The Journal of Systems and Software 111 (2016) 242–253

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

Towards better Scrum learning using learning styles

Ezequiel Scott, Guillermo Rodríguez, Álvaro Soria*, Marcelo Campo

ISISTAN Research Institute (CONICET-UNICEN), Campus Universitario, Paraje Arroyo Seco, Tandil, Buenos Aires, Argentina

ARTICLE INFO

Article history:
Received 19 March 2015
Revised 6 October 2015
Accepted 8 October 2015
Available online 23 October 2015

Keywords:
Agile software development
Software engineering education
Learning styles

ABSTRACT

Considerable attention has been paid to teaching Scrum in software engineering education as an academic response to the software industry's demands. In order to reinforce and strengthen the understanding of Scrum concepts, professors should personalize the learning process, catering for students' individual learning characteristics. To address this issue, learning styles become effective to understand students' different ways of learning. In this context, the meshing hypothesis claims that when both teaching and learning styles are aligned, the students' learning experience is enhanced. However, the literature fails to evidence support for the meshing hypothesis in the context of software engineering education. We aim to corroborate the meshing hypothesis by using teaching strategies matching the Felder–Silverman Learning Style Model in a Scrum course. Based on previous findings, we focused on the processing dimension of the model. To validate our approach, two experiments were conducted in an undergraduate software engineering course in the academic years 2013 and 2014. We provided students with a Scrum class by applying teaching strategies suiting students' learning style. Test results corroborate that students' outcomes improved when receiving the strategy that match their learning styles. Our data highlight opportunities for improving software engineering education by considering the students' learning preferences.

1. Introduction

During recent years, agile methodologies have become increasingly popular in both the software industry and academic research (Dingsoyr et al., 2012). The main reasons that support industrial adoption of agile practices are improved performance and fast response time; additionally, agile methodologies allow for delivering better outcomes than plan-driven ones. Accommodating industrial requirements, universities have started to teach agile practices in software engineering courses (Martin Kropp, 2013). Out of the variety of agile methodologies, Scrum is in the cutting-edge of software engineering education (Ambler, 2006; Hossain et al., 2009; Paasivaara et al., 2009; Salo and Abrahamsson, 2008; VersionOne, 2012) because its widespread adoption in the industry (VersionOne, 2012). As university level teaching aims at preparing students for both facing common software engineering challenges and facilitating their insertion in professional contexts, the software engineering curricula should respond by providing students with knowledge and experiences on how to use Scrum in practice.

After exploring several research works that aim to teach Scrum in software engineering undergraduate courses (Smith et al., 2011; Broman et al., 2012; Damian et al., 2012; Scharf and Koch, 2013; Paasivaara et al., 2014; Lu and DeClue, 2011), we identified the need to provide students with relevant personalized learning scenarios that might offer more meaningful learning experiences. The personalization of the learning process is aligned with learner-centered principles, in which professors should consider students' learning preferences, their strengths and weaknesses (McCombs and Whisler, 1997). In this context, learning styles are not only useful tools to detect how learners perceive, interact with, and respond to the learning environment, but also valuable indicators for the cognitive, affective, and psychological students' behaviors (Keefe, 1988). Moreover, the meshing hypothesis states that learners have different ways of learning, and by considering these preferences, students' learning experiences may be enhanced along with their educational outcomes (Pashler et al., 2008).

Pashler et al. (2008) have also summarized both concepts and evidence related to the meshing hypothesis. Their findings show that most of the research works reviewed disregarded both adequate evidence and rigorous experimental designs to justify the impact of learning styles on students' learning experience. Although a significant relation between students' learning style and the way these students learn Scrum was reported (Scott et al., 2014a), the research failed to provide evidence of the fact that students' learning style

* Corresponding author. Tel.: +54 249 443 9681 Ext. 32
E-mail addresses: ezequiel.scott@isistan.unicen.edu.ar (E. Scott), guillermo.rodriguez@isistan.unicen.edu.ar (G. Rodríguez), alvaro.soria@isistan.unicen.edu.ar (Á. Soria), marcelo.campo@isistan.unicen.edu.ar (M. Campo).

http://dx.doi.org/10.1016/j.jss.2015.10.022
0164-1212/

might impact on students' learning experience. As a consequence, we claim that more research is required on effective approaches in software engineering education to validate the use of teaching strategies (i.e. instructional methods) that fit students' learning styles.

Based on the suggestions made by Pashler et al. (2008), we present an appropriate methodology that evidences encouraging results when teaching Scrum by considering students' learning styles. To validate the meshing hypothesis, an experiment was conducted in the academic year 2013, and replicated in 2014 in the context of the Software Engineering course at Universidad Nacional del Centro de la Provincia de Buenos Aires (UNCPBA). The experiment consisted of three steps. Firstly, students were required to complete the Index of Learning Style questionnaire (Felder and Soloman, 1991). By considering previous findings (Scott et al., 2014a), we focused on the processing dimension that was applied to divide students into two groups, each one containing active and reflective students. One group was given a class on basic Scrum concepts suitable for reflective students, and the second group was given the same concepts in a suitable manner for active students. Secondly, the students conducted a one-Sprint capstone project to reinforce the concepts learned in the previous step. Thirdly, at the end of the Sprint both groups of students were evaluated by means of a test to assess students' learning of Scrum. The results showed that students who received the class suitable for their learning styles achieved better educational outcomes. This contribution is a step forward to improve the teaching of Scrum by considering the learning preferences of students.

The remaining of this paper is organized as follows. Section 2 introduces the background topics in the area of Scrum teaching and learning styles, as well as related works in the context of learning styles in teaching. Section 3 presents our research approach to corroborate the meshing hypothesis. In Section 4, the results obtained from the experiments are presented. Section 5 analyzes the most valuable findings, lessons learned and threats to validity. Finally, in Section 6, we state our conclusions and identify future lines of work.

2. Background

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 may take place. Nowadays, agile software development has received significant academic attention because of its widespread application in the commercial world as an effective vehicle towards an optimal balance between fast software development and software quality. For this reason, software engineering courses have to be continuously reoriented to cater for the demands of the software industry without neglecting academic quality. Students should both really understand and assimilate agile methodologies, and enrich their knowledge by means of hands-on experiences. Thus, showing students advisable software engineering practices is crucial, so that they are capable of ongoing success in the software engineering field (von Wangenheim et al., 2013). However, students learn in many ways and have singular learning preferences, so professors need to know their strengths and weaknesses in order to enhance their learning experience. Thus, the challenge is how to personalize the learning process catering for individual learning characteristics of students by using learning styles.

2.1. Teaching Scrum

Out of the various agile approaches, Scrum has gained wide acceptance in the software industry because of various reasons. First, Scrum 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 prioritizing and satisfying requirements. Third, the retrospective meetings and continual customer contact allow for early detection of impediments and rearrangements of working plans (Schwaber and Beedle, 2002).

In an educational context, all the aforementioned 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. According to the recommendations from the Software Engineering 2004 Curriculum (LeBlanc et al., 2006), software engineering is an activity that requires students to acquire teamwork-related skills. In this line, Scrum promotes a work model to produce high-quality software and permits quick adaptation to changing requirements, allowing students to acquire skills beyond technical and scientific scenarios, such as teamwork-related abilities (Diaz et al., 2009). 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 for undergraduate students (Nerur and Balijepally, 2007). Learning agile software development methodologies will unquestionably increase the marketability of students as professional software engineers. However, how agile methods should be taught at the undergraduate level is still a challenging issue (Lu and DeClue, 2011).

The use of capstone projects has gained considerable attention in software engineering education as a strategy to teach Scrum and all possible issues that commonly arise in real software engineering projects, such as delayed deadlines, cancellation of projects, errors in the application, application not suiting the customer, changing business requirements, and personnel turnover (Mahnic, 2010; Devedzic and Milenkovic, 2011; Schroeder et al., 2012; Mahnic, 2012). In Coupal and Boechler (2005), an experience comparing a project developed by students following an agile approach to their previous projects developed in a traditional way has been reported. In Devedzic and Milenkovic (2011), the authors described their eight years of experiences in teaching agile software methodologies to various groups of students at different universities by weaving together Scrum and XP. Based on the experience acquired, they recommended how to overcome potential problems in teaching agile software development. In Mahnic (2011) the achievement of teaching goals and the empirical evaluation of students' progress in estimation and planning skills using Scrum have been discussed. Also, the behavior of students when using Scrum for the first time has been observed (Mahnic, 2012). Rico reported a capstone course based on agile methods (Rico and Sayani, 2009), in which the students were able to choose an agile method out of Scrum, XP, Feature-driven development (FDD), Dynamic Systems Development Method (DSDM) and Crystal Clear, among others. A pedagogical approach was reported in Tan et al. (2008), in which the authors proposed a hybrid course of Information Systems based on XP, Scrum and FFD. By simulating a software engineering environment as close to real world as possible, Scharf et al. presented a Scrum undergraduate course complemented with useful tools to implement this agile method (Scharf and Koch, 2013). Following the same line, Paasivaara et al. introduced a simulation game to teach Scrum roles, events and related concepts achieving promising results through students' feedback (Paasivaara et al., 2014). Particularly, a previous research work has revealed promising evidence of the use of a capstone project to teach Scrum (Soria et al., 2012). For this reason, we considered the teaching model proposed in the research to support our experiments and introduced personalized teaching strategies to cater for individual learning preferences.

However, the teaching model proposed, in line with most of the works mentioned above, disregards individual characteristics of students that could affect their learning experience. Since students learn in many ways, professors should know the students' profile, their strengths and weaknesses in order to help them to learn agile

Table 1
Dimensions of Felder's learning styles.

Dimension	Values
Perception	Sensing-Intuitive
Understanding	Sequential-Global
Processing	Active-Reflective
Input	Visual-Verbal

practices. The challenge of personalizing the teaching of Scrum is for the professors to teach Scrum concepts catering for individual learning characteristics of students. To address this issue, the professors should obtain insights of students' preferences to design teaching strategies suitable for these preferences (Felder and Silverman, 1988).

2.2. Overview of learning styles

A widely accepted tool to obtain insights of students' preferences when learning is the learning style, which is increasingly being incorporated into technology-enhanced learning. A learning-style model classifies students according to a number of scales that represent the ways in which they receive and process information. Out of the learning-style models, the Felder–Silverman Learning Style Model (FSLSM) is designed for traditional learning and is the most used model in technology-enhanced learning. Moreover, the selection of FSLSM is supported by numerous reasons. First, previous evidence on the relationship between Scrum and learning styles in the context of an undergraduate software engineering course was found (Scott et al., 2014a). Second, we found that the processing dimension bore the most statistically significant relation to Scrum practices (Scott et al., 2013). Third, FSLSM has been widely used in the context of Computer Science education (Graf and Liu, 2009). Fourth, the learning styles can be obtained by means of the Index of Learning Styles (ILS) instrument, a questionnaire based on 44 items to which each student responds according to their learning preferences (Felder and Spurlin, 2005). The results obtained from the ILS distinguish the learners' preferences according to the four dimensions of the model, and they allow for describing trends about stronger and weaker preferences by means of a numeric scale. Finally, several approaches have tackled the automated detection of learning styles, which could replace the questionnaire (Feldman et al., 2014).

The FSLSM comprises 16 learning styles. Table 1 shows the dimensions of the learning styles in this model, namely perception, input, processing, and understanding, together with the values that each dimension may take. The perception dimension indicates the type of information the student prefers to receive: sensory (external) such as sights, sounds, physical sensations; or intuitive (internal) such as possibilities, insights, and hunches. The understanding dimension shows how the student progresses towards understanding: sequentially in continual steps, or in fits and starts. The processing dimension indicates how the student prefers to process information: actively through engagement in physical activity or discussion, or reflectively through introspection. Finally, the input dimension models the sensory channel through which external information is most effectively perceived by the student: visual pictures, diagrams, graphs, demonstrations, or auditory words, and sounds.

In this context, *sensing* students prefer to learn using concrete material, while *intuitive* students prefer more abstract material. *Sequential* students learn better in linear and well-defined steps, while *global* students prefer long steps with more freedom. *Active* students prefer doing tasks or talking about concepts, while *reflective* students are likely to manipulate and examine the information introspectively. Lastly, *visual* students prefer to learn through images or other visual representations, instead of narratives or sounds that explain concepts as *verbal* students do (Felder and Silverman, 1988). This learning-style model allows for a better understanding of students' differences when learning. Bearing students' preferences in mind for Software Engineering courses would eventually enhance the students' learning experience.

Numerous research works have been conducted to incorporate learning styles and provide courses suitable for the individual learning styles of students (Graf and Liu, 2009; Essalmi et al., 2010; Carver et al., 1999; Kuljis and Liu, 2005). However, according to Pashler et al. (2008), the literature has failed to evidence suitable and appropriate support for applying learning-style assessments in course designs. At most one arguable research work has contributed to the idea that providing students with a suitable instructional method improves the students' performance (Sternberg et al., 1999). Other revised works reported appropriate and well-designed research methods, but the results rejected the hypothesis. For instance, Massa and Mayer (2006) conducted an experiment in the context of an electronics course, in which they were able to reject the hypothesis that suitable instructional methods should improve students' performance by taking into account the input dimension of their learning style. In line with visual/verbal dimension, but in a non-educational context, Constantinidou and Baker (2002) carried out an experiment with adult users and found no evidence to support relation between the students' input dimension and the learning of items by means of visual or auditive strategies. Likewise, in a medical-education context, Cook et al. (2009) have focused on corroborating the hypothesis that providing students with a suitable method according to their perception dimension of their learning style would impact positively on students' performance; however, the authors have failed to found evidence.

Based on previous evidence of the relation between the Scrum practices and the dimensions of the Felder–Silverman learning styles of the students (Scott et al., 2013; 2014a), we aim to demonstrate the use of learning styles to achieve personalized software engineering courses. To address this issue, the present work relies on a specific version of learning-style hypothesis widely addressed within the educational literature. We refer to this hypothesis as the meshing hypothesis, which states that if students are provided with an instruction suitable for their learning preferences (e.g., for an "active learner," emphasizing hands-on tasks), the students could enhance their learning experience (Pashler et al., 2008).

3. Research approach

The goal of our work is to corroborate the meshing hypothesis by conducting an appropriate experiment with software engineering undergraduate students; we rely on the professor's background on learning styles, her/his teaching strategies and their implementation in the classroom. To achieve our goal, we propose the study of the application of suitable instructional methods to software engineering undergraduate students and the impact of these methods on the students' learning of Scrum. In particular, we focus on measuring the knowledge obtained by the students after receiving a class with suitable strategies depending on the processing dimension of the students' learning style. Thus, we aim to address the following research questions:

- How should Scrum concepts be taught to suit the processing dimension of each student's learning style? To address this question, the professor should design a set of classes based on the learning characteristics of active and reflective students (Felder and Silverman, 1988). For instance, when teaching the widespread estimating technique called Planning Poker, the professor should design learning activities that contain, on the one hand, theoretical and abstract concepts of Planning Poker for reflective students, and on the other hand, hands-on tasks and simulations for active students. We attempt to answer this question in Section 3.1.

- How do students apply the concepts learned in a software engineering context? To deal with this issue, professors should define a scenario along the course so that students can handle software engineering practices by simulating a professional context. For example, students should play different roles, face typical challenges that occur in software development, estimate software complexity and deal with customer issues, among others. Our proposed approach attempts to shed some light on this question in Section 3.2.
- How should the students' learning be assessed? And, how should the students' test results be compared? The professors should design an evaluation instrument to assess students' learning and find differences between test results obtained by active and reflective students. This instrument should be the same for both types of students and designed so that none of the learning styles is favored. This issue will be addressed in Section 3.3.

To address the above-mentioned research questions, we have implemented a methodology as suggested by Pashler et al. (2008) in the context of an undergraduate Software Engineering course following a teaching model proposed in Soria et al. (2012); Scott et al. (2014b), in which a Scrum-based capstone project is conducted. By following this teaching model, we particularly modified the phase named Initial Phase that aims to both teach basic Scrum concepts and familiarize students with the Scrum framework. In the first step of the methodology, we provide students with Scrum concepts by following teaching strategies that were designed aligned with the processing dimension of the FSLSM. Therefore we organized the teaching practices into two instructional methods: passive and active instructional method (Felder and Silverman, 1988); as discussed in Section 3.1.2, the passive instructional method aims to teach practices by maximizing reflective students' learning experience, whereas the active instructional method aims to teach practices by maximizing active students' learning experience.

In the second step of the methodology, a capstone project is followed according to the teaching model. At this point, the students are organized into teams and engaged to put the learned concepts into practice during one Sprint. Here, we measure the impact of the personalized teaching on students' outcomes.

The third step of the proposed methodology consists in assessing the students' learning of Scrum concepts by means of a test at the end of the Sprint. We gather the scores and analyze relations between these scores and students' learning style. In the following subsections, the steps of the methodology are described in detail.

3.1. Step 1: conducting the teaching of Scrum concepts

The first step of our proposed methodology is to set up the course. To address this issue, we firstly need to detect students' learning style to allocate them to groups; and secondly, we provide each group with a class about basic Scrum concepts suitable for either active or reflective learning style. That is to say, a group will be taught by means of an active instructional method, and the other group will be taught by means of a passive instructional method.

3.1.1. Group formation

Since we aim to corroborate the meshing hypothesis, we need to form students' groups, and thus, we use the ILS values taken from the Index of Learning Styles (ILS) (Felder and Soloman, 1991) adapted to the Spanish language [1]. The students complete the questionnaire and, then, we process the answers. As a result, we obtain the values of the four dimensions of the students' learning style (See Section 2.2). In our work, we only consider the processing dimension because of two main reasons. Firstly, we decided to use only one dimension so as to perform a better and deeper analysis of the impact of suitable teaching strategies on students' performance. Secondly, and more important yet, promising evidence of the relation between Scrum and the processing dimension of the FSLSM was found (Scott et al., 2013).

Once the students' learning style is detected, we divide the students into two groups, each of which containing a balance between active and reflective students (Pashler et al., 2008). Then, we provide each group with a suitable instructional method to teach essential software engineering practices commonly used in Scrum.

3.1.2. Using suitable instructional methods

To teach Scrum practices, we design the classes in the context of user story definition, user story splitting, user story estimation and planning. We choose these practices because of their scaffolding nature as a first step to corroborate the meshing hypothesis. In fact, the Scrum practices included in this study are based on a previous work (Scott et al., 2014b). According to it, the first phase of Scrum includes practices ranging from User Story definition to Agile Planning. These practices are important since they are the basis for subsequent practices such as Sprint review, Daily Scrum, andSprint retrospective meetings. For example, students will only be able to perform the Daily Scrum as long as they have planned their User Stories. Our findings are only concentrated on this first phase nevertheless other important practices are performed by the students in Step 2 of our approach.

The goal of the course is that students learn to define user stories by interacting with the Product Owner, identify epics and describe user stories correctly. The students should learn that a correct user story description must have role, desired feature, acceptance criteria and an adequate level of granularity; moreover, it may include non-functional requirements when necessary (Cohn, 2004). In case of complex user stories or epics, students should learn to apply user story splitting to obtain simpler user stories. Regarding user stories estimation, Release and Sprint plans are created by the Team during their respective planning meetings using the estimates assigned to the user stories (Cohn, 2005; Schwaber and Beedle, 2002). As a rule, these estimates are described by means of story points, which represent the complexity of a User Story. Students should learn that at the beginning of each Sprint, the Team estimates the number of story points that can be developed during a Sprint. Once a Sprint has started, the Team further decomposes each User Story into constituent tasks that must be performed to deliver a required functionality by the end of the Sprint. Students should also understand that the team is responsible for assigning the estimated duration in hours to each task of the Sprint. At this point, students are encouraged to analyze the relationship between story points and working hours. In doing so, they should become aware of the fact that this relationship is not a fixed one (e.g. 1 point does not equal to 8.3 hours). Moreover, they should learn the importance of estimating tasks in hours in order to confirm whether they have assigned an appropriate amount of work to the sprint. Another thing we mention is that tasks are to be carried out by one student at the time as they emerge throughout the Sprint. Regarding user stories planning, we provide the students with the well-established group estimation technique called Planning Poker recommended by agile software development methods for estimating the size of user stories and developing release and iteration plans (Mahnic and Hovelja, 2012).

Following our methodology, we provide the groups with the aforementioned Scrum concepts by instructional methods. One group is taught with a passive instructional method, and the other group with an active instructional method. Table 2 shows the organization of the topics by instructional methods; these instructional methods were designed by considering the FSLSM and related works (Graf et al., 2007; Felder and Silverman, 1988).

[1] http://www.ua.es/dpto/dqino/RTM/Invest_docente/ilsweb_es.html.

Table 2
Organization of the Scrum topics by instructional method.

	Main topic	Sub-topic	Practices for Reflective Students (Passive teaching style) After explain the concept, the professor takes a pause and allow students to think about…	Practices for Active Students (Active teaching style) After a brief explanation of the topic, the professor helps to …
1.	Where we stand in the Scrum framework?	-		
2.	User stories	2.1. User stories parts	… different roles that could need something from the system	… identify user stories' parts (e.g. role, acceptance criteria)
		2.2 When should you stop to disaggregate?	… a single feature to be developed	… elaborate user stories from a given requirement. Then, disaggregate them correctly
		2.2. Functional and non-functional requirements on user stories	… how to specify quality attributes in US	… add to the US quality attributes
		2.3. Epics	… if it is possible to develop the US	… identify a potential Epic from the USs
3.	Agile estimating	3.1. Size, velocity and duration of sprints	… how much work can the team complete along the Sprint?	… suppose different values of size, velocity of each team and calculate the number of Sprints estimated
		3.2. Story points and working hours	… why is important to understand that Story Points cannot be mappable to working hours	… discuss within the team about the relationship between story points and working hours
4.	Agile planning	4.1. Planning poker introduction	… a consensus-based game	
		4.2. Complexity units in the estimation	… a matter of comparing different opinions on the effort required by a task	… discuss within the team about the use of a fixed scale or a Fibonacci's scale
		4.3. How to proceed to perform planning poker?	… the synergy generated in the game and the estimation results	… perform planning poker in the group for assigning story points to each US previously identified
		4.4. Release plan	… how the team should organize all the features to be developed along the iterations	… organize the estimated US along the number of iterations previously calculated

- **Passive instructional method**

The passive instructional method is suitable for reflective students, who are enthusiastic about examining and manipulating the information introspectively. They prefer to think about and reflect on the material in-depth rather than put the concepts into practice. Besides, they work better alone or at most with one co-worker that can be an acquaintance or friend. As can be appreciated, the four main topics covered are an introduction to Scrum framework, user story parts and their disaggregation, agile estimating and agile planning. For each topic, we provide the students with a theoretical explanation of each sub-topic depicted in the Table 2. After the explanation, we take a short pause in order to let students think about the topic. Additionally, we trigger the reflection through an instructional intervention, such as "Can you imagine different kind of actors for the User Story?". Then, a student may answer the question and the professor makes a correction or explains the concept in-depth if necessary. Next, the professor continues with the same strategy for each sub-topic, explaining a concept, and then, making a short pause that allows students to reflect on the given idea.

The teaching of practices starts with an overview of the Scrum framework. Then, an introduction to user story specification and required information takes place. The professor takes a pause to engage students to think about different roles that could need something from the system used as example (Table 2 row 2, column 3). The next topic is user story splitting that consists in explaining different patterns to break down complex user stories or epics into simpler user stories. Here, the instructional intervention is "When should you stop to disaggregate user stories?", by which students think about different alternatives following their instinct. Then, the professor gives the students the appropriate answer to the question: a single feature to be developed is a possible way to detect a potential user story (row 3, column 3). After that, the next sub-topic is related to the management of functional and non-functional requirements within the user stories. In this context, an instructional intervention is "How can you model non-functional requirements?" so as to motivate students to use also the user story as a means to represent quality-attribute properties and technical constraints (row 4, column 3). After providing user story properties, the professor introduces the concept of Epics, i.e. large user stories that lack enough information to be developed. Given a requirement specification, students have to discuss if it is an epic or a user story by considering if they could develop it with the information provided (row 5, column 3).

Once user stories are defined, they are ready to be estimated. Thus, the professor introduces agile estimating and related concepts such as size, velocity and Sprint duration. The professor considers the following question: "How much work can the team complete along the Sprint?" (row 6, column 3). The professor focuses on the idea that a story point is a relative measure used for estimating size and the instructional intervention is Why is important to understand that Story Points cannot be mappable to working hours?" (row 7, column 3). After discussing the idea, the professor presents the Planning Poker technique as a consensus-based game (row 8, column 3), and introduces the aforementioned story points as complexity units. Based on the complexity units, team members are engaged to discuss and exchange opinions to achieve the estimate of a given user story (row 9, column 3). In this context, the professor makes other two instructional interventions: "Can you use the technique to estimate tasks?" and "What should you do when there is no agreement?". The former aims to indicate that Planning Poker is unsuitable to estimate tasks' working hours, whereas the latter aims to show that the Scrum Master is in charge of making a decision when the team fails to achieve an agreement when estimating. The professor insists on the idea that the synergy generated during the estimation process is crucial to achieve more accurate estimates (row 10, column 3). Finally, the last topic is agile planning that introduces students to the concept of release plan. Here, instructional intervention is "How should the team organize all the features to be developed along the iterations?" (row 11, column 3). After the students' feedback, the professor teaches the concept of velocity, i.e. the number of story points that a team can achieve at the end of each Sprint. Having this idea in mind, teams are able to plan how many iterations are necessary to complete the required user stories.

- **Active instructional method**

As for active students, we designed an instructional method by considering that these students prefer doing tasks or talking about concepts, and tend to directly process the information in order to

research on it, validate it or experiment with it. Likewise reflective students, topics covered are an introduction to Scrum framework, user story parts, user story disaggregation, and agile estimation and planning; however, the teaching strategies are significantly different.

As shown in Table 2 (column 4, Practices for Active Students), the instructional method starts with a brief introduction to the Scrum framework. Unlike reflective students, active students are given an opportunity to a hands-on experience along the instructional method. First off, they are organized into 5-member teams and are given a sample user story so as to detect its parts within 3-min time box (row 2, column 4). Afterwards, the teams are engaged to elaborate a user story for a requirement specification provided by the professor. Then, the professor introduces user story splitting, explaining alternatives to break down epics and complex user stories into simpler ones, and provides teams with a sample epic to be broken down into simpler user stories within 5-min time box (row 3, column 4). Later, the professor proposes adding quality-attribute properties to the user story within 2-min time box (row 4, column 4). The next step consists in showing students how to detect potential epics from a set of requirement specifications and deal with them to create user stories representing a single functionality within 7-min time box (row 5, column 4).

As in the case of reflective students, the professor introduces agile estimating and sizing to estimate the user stories. However, unlike the passive instructional method, the professor motivates students to suppose different values of size of user stories and the velocity of the team to calculate the number of Sprints needed to complete the user stories; this task should be performed within a 5-min time box (row 6, column 4). The professor explains that user story size is related to user story complexity, and measures this complexity in terms of story points. At this stage, teams of students are encouraged to analyze and discuss the relationship between story points and working hours. In doing so, the students should become aware of the impossibility to directly translate story points into working hours. This activity should be performed within a 5-min time box (row 7, column 4). In order to train students in widely used estimation techniques, the professor briefly explains Planning Poker and introduces the Fibonacci's sequence to compare user stories. For active students, the professor takes a 3-minute time box to let the teams of students discuss the use of a fixed scale or a Fibonacci's sequence (row 9, column 4). Afterwards, the professor provides each team with a set of 6 sample user stories belonging to the capstone project and asks teams to estimate those user stories within a 15-min time box (row 10, column 4). Team members are engaged to exchange opinions to achieve the estimates for the given user story. The technique requires team members to propose possible estimates for the user stories using cards with the Fibonacci's sequence 0.5, 1, 2, 3, 5, 8, 13, and 20 to anticipate the complexity of user stories. The professor explains that the use of Fibonacci's 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 et al., 2009). Following Planning Poker, team members select their cards simultaneously and those members with the highest and lowest story points have to justify their estimates. Then, all team members propose and estimation again until consensus is reached or the Scrum Master decides to finish the process after three iterations, choosing the average of story points as the estimate. Finally, the professor teaches agile planning and introduces the concepts of velocity and release plans. In contrast to the passive instructional method, the 5-min time-boxing hands-on activity consists in planning how many iterations are necessary to complete the given user stories. To carry out this activity, teams are asked to determine their velocity by considering team-member skills, commitment and experience. Then, bearing in mind the number of story points achievable at the end of the Sprints, each team elaborates a release plan including the number of Sprints and the user stories assigned to each Sprint (row 11, column 4).

3.2. Step 2: conducting the capstone project

The second step of the proposed methodology consists in simulating a professional software engineering context by means of a capstone project. Based on our previous experiences (Scott et al., 2014b; Soria et al., 2012) we agree with the idea that this project allows students to get a deeper understanding of Scrum. In this experiment, students can internalize the concepts learned in the previous step as they extrapolate them to a real small project by means of a balance between activities suitable for active and reflective students. Furthermore, students are encouraged to exercise skills different from their natural their learning preferences and face common issues that arise along a software development project in professional contexts.

3.2.1. Scrum teams formation

Prior to conducting the capstone project, students are organized into Scrum teams. In line with the idea of simulating a real professional context, we re-arranged students into Scrum Teams of 7–8 members at random in order to keep diversity and resemble real-life teams. Moreover, we made sure that each team had both active and reflective students by assigning them at random. This way, students can work in heterogeneous teams and they have the chance to broaden their understanding of Scrum. Once the Scrum teams are formed, they are ready to start the capstone project.

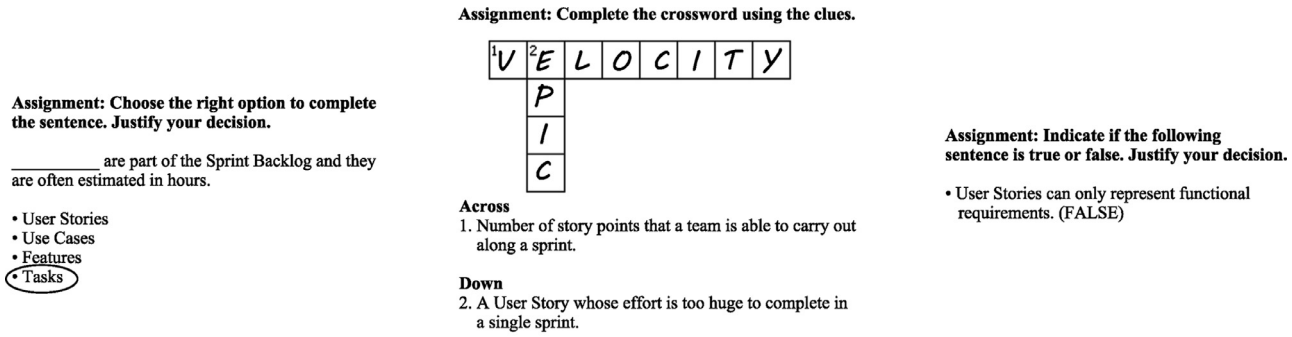
3.2.2. Describing the capstone project

The capstone project starts with the Sprint Planning phase. During this phase, the Product Owner and the Scrum Team select a subset of user stories from the Product Backlog to build the Sprint Backlog. The teams use Planning Poker, which demands individual commitment, as those responsible for developing a user story are the ones who estimates its complexity. The next phase is Development, in which Scrum Teams develop tasks associated with the user stories estimated in the previous phase. Each user story goes through a development process which consists in analyzing, designing, building and testing. In this context, the concept of "done" arises; a user story is considered "done" if the whole development process is fulfilled with the required artifacts.

At this stage, teachers introduce other important Scrum practices such as Daily Meeting, Sprint Review and Sprint Retrospective. A 15-minute Daily Meeting is held to report the project status; in this meeting, either face to face or on-line, students answer three questions: "What did you do yesterday? What will you do today? Are there any impediments in your way?". Twice a week, the Scrum Teams have a face-to-face meeting called Weekly Meeting (up to 15 min long) so that the students can show their tasks and estimates as well as reflect on impediments and improvements. In the next phase, the Sprint Review and Sprint Retrospective meetings are conducted. During the Sprint Review, the Scrum Team displays the user stories done during the Sprint by a demo to examine the work done and get feedback from the Product Owner. During the Sprint Retrospective, each Scrum team reflects on what happened during the Sprint and proposes strategies to improve the performance in subsequent Sprints.

3.3. Step 3: assessing the students' learning of Scrum

To assess the students' learning, we utilize a test elaborated by the authors, who have experience in software engineering teaching. The test is designed by using different kind of items to evaluate particular students' abilities. Fig. 1 shows the three types of exercises used in the test: multiple-choice, true-or-false quizzes and crossword. Multiple-choice items (Fig. 1a) allow professors to evaluate the ability to associate and compare basic Scrum artifacts, namely User Story, Scrum



(a) Multiple-choice. (b) Crossword. (c) True/False quiz.

Fig. 1. Examples of assignments used in the test.

roles, Product Backlog and Sprint Backlog, among others. Crossword (Fig. 1b) is used to evaluate the ability to learn and retain terminology commonly used in Scrum, such as velocity, story point, epic, user story splitting and focus, among others. True-or-false quizzes (Fig. 1c) are useful to evaluate the ability to reason and justify learned concepts; here, the exercises aim to measure how well Scrum practices are performed by the proposition of hypothetical scenarios.

To assess the quality of the test, we use item analysis. Using item analysis (Thorndike and Hagen, 1961) allows for retaining the test items that exert impact on the students' scores, and discarding those noisy items that fails to provide the research with valuable insights. In item analysis, the quality of an item is given by its difficulty and its discrimination index. The item difficulty (ρ) indicates the proportion of students who answered the item correctly, whereas the discrimination index (Δ) shows the correlation between the item performance and the overall test score. Therefore, the item difficulty is given by

$$\rho = \frac{n}{N} \tag{1}$$

where n is the number of students who responded the item correctly, and N the total number of students. The discrimination index is determined by

$$\triangle = \frac{M_{correct} - M_{total}}{SD_{total}} \cdot \frac{\sqrt{\rho}}{1-\rho}$$

where $M_{correct}$ is the mean score of whose answer correctly; M_{total} is the mean score total; SD_{total} is the standard deviation of all the scores; and ρ is given by Eq. 1. Items with extreme low ($\rho < .2$) or high ($\rho > .8$) difficulty, as well as items with low discrimination indexes ($\Delta \leq 0$) were discarded.

According to Pashler et al. (2008), the meshing hypothesis receives support, if and only if, the experiment reveals what is commonly known as a cross-over interaction between learning style and instructional method. Particularly in our approach, the cross-over interaction occurs when active/reflective students being taught with a suitable instructional method outscore active/reflective students being taught with an unsuitable method.

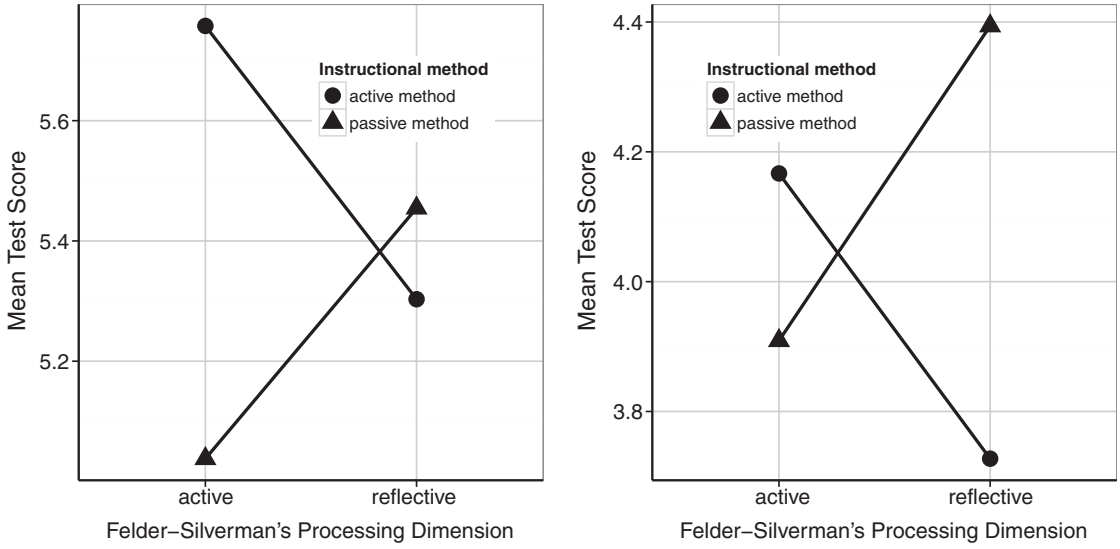
4. Application and evaluation of the approach

We applied our research approach in the context of the Software Engineering Workshop course. This one-semester elective course can be taken in the last year of the Systems Engineering BSc program at the Faculty of Exact Sciences (Department of Computer Science-UNCPBA) and aims to strengthen students' understanding of Scrum, to introduce the basics of Agile Methodologies, as well as, to get them acquainted with a professional context. In the previous years of the program students learn about software system design, object-oriented programming, operating systems and networks, and database management in order to guarantee all the students have the same prior knowledge. Thirty-five students took part in the experiment in 2013; then, we replicated it in the academic year 2014 with thirty-eight students.

We carefully applied our proposed methodology as follows. We used the ILS questionnaire and, by considering the processing dimension of the students' learning style, divided students into groups. Although we designed the experiment taking into account a balance between the number of active and reflective students in each group, we considered for the experiment only the students who attended the class. Secondly, we provided students' groups with the instructional methods and engaged students to carry out a capstone project. Thirdly, we designed a test containing 30 items: 5 multiple-choice, 10 true-or-false, and 15 crossword items. Then, we conducted the item analysis and provided the students with the test, which finally contained 22 items, scored as 1 point each: 4 multiple-choice, 9 true-or-false, and 9 crosswords. However, to achieve a better understanding of the scores, they were normalized into a 10-point scale. Finally, to visually corroborate our hypothesis, we utilized a graphic that shows the aforementioned cross-over interaction, as suggested by Pashler et al. (2008). This concept depicts the types of students' learning preferences (active or reflective) in the horizontal axis, and the average scores obtained by means of the instructional method (active or passive) in the vertical axis.

4.1. Results of academic Year 2013

The first step to apply the methodology consisted in utilizing the ILS to obtain the students' learning style. We found that 8 students (22.85%) were reflective, whereas 27 students (77.15%) were active. Then, we divided the students into two groups. The first group contained a total of 14 students, 2 of whom were reflective (14.28%) and the remaining were active (85.72%); and the second group contained a total of 21 students, 6 of whom were reflective (28.57%) and the remaining were active (71.43%). To corroborate our hypothesis, we gathered the scores obtained in the tests and calculated the arithmetic mean for reflective and active students by considering the instructional method provided. Fig. 2a illustrates the aforementioned cross-over interaction; the vertical axis shows the arithmetic mean of the test scores, whereas the horizontal axis shows the processing dimension values. The instructional method that improved the mean test score of one group of students is different from the instructional method that improved the mean test score of the second group of students. To reinforce the figure, Table 3 shows the results in terms of arithmetic mean ($\bar{x}$) and standard deviation (s^2) of scores for each instructional method applied to each students' learning preference. The standard deviation highlights how scattered are the exam scores of all students who received the class with the same instructional method. Regarding reflective students, the ones that were given the suitable instructional method (passive method) obtained a score of



(a) Results of the academic year 2013. (b) Results of the academic year 2014.

Fig. 2. Test mean scores of learning-style groups after the cross-over interaction with two learning-style-based instructional methods.

Table 3
Mean and standard deviation for the academic year 2013.

	Reflective students ($\bar{x} \pm s^2$)	Active students ($\bar{x} \pm s^2$)
Passive method	5.45 ± 0.64	5.04 ± 1.26
Active method	5.30 ± 1.24	5.76 ± 1.50

Table 4
Mean and standard deviation for the academic year 2014.

	Reflective students ($\bar{x} \pm s^2$)	Active students ($\bar{x} \pm s^2$)
Passive method	4.39 ± 1.40	3.91 ± 1.00
Active method	3.73 ± 1.80	4.17 ± 1.22

5.45 on average with a standard deviation of 0.64 (row 2, column 2), whereas the students that were given the unsuitable instructional method (active method) obtained a score of 5.30 on average with a standard deviation of 1.24 (row 3, column 2). With regard to active students, the ones that were given the suitable instructional method (active method) obtained a score of 5.76 on average with a standard deviation of 1.50 (row 3, column 3), whereas the students that were given the unsuitable class (passive method) obtained a score of 5.04 on average with a standard deviation of 1.26 (row 2, column 3).

Our data demonstrated that students who were given the Scrum class by means of the suitable instructional method obtained better score on average. As we argued before, we found evidence of the type of interaction between instructional method and assessment of an individual's learning style to support personalized teaching. Basically, we found that a student's learning experience was enhanced by a suitable instructional method.

4.2. Results of academic Year 2014

Similar to the academic year 2013, we firstly obtained the students' learning style by means of the ILS, and we found that 11 students (28.95%) were reflective and 27 (71.05%) were active. Then, we divided the students into two groups: the first group contained a total of 21 students, 6 of whom were reflective (28.57%) and the remaining were active (71.43%), and the second group contained a total of 17 students, 5 of whom were reflective (29.41%) and the remaining were active (70.59%). Fig. 2b illustrates the crossover interaction; the arithmetic mean of the test scores are depicted in the vertical axis, whereas the values of the processing dimension are shown in the horizontal axis. The instructional method that improved the mean test score of one group of students is different from the instructional method that improved the mean test score of the second group of students. In this line, Table 4 shows the results in terms of arithmetic mean ($\bar{x}$) and standard deviation (s^2) of scores for each instructional method applied to each students' learning preference. As for reflective students, the ones that were given the suitable class (passive method) obtained a score of 4.39 on average with a standard deviation of 1.40 (row 2, column 2), whereas the students that were given the unsuitable class (active method) obtained a score of 3.73 on average with a standard deviation of 1.80 (row 3, column 2). With regard to active students, the ones that were given the suitable class (active method) obtained a score of 4.17 on average with a standard deviation of 1.22 (row 3, column 3), whereas the students that were given the unsuitable class (passive method) obtained a score of 3.91 on average with a standard deviation of 1.00 (row 2, column 3).

These results reinforce the evidence that students who were given the Scrum class by means of the suitable instructional method obtained better score on average. In agreement with the first experiment, we found that a student's learning experience was enhanced by a suitable instructional method. After examining the results obtained from the experiments, it is possible to state that an instructional method was proven effective for students with certain learning preferences fail to be a one-fits-all method for students with different learning preferences.

5. Analysis of results

The proposed work aimed to corroborate the meshing hypothesis elaborated in the context of Software Engineering. To achieve this goal, we implemented an appropriate methodology as suggested by Pashler et al. (2008), which was applied in two experiments in the context of an undergraduate Scrum-based course. Firstly, we ran the experiment with 35 students of the Software Engineering Workshop course in the academic year 2013; and secondly, we replicated the experiment with 38 students in the academic year 2014. We have not only found evidence for the meshing hypothesis, but also provided professors with two instructional methods suitable for reflective and active students to teach user story splitting, agile estimating and

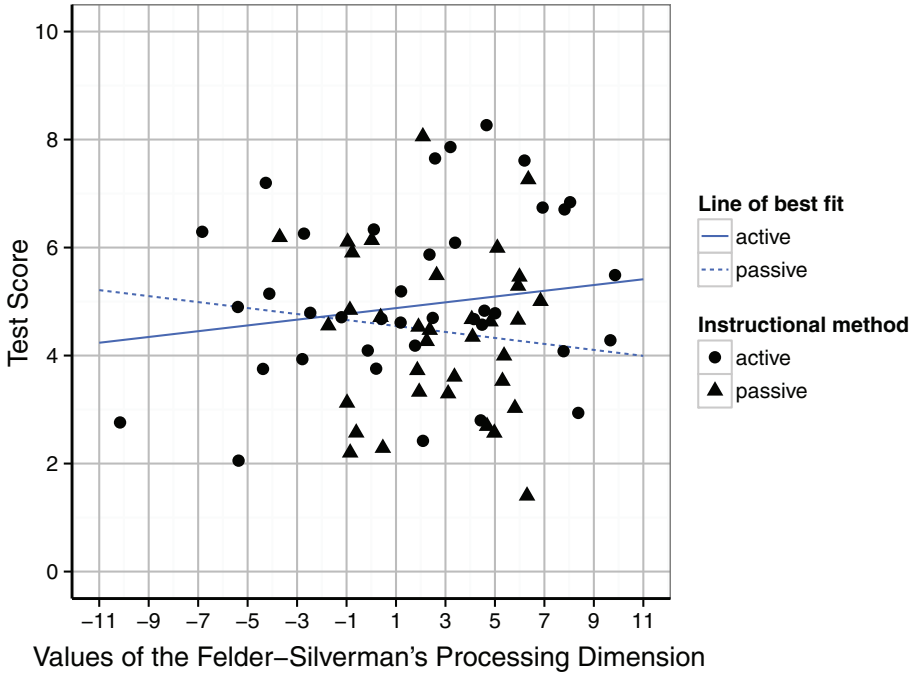


Fig. 3. Distribution of test scores from the academic years according to students' Processing Dimension.

agile planning. Besides, these instructional methods are aligned with the types of activities related to the processing dimension proposed by the FSLSM.

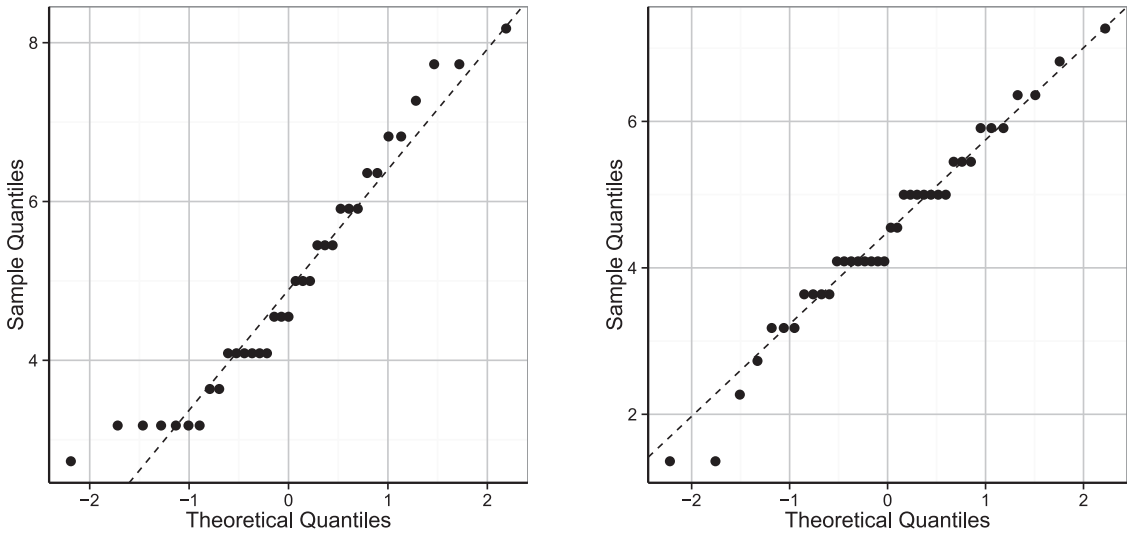
We recap the test scores from both experiments and organize them into a graphic to arrive at valuable conclusions. Fig. 3 summarizes the cross-over interaction of both case studies. Different from Figs. 2a and b, 3 shows the individual scores and allows for appreciating how the learning experience of a student is enhanced by being taught with the suitable instructional method when the processing dimension value is maximized. The scores are shown in the vertical axis and dots indicate the scores obtained by students who received the active instructional methods whereas triangles indicate the scores obtained by students who participated in the passive instructional method. The processing dimension values are shown in the horizontal axis: values from -11 to -1 mean reflective students, whereas values from 1 to 11 mean active students. Then, the dispersion of points across the graphic shows the students' preferences to learn: dots outliers on the right show that active students were taught by means of the active method and obtained the highest scores, whereas triangles on the left show that reflective students were taught by means of the active method and obtained the lowest scores.

With the aim to statistically corroborate the relation between the values of processing dimension of the students' learning style and the student's test scores by following the suitable instructional method, we analyzed Pearson's correlation coefficient (r) for both groups of students: those who received the active instructional method, as well as, those who received the passive one. To depict these relationships, we drew a line of best fit per instructional method over the dots in Fig. 3. The lines are the best approximation for the given sets of data and depict the correlation between the variables. Although the correlation values are slightly significant, the full line shows a positive correlation for the students who received the active instructional method ($r = 0.169$), whereas the dashed line shows a negative correlation for the those who received the passive one ($r = -0.151$). In other words, if we concentrate on students who received the active instructional method, the stronger the preference for active processing they have, the higher test scores they obtain. Similarly, if we concentrate on students who received the passive method, the stronger the preference for reflective processing they have, the higher test scores they obtain. It is worth nothing that the point cloud around the zero in the horizontal axis shows that those students behave similarly with the active or passive method. Furthermore, we can also observe that the cross-over interaction remains acceptable by the intersection between the dashed line and the full line provides substantial evidence to corroborate the meshing hypothesis in the two experiments.

Table 5
Mean test scores and standard deviation of students according to the instructional methods for both academic years.

	Reflective students ($\bar{x} \pm s^2$)	Active students ($\bar{x} \pm s^2$)
Passive method	4.66 ± 1.30	4.41 ± 1.24
Active method	4.59 ± 1.66	5.05 ± 1.58

Accompanying the Fig. 3, Table 5 summarizes the results of the two experiments together, by showing the arithmetic mean and standard deviation of test scores. As for reflective students, the ones that were given the suitable class (passive method) obtained a score of 4.66 on average with a standard deviation of 1.30, whereas the students that were given the unsuitable class (active method) obtained a score of 4.59 on average with a standard deviation of 1.66. With regard to active students, the ones that were given the suitable class (active method) obtained a score of 4.41 on average with a standard deviation of 1.24, whereas the students that were given the unsuitable class (passive method) obtained a score of 5.05 on average with a standard deviation of 1.58. By analyzing the experiments both jointly and separately, we observed two interesting findings. On the one hand, we observed that reflective students obtained higher scores on average than active students when taught by means of the passive instructional method, whereas active students obtained higher scores on average than reflective students when taught by means of the active instructional method. This finding corroborates the meshing hypothesis. On the other hand, it is worth to note that active students, who were taught by means of the active instructional method,



(a) Q-Q Plot for test scores of students who were taught with suitable instructional methods (group A).

(b) Q-Q Plot for test scores of students who were taught in a way that did not correspond to their learning style (group B).

Fig. 4. Q-Q Plots of test score distributions.

obtained higher scores on average than reflective students being taught by means of the passive instructional method; however, the standard deviation of the active students' scores is also higher. In this context, a line of research worth pursuing further is the study and analysis of this finding to obtain valuable insights.

Additionally, we studied the statistical significance of the difference in the mean scores of the two groups: students who were taught with suitable instructional methods (group A) and students who were taught in a way that did not correspond to their learning style (group B). To test this hypothesis, we carried out a *t-test* for independent samples that has several assumptions (Montgomery et al., 1984). The first one is that both samples are drawn from independent populations. The second assumption is that the populations can be described by a normal distribution. The third assumption refers to the variance homogeneity of both groups. The experimental design that was followed according to the suggestions of Pashler et al. (2008) satisfies the first assumption. To verify the second assumption, we analyzed the results from the Shapiro-Wilk normality test, concluding that both groups comes from a normally distributed population. It is worth noting that group A ($W = 0.939,\ p = 0.054$) has shown a less significant result than group B ($W = 0.974,\ p = 0.535$). However, the normality can also be graphically supported by the quantile-quantile (QQ) plot against the standard normal distribution as shown in Fig. 4a and 4b. Regarding the third assumption, to test the variance homogeneity, we carried out an *f-test* that shows reasonable evidence about the equality of variances in both groups ($F = 1.2512,\ p = 0.504$). After checking the assumptions, we conducted the *t-test* to statistically analyze the difference between the means of both groups. Although the mean of group A ($\bar{x}_A = 4.96$) is slightly greater than the mean of group B ($x_B = 4.46$), the results of the *t-test* show that there is no statistically significant difference between both means ($T = 1.489,\ p = 0.1408$). Moreover, to estimate the cost-effectiveness of aligning instructional methods with different learning styles, we resorted to Cohen's size of effect ($d = 0.241$). According to Cohen (1988), this value of d shows a small size effect on the students' outcomes obtained by applying the instructional methods. The size of effect of our experiment also supports that the *t-test* fails to reveal a significant difference between the means since the lower the size of effect is, the greater the size of the sample is needed. For all of the above, we consider that an issue worth pursuing further is augmenting the sample size.

Overall, we found evidence on the fact that students' learning experience is enhanced when they receive the suitable instructional method. However, there are a number of threats to validity in the evaluation of the proposed methodology. The first threat to validity is that we have only considered the processing dimension of the FSLSM since our study is based on a statistically significant relationship between the way Scrum practices are performed by the students and their processing dimension of the learning style (Scott et al., 2014a). Moreover, the second threat to validity is the lack of evidence of the impact of other instructional methods for the remaining FSLSM's dimensions on the students' outcomes. . Another threat to validity is related to the generalization of our results; since we used a 73-student sample, we are only able to draw conclusions for this particular population. That is to say, we have only studied the meshing hypothesis with Software Engineering students at UNICEN University (Department of Computer Science – UNCPBA, Tandil, Argentina). Thus, we can state that differences in cultural and educational backgrounds could impact on the suitability of instructional activities for the students. In a parallel way, the teaching was delivered by experienced professors on Scrum and learning styles. We consider that the impact of the teachers' ability and background on the design and implementation of instructional methods according to learning styles is worthy of further research.

After conducting our research, we are able to provide other professors with findings from the experimentation such as the teaching of some Scrum concepts (Table 2) by using suitable instructional methods. By considering our findings, students are expected to improve their outcome at the end of Scrum courses that include learning-style interventions. As a final note, we are still unable to assess costs and benefits of teaching Scrum by using our approach on a course as we only conducted an experiment to corroborate the meshing hypothesis. In fact, as future work, we are working on the design of a guide with suggestions to aid professors in aligning Scrum concepts with the FSLSM. Moreover, we are planning to obtain these insights in future research. Overall, the findings of our study allow for shedding light on the potential of using learning styles in the teaching of software engineering practices in the context of Scrum.

6. Conclusions

Teaching Scrum in undergraduate software engineering courses is an effective way to upgrade and add practical value to professional training. In this paper, we have corroborated that the students' knowledge of Scrum was improved when students were given suitable instructional methods according to the processing dimension of the students' learning styles. The experiments were carried out in the context of an undergraduate software engineering course and have provided evidence to support the meshing hypothesis. Particularly, the experiments have revealed that students who were provided with learning opportunities in line with their learning' preferences achieved better educational outcomes. Based on the experiments, a novel model of software engineering education, with pedagogical background in the FSLSM, was introduced. Furthermore, we provide professors with teaching strategies suitable for active/reflective students so that they improve their learning experience in the context of the Scrum methodology.

We consider that our findings represent a step towards better scrum learning by using learning styles, yet more experiments are needed to generalize our results. In this line, we are planning to widen the evaluation results by replicating more experiments in the software engineering course to enlarge our sample size. Hence, we expect to improve our findings by obtaining statistical significant results. Additionally, we are incorporating other assessment methods that consider performance metrics along the capstone project development. Furthermore, we are planning to consider other learning style dimensions as students' learning includes a combination of these preferences: input (visual/verbal), perception (sensing/intuitive), processing (active/reflective), and understanding (sequential/global). Another line of research worth pursuing is the study of alternative learning style models, which would allow us to obtain additional information about the personality of students in order to create optimal working teams. Along this line, we will analyze the performance of more experienced professional teams in software development so as to validate our hypothesis in other software engineering contexts.

Acknowledgments

We acknowledge the financial support provided by doctoral grants from Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET). We also thank to the students who participated in the experiments. Furthermore, we wish to thank the reviewers for providing valuable feedback to improve our work.

References

Ambler, S.W., 2006. Survey says: Agile works in practice. Dr. Dobb's J. 31 (9), 62–64.
Broman, D., Sandahl, K., Baker, M., 2012. The company approach to software engineering project courses. IEEE Trans. Educ. 55 (4), 445–452.
Carver Jr, C.A., Howard, R.A., Lane, W.D., 1999. Enhancing student learning through hypermedia courseware and incorporation of student learning styles. IEEE Trans. Educ., 42 (1), 33–38.
Cohen, J., 1988. Statistical Power Analysis for the Behavioral Sciences. Lawrence Earlbaum Associates.
Cohn, M., 2004. User Stories Applied for Agile Software Development. A-Wesley.
Cohn, M., 2005. Agile Estimating and Planning. Prentice Hall.
Constantinidou, F., Baker, S., 2002. Stimulus modality and verbal learning performance in normal aging. Brain Lang. 82 (3), 296–311.
Cook, D.A., Thompson, W.G., Thomas, K.G., Thomas, M.R., 2009. Lack of interaction between sensing–intuitive learning styles and problem-first versus information-first instruction: A randomized crossover trial. Adv. Health Sci. Educ. 14 (1), 79–90.
Coupal, C., Boechler, K., 2005. Introducing agile into a software development capstone project. In: Proceedings of the Agile Conference, 2005. IEEE, pp. 289–297.
Damian, D., Lassenius, C., Paasivaara, M., Borici, A., Schroter, A., 2012. Teaching a globally distributed project course using scrum practices. In: Proceedings of the Collaborative Teaching of Globally Distributed Software Development Workshop (CTGDSD), 2012, pp. 30–34.
Devedzic, V., Milenkovic, S.R., 2011. Teaching agile software development: A case study. IEEE Trans. Educ., 54 (2), 273–278.
Diaz, J., Garbajosa, J., Calvo-Manzano, J.A., 2009. Mapping CMMI level 2 to scrum practices: An experience report. In: Software Process Improvement. Springer, pp. 93–104.
Dingsoyr, T., Nerur, S., Balijepally, V., Moe, N.B., 2012. A decade of agile methodologies: towards explaining agile software development. J. Syst. Softw. 85 (6), 1213–1221.
Essalmi, F., Ayed, L.J.B., Jemni, M., Graf, S., et al., 2010. A fully personalization strategy of e-learning scenarios. Comput. Hum. Behav. 26 (4), 581–591.
Felder, R.M., Silverman, L.K., 1988. Learning and teaching styles in engineering education. Eng. Educ. 78 (7), 674–681.
Felder, R.M., Soloman, B.A., 1991. *Index of Learning Styles.* North Carolina State University. Retrieved October 22, 2015 from http://www.ncsu.edu/felder-public/ILSpage.html.
Felder, R., Spurlin, J., 2005. Applications, reliability, and validity of the index of learning styles.. Intl. J. Eng. Educ. 21(1), 103–112.
Feldman, J., Monteserin, A., Amandi, A., 2014. Detecting students' perception style by using games. Comput. Educ. 71, 14–22.
Graf S., K., Liu, T.-C., 2009. Supporting teachers in identifying students' learning styles in learning management systems: An automatic student modelling approach. Educ. Technol. Soc. 12 (4), 3–14.
Graf, S., Viola, S.R., Leo, T., et al., 2007. In-depth analysis of the Felder–Silverman learning style dimensions.. J. Res. Technol. Educ. 40 (1), 79–93.
Hossain, E., Babar, M.A., Paik, H.-y., 2009. Using scrum in global software development: a systematic literature review. In: Fourth IEEE International Conference on Global Software Engineering, 2009. ICGSE 2009. IEEE, pp. 175–184.
Keefe, J.W., 1988. Profiling and Utilizing Learning Style. ERIC.
Kuljis, J., Liu, F., 2005. A comparison of learning style theories on the suitability for elearning. In: Proceedings of the IASTED Conference on Web Technologies, Applications, and Services, pp. 191–197.
LeBlanc, R.J., Sobel, A., Diaz-Herrera, J.L., Hilburn, T.B., et al., 2006. Software Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering. IEEE Computer Society.
Lu, B., DeClue, T., 2011. Teaching agile methodology in a software engineering capstone course. J. Comp. Sci. Coll. 26 (5), 293–299.
Mahnic, V., 2010. Teaching scrum through team-project work: students' perceptions and teacher's observations. Int. J. Eng. Educ. 26 (1), 96–110.
Mahnic, V., 2011. A case study on agile estimating and planning using scrum. Elektronika ir Elektrotechnika 111 (5), 123–128.
Mahnic, V., 2012. A capstone course on agile software development using scrum. IEEE Trans. Educ., 55 (1), 99–106.
Mahnic, V., Hovelja, T., 2012. On using planning poker for estimating user stories. J. Syst. Softw. 85 (9), 2086–2095.
Martin Kropp, A.M., 2013. Swiss agile study - einsatz und nutzen von agilen methoden in der schweiz. Retrieved October 22, 2015 from www.swissagilestudy.ch.
Massa, L.J., Mayer, R.E., 2006. Testing the ati hypothesis: Should multimedia instruction accommodate verbalizer-visualizer cognitive style? Learn. Individ. Differ. 16 (4), 321–335.
McCombs, B.L., Whisler, J.S., 1997. The Learner-Centered Classroom and School: Strategies for Increasing Student Motivation and Achievement. Jossey-Bass San Francisco.
Miranda, E., Bourque, P., Abran, A., 2009. Sizing user stories using paired comparisons. Inf. Softw. Technol. 51 (9), 1327–1337.
Montgomery, D.C., Montgomery, D.C., Montgomery, D.C., 1984, Vol. 7. Design and Analysis of Experiments. Wiley New York.
Nerur, S., Balijepally, V., 2007. Theoretical reflections on agile development methodologies. Commun. ACM 50 (3), 79–83.
Paasivaara, M., Durasiewicz, S., Lassenius, C., 2009. Using scrum in distributed agile development: A multiple case study. In: Fourth IEEE International Conference on Global Software Engineering, 2009. ICGSE 2009. IEEE, pp. 195–204.
Paasivaara, M., Heikkila, V., Lassenius, C., Toivola, T., 2014. Teaching students scrum using lego blocks. In: Proceedings of the 36th International Conference on Software Engineering, pp. 382–391. URL: http://doi.acm.org/10.1145/2591062.2591169.
Pashler, H., McDaniel, M., Rohrer, D., Bjork, R., 2008. Learning styles concepts and evidence. Psychol. Sci. Publ. Interest 9 (3), 105–119.
Rico, D., Sayani, H., 2009. Use of agile methods in software engineering education. In: Proceedings of the AGILE '09., pp. 174–179.
Salo, O., Abrahamsson, P., 2008. Agile methods in european embedded software development organisations: a survey on the actual use and usefulness of extreme programming and scrum. Softw., IET 2 (1), 58–64.
Scharf, A., Koch, A., 2013. Scrum in a software engineering course: An in-depth praxis report. In: Proceedings of the IEEE 26th Conference on Software Engineering Education and Training (CSEE T), 2013, pp. 159–168.
Schroeder, A., Klarl, A., Mayer, P., Kroiß, C., 2012. Teaching agile software development through lab courses. In: Proceedings of the Global Engineering Education Conference (EDUCON), 2012 IEEE. IEEE, pp. 1–10.
Schwaber, K., Beedle, M., 2002, Vol. 1. Agile Software Development with Scrum. Prentice Hall Upper Saddle River.
Scott, E., Rodríguez, G., Soria, Á., Campo, M., 2013. El rol del estilo de aprendizaje en la enseñanza de prácticas de scrum: Un enfoque estadístico. In: Proceedings of ASSE 2013 Argentine Symposium on Software Engineering.
Scott, E., Rodriguez, G., Soria, A., Campo, M., 2014a. Are learning styles useful indicators to discover how students use scrum for the first time? Comput. Hum. Behav. 36, 56–64.
Scott, E., Rodríguez, G., Soria, Á., Campo, M., 2014b. Experiences in Software Engineering Education: Using Scrum, Agile Coaching, and Virtual Reality. In: Yu, L. (Ed.), Overcoming Challenges in Software Engineering Education: Delivering Non-Technical Knowledge and Skills. IGI Global, Hershey, PA, pp. 250–276.

Smith, T., Cooper, K.M., Longstreet, C.S., 2011. Software engineering senior design course: Experiences with agile game development in a capstone project. In: Proceedings of the 1st Int. Work. on Games and Software Engineering, pp. 9–12. URL: http://doi.acm.org/10.1145/1984674.1984679

Soria, A., Campo, M.R., Rodriguez, G., 2012. Improving software engineering teaching by introducing agile management. In: Proceedings of ASSE 2012 Argentine Symposium on Software Engineering, Vol. 8, p. 12.

Sternberg, R.J., Grigorenko, E.L., Ferrari, M., Clinkenbeard, P., 1999. A triarchic analysis of an aptitude-treatment interaction.. Eur. J. Psychol. Assess. 15 (1), 3–13.

Tan, C.-H., Tan, W.-K., Teo, H.-H., 2008. Training students to be agile information systems developers: A pedagogical approach. In: Conf. on Comp. Pers. Doc. consorti. and Research, pp. 88–96.

Thorndike, R.L., Hagen, E., 1961. Measurement and Evaluation in Psychology and Education. Wiley.

VersionOne, 2012. State of agile development survey results. Retrieved October 22, 2015 from https://www.versionone.com/pdf/7th-Annual-State-of-Agile-Development-Survey.pdf

von Wangenheim, C.G., Savi, R., Borgatto, A.F., 2013. Scrumia: An educational game for teaching scrum in computing courses. J. Syst. Softw. 86 (10), 2675–2687.

Ezequiel Scott received the Computer Engineer degree from Universidad Nacional del Centro de la Provincia de Buenos Aires (UNICEN), Tandil, Argentina, in 2012. He is currently pursuing the Ph.D degree in Computer Science at the same University. Since 2012, he has been part of ISISTAN Research Institute (CONICET - UNICEN), where has worked on projects related to software engineering, virtual reality and games for education. His research interests include virtual learning environments and agile methodologies. Mr. Scott has obtained a scholarship from CONICET to complete his doctoral studies.

Guillermo Rodríguez received the Computer Engineer degree from Universidad Nacional del Centro de la Provincia de Buenos Aires (UNICEN), Tandil, Argentina, in 2010, and the Ph.D degree in Computer Science at the same university in 2014. Since 2008, he has been part of ISISTAN Research Institute (CONICET - UNICEN), where has worked on projects related to software engineering, virtual reality and games for education. His research interests include software architecture materialization.

Álvaro Soria received the Computer Engineer degree from Universidad Nacional del Centro de la Provincia de Buenos Aires (UNICEN), Tandil, Argentina, in 2001, and the Ph.D degree in Computer Science at the same university in 2009. Since 2001, he has been part of ISISTAN Research Institute (CONICET - UNICEN). His research interests include Software Architectures, Quality-driven Design, Object-oriented Frameworks and Fault Localization.

Marcelo Campo received the Computer Engineer degree from Universidad Nacional del Centro de la Provincia de Buenos Aires (UNICEN), Tandil, Argentina, in 1988, and the Ph.D degree in Computer Science from Instituto de Informática de la Universidad Federal de Rio Grande do Sul (UFRGS), Brazil, in 1997. He is currently an Associate Professor at Computer Science Department and Director of the ISISTAN Research Institute (CONICET - UNICEN). His research interests include intelligent aided software engineering, software architecture and frameworks, agent technology and software visualization.