal consisted of a lecture plus an assessment orientated to the comparison of Scrum with traditional methodologies. The assessment results show that students obtained better scores in the questionnaire based on the mentioned comparison.

Among the lessons learned, which could be applied to future repetitions of the teaching experience, we included:

- Some of the international, reference curricula do not explicitly consider Scrum. Even when such curricula consider Scrum, for example included in other wide topic areas like agile methods, it is neither planned with as much time as expected considering its importance in the industry.
- Despite previous lesson learned, the core concepts of Scrum can be acquired in a short teaching-learning time as demonstrated both questionnaires. However, Scrum skills must be learnt in contexts closer to the real software development environments.
- The comparison with traditional methodologies like UP can help to the comprehension of Scrum and other agile methods. Academia, based on reference curricula, traditionally spend more time in such traditional methods (because of historical reasons). Thereby, a comparison strategy with such topics, which are well known by students, help to detect differences and common points. Therefore, the learning process may be faster.

As a future work, we will conduct an extended result analysis through the replication of this experience in the next years. Also, this strategy could be combined with the strategy mentioned in some related work about providing a pseudo-real environment in lab sessions to carry out a development project following Scrum.

Acknowledgment

This work has been developed within the SEQUOIA Project, funded by Fondo Europeo de Desarrollo Regional FEDER and (Ministerio de Economía y Competitividad, (TIN2015-63502-C3-1-R) (MINECO/FEDER). The work was also supported through a grant Dr. Ricardo Pérez-Castillo enjoys from JCCM within the initiatives for talent retention and return in line with RIS3 goals. The research is additionally part of the project ECD: *Evaluación y Certificación de la Calidad de Datos* (PTQ-16-08504), cofunded by the program *Torres Quevedo* by the Spanish *Ministerio de Economía, Industria y Competitividad*.

References

[1] Boehm, B. and R. Turner, *Balancing Agility and Discipline: A Guide for the Perplexed, Portable Documents*. 2003: Addison-Wesley Professional.

[2] Beck, K., *eXtreme Programming Explained: Embrace Change*. 2004.

[3] Schwaber, K., *Agile project management with Scrum*. 2004: Microsoft press.

[4] Holcombe, M., *Running an agile software development project*. 2008: John Wiley & Sons.

[5] Collier, K., *Agile analytics: A value-driven approach to business intelligence and data warehousing*. 2012: Addison-Wesley.

[6] VERSIONONE.COM, *11th Annual State of Agile Report*. 2017, versionone.com. p. 17.

[7] Rodríguez, G., A. Soria, and M. Campo. *Teaching scrum to software engineering students with virtual reality support*. in *International Conference on Advances in New Technologies, Interactive Interfaces, and Communicability*. 2011. Springer.

[8] Fairley, P.B.a.R.E., ed. *Guide to the Software Engineering Body of Knowledge, Version 3.0*. 2014, IEEE Computer Society.

[9] Computing Curriculum Project, *Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering (SE2014)*. 2014, Joint Task Force on Computing Curricula IEEE Computer Society and Association for Computing Machinery (ACM).

[10] Bloom, B.S., *Taxonomy of educational objectives; the classification of educational goals*. 1956, New York: Longmans, Green.

[11] Tilley, S., et al. *Report from the 2nd International Workshop on Software Engineering Course Projects (SWECP 2005)*. in *Software Engineering Education and Training, 2006. Proceedings. 19th Conference on*. 2006. IEEE.

[12] Scharf, A. and A. Koch. *Scrum in a software engineering course: An in-depth praxis report*. in *Software Engineering Education and Training (CSEE&T), 2013 IEEE 26th Conference on*. 2013. IEEE.

[13] Rodriguez, G., Á. Soria, and M. Campo, *Virtual Scrum: A teaching aid to introduce undergraduate software engineering students to scrum.* Computer Applications in Engineering Education, 2015. 23(1): p. 147-156.

[14] Alfonso, M.I. and A. Botia. *An iterative and agile process model for teaching software engineering*. in *Software Engineering Education & Training, 18th Conference on*. 2005. IEEE.

[15] Rico, D.F. and H.H. Sayani. *Use of agile methods in software engineering education*. in *Agile Conference, 2009. AGILE'09.* 2009. IEEE.

[16] Mahnic, V., *A capstone course on agile software development using Scrum.* IEEE Transactions on Education, 2012. 55(1): p. 99-106.

[17] Comission, E. *European Credit Transfer and Accumulation System (ECTS)*. Supporting education and training in Europe and beyond 2017 17/11/2017]; Available from: http://ec.europa.eu/education/resources/european-credit-transfer-accumulation-system_en.

[18] Jacobson, I., et al., The unified software development process. Vol. 1. 1999: Addison-wesley Reading.