Innovación Educativa
ISSN: 1665-2673
innova@ipn.mx
Instituto Politécnico Nacional
México

Ventura Roque Hernández, Ramón; Herrera Izaguirre, Juan Antonio; López Mendoza, Adán; Salinas Escandón, Juan Manuel

A Practical Approach to the Agile Development of Mobile Apps in the Classroom

Innovación Educativa, vol. 17, núm. 73, enero-abril, 2017, pp. 97-114
Instituto Politécnico Nacional
Distrito Federal, México

Available in: http://www.redalyc.org/articulo.oa?id=179450594004

A Practical Approach to the Agile Development of Mobile Apps in the Classroom

Ramón Ventura Roque Hernández,
Juan Antonio Herrera Izaguirre,
Adán López Mendoza,
Juan Manuel Salinas Escandón
Universidad Autónoma de Tamaulipas, México

Abstract
This article presents a study where two groups of university students learned the principles of the agile development of mobile applications. The participating university students built their own version of an application in Java for Android following the principles of two agile methodologies: SCRUM and Extreme Programming (XP). Each team of students was assigned either a SCRUM or XP methodology for the development of their application in two iterations. In the second iteration the requirements were intentionally modified to provoke changes in the software being developed by each team. After the completion of the development process, a questionnaire was applied, and interviews with participants were conducted. The purpose of the questionnaire and the interviews was to gain insight into the participating students' perceptions about teamwork, the methodologies used, their personal motivation, and their attitude towards changing requirements. A Mann-Whitney test was performed on the acquired data. The results show that the team that implemented the XP methodology accepted the changing requirements more than the SCRUM team. Moreover, higher levels of participation and cooperation were observed among participants who used the XP methodology than among those who used SCRUM.

Keywords
Agile Software Development, Java Programming, learning Software Engineering, Scrum Approach, XP Approach.

Un acercamiento práctico al desarrollo ágil de aplicaciones móviles en el aula

Resumen
Este artículo presenta un estudio en el que dos grupos de estudiantes universitarios aprendieron los principios de la agilidad en el desarrollo de aplicaciones móviles y construyeron su propia versión de una aplicación en Java para Android siguiendo los principios de dos metodologías ágiles: SCRUM y Programación Extrema (XP). A cada equipo se le asignó una metodología: SCRUM o XP para el desarrollo de su aplicación en dos iteraciones, En la segunda iteración, los requerimientos fueron modificados intencionalmente para provocar cambios en el software que cada equipo creaba. Después de terminar el proceso de desarrollo, se administró un cuestionario y se condujeron entrevistas con el propósito de investigar la percepción

Palabras clave
Aprendizaje de la ingeniería del software, desarrollo ágil de software, programación extrema, programación en Java, SCRUM.

Recibido: 21/06/2016
Aceptado: 13/09/2016

de los estudiantes acerca de: el trabajo de equipo, las metodologías utilizadas, su motivación personal, y su actitud acerca de los requerimientos cambiantes. Se realizó la prueba de Mann-Whitney con los datos obtenidos. Los resultados muestran que el equipo que implementó la metodología XP aceptó mejor los requerimientos cambiantes que el equipo que usó SCRUM. Además, también se observó mayor participación y cooperación entre los participantes de la metodología XP que entre los que usaron SCRUM.

Introduction

Special activities related to agile software development should be promoted in universities with computer science programs to prepare students for situations they may encounter when creating new applications in the real business world. It should be kept in mind that fast and good quality software development is crucial in business applications, and students should be proficient enough to face this challenge successfully. Agile practices seem to be appropriate to software development performed in universities as part of their academic programs or as part of special projects for developing software for external companies. It is well known that in programming courses, students must develop a program for a specific problem, and in software engineering courses, the phases of software systems development are explained (Arman & Daghameen, 2007). That means that in programming courses, syntax, structures, and algorithms should be more priority than methodological aspects, without disregarding good practices. Moreover, good quality working software that is delivered quickly is a must in special university projects for companies. In these scenarios, agile practices help by focusing on rapid programming of functioning software that meets the specified requirements (Pressman, 2013).

There are many agile approaches; for example: Scrum, Extreme Programming, and Kanban (Singh, Mishra, Singh, & Upadhyay, 2015). Most of them have been extensively studied in real industry environments but not from university educational perspectives (Fuertes & Sepúlveda, 2016). Accordingly, we believe that agile software development is a broad concept that needs to be delimited and oriented according to specific educational requirements to positively influence the teaching of software development. An agile approach in university courses should foster rapid software development, promote application quality, and make changing requirements easy to manage. Additionally, an agile approach should also stimulate communication, work organization, active team participation, good relationships among students, and motivation for present and future learning.

This article presents a study aimed at comparing two popular software development approaches through a typical real-life busi-

ness simulation conducted inside a university lab. Two groups of students worked separately in building a mobile application within a tight deadline. The development was experimental and focused on programming software for the Android platform (Gironés, 2014) using Java (Friesen, 2014), following the agile methodologies SCRUM (Sims & Johnson, 2012) and Extreme Programming (XP) (Beck & Andres, 2005; Letelier & Penadés, 2006). The students committed themselves to deliver a functioning mobile app in twenty hours, even when they hadn't developed any application using agile approaches before. Students had previously developed software but with different tools and methodologies to those used in this study. The results from the students' perspectives showed that the SCRUM team had a greater preference for permanent requirements. They felt that much of the work had to be redone when dealing with changing requirements. On the other hand, XP seemed to promote the participation of the developers more than the SCRUM approach. These preliminary results provide some guidelines for further research in comparing methodologies and deciding which of them is more suitable for teaching programming courses under specific circumstances.

The paper is outlined as follows: first the background of the main topics that converge in this case study is presented; then the methodology used in this research is explained; later, the results are presented, and finally the conclusions are derived.

Related Work

Context

Agility allows the rapid construction of computer programs by adopting iterative and incremental models where analysis, design and construction activities are interspersed (Pressman, 2014). There are several methodologies with these features, all of which are based on a set of principles gathered in the agile manifest (Kendall & Kendall, 2013). Literature extensively discusses the philosophy, principles and practices of agile methodologies (Kendall & Kendall, 2013; Pressman, 2013; Beck & Andres, 2005). In this field, empirical research has focused more on industrial settings than on educational scenarios (Fuertes & Sepúlveda, 2016). Also, it must considered that the software engineering literature recognizes that not all facts found in literature are based on empirical evidence; for example, there are many procedures or techniques purported to be better than others based on opinions instead of real objective data (Juristo & M., 2001). On the other hand, software engineering is a practical area, where the teaching and learning process should not rely on the single automatic repetition of concepts or theoretical case studies, but should also involve activities that present scenarios to students to generate their own knowledge from new experiences.

Earlier studies

Agile development has been a topic of interest in several studies conducted in business and industrial scenarios during the last five years. For instance, Yetunde & Kolade (2015) studied collaboration and strategies used to integrate usability activities into a big scale agile project ; they found that some tactics like negotiating inclusion and establishing credibility were useful in succeeding in this process. On the other hand, Serrador & Pinto (2015) studied efficiency and user satisfaction in several industrial projects , and a positive effect of agile software methodologies was observed. Papatheocharus & Andreou (2014) used a questionnaire to study communication, management and quality assurance, aspects of agile teams. The results showed that agility improved the development process. McHugh, Conboy, & Lang (2012) analyzed several case studies in which the importance of human aspects in software development was highlighted. Their study observed that agile practices increased trust among programmers. In a previous work by McHugh, Conboy, & Lang (2011), a study involving three agile practices was conducted, and it was found that motivation was recognized to be highly important for the project and its leaders. In that research, it was also stressed that motivation is not addressed as much as other topics in the context of agile practices.

Agile software practices in the teaching field have also caught the attention of some researchers. For example, Salleh, Mendes, & Grundy (2014) studied the personality of students in pair programming practices; they found that openness was a significant factor to differentiate academic performance in students and that Pair Programming caused increased satisfaction, confidence and motivation in the class. Additionally, von Wangenheim, Savi, & Ferreti Borgatto (2013) and Rodriguez, Soria, & Ocampo (2015), studied the educational resources needed to teach SCRUM; they presented inexpensive games to reinforce the application of SCRUM. They reported that their approaches engaged students in the SCRUM activities inside the classroom; good motivation and good user experiences were described as well. Additionally, Devedzic & Milenkovic (2011) analyzed the experiences and problems encountered when teaching agile software development with SCRUM and XP in different scenarios and cultures; they learned that iterations should be short, that pairing up students helps them increase their motivation levels, that practices are useful to increase commitment among students, and that teams should be small and self-organized. Schroeder, Klarl, Mayer, & Kroiss (2012) studied the importance of lab practices when teaching agile approaches; they discovered that SCRUM was suitable to introduce students to software processes and that it motivated students by posing fun challenges to them. Kropp & Meier (2013), Kofune & Koita (2014), Soria, Campo, & Rodríguez (2012),

J. Faria, Yamanaka, & Tavares, (2012) presented their experiences in the teaching of agility and offered models to successfully address these types of courses. For example, Kofuna & Koita (2014) derived an approach to teach programming based on agile practices; they promoted critical thinking and communication among students using the trial and error practice. It was found that students were very motivated to learn. Moreover, in the work of Enríquez & Gómez (2015), a model for improving agile software development training in small companies is described; the model is based on SCRUM and consists on meetings, tasks, practices and steps. In the work of Kropp & Meier (2014), the pyramid of agile competencies that represents the different levels of competence needed for agility is derived; three levels are identified: Agile values, Management practices, and Engineering practices. They recommend practical approaches to acquire these competencies in the classroom.

The many reasons to teach agile software development in universities are discussed by Hazzan & Dubinsky, (2007). The most relevant are: Agility comes from and is used in the industry, it educates for teamwork, deals with human aspects, encourages diversity and supports the learning process. Nowadays, companies increasingly implement agile practices and experience the lack of skilled personnel (Kropp & Meier, 2013). Universities haven't been able to produce the appropriately skilled professionals (Kropp & Meier, 2014), and a gap exists between what is taught in the classroom and what is required by industry (Rodriguez, Soria, & Ocampo, 2015). This is why courses should be re-designed in order to teach students according to the demands of the software industry (Soria, Campo, & Rodríguez, 2012). In this process, the human side of software development should be taken seriously by academia because it is a field in which students have to make progress to benefit their transition to job market; this includes the organization of a development process, work coordination, and dealing with people with different skills, points of view, and motivations (Schroeder, Klarl, Mayer, & Kroiss, 2012). In the private sector, agility has surpassed the software development activities and is identified as a valuable tool for other fields such as management. Nevertheless, the lack of training is still a limitation that leads to misunderstands and undermines the benefits of agile work (Rigby, Sutherland, & Takeuchi, 2016).

Theory

Extreme Programming

Extreme Programming (XP or eXtreme Programming) is one of the most widely used agile methodologies (Rizwan, & Qureshi, 2011). It is an efficient, low risk, flexible, predictable, scientific,

and fun method for developing software (Beck & Andres, 2005). For example, XP encourages quick iterations for product delivery consisting of one to two weeks. XP requires the client and the team working together in the same place where the development takes place, and that the programming take place in groups of two people sharing the same computer (pair programming). XP also states that the size of the releases should be small but with a complete sense of value; in addition, it recommends the design of the system to be as simple as possible, the tests written before programming, and developers not to work overtime or take work home with them, and to participate in a symbolic ceremony at the end of each iteration.

Instead of creating long documents with functional requirements, an XP project starts making end users create software "user stories" that describe what new applications need to do. The requirements' test is done before coding and automated code testing is performed throughout the project. "Refactoring", frequent-design simplification, and improvement of the code, is also a basic tenet (Copeland, 2001). Devotees say the XP methodology helps generate code faster and with fewer errors.

SCRUM

SCRUM (Pressman, 2014) is an agile methodology that encompasses a series of iterative practices for developers to work as a team, contributing their individual skills to develop quality software. In SCRUM, software is incrementally developed, generating different versions, and at the end of each iteration, a functional end product is delivered. The customer can make changes or continue with development as was originally planned. The Sprint is the fundamental cycle or iteration of the SCRUM process. It is considered that two to four weeks is the most common amount of time for a Sprint.

In SCRUM there are meetings where the feedback process and the collection and clarification of requirements take place. These meetings are the daily Scrum, the sprint planning meeting, the review meeting and the sprint retrospective. The daily Scrum consists of a meeting at the beginning of each work session where each participant discloses what has been completed, what they expect to complete and any impediments found. The Sprint planning meeting is held at the beginning of iterations and consists of two parts: in the first part, the team commits to a set of goals; in the second, the team identifies specific tasks. The Sprint review meeting is where the team presents the completed requirements and the ones yet to be completed. The Sprint retrospective is a meeting that takes place at the end of each Sprint, where the team focuses on the lessons learned during the work accomplished in that iteration.

Pair Programming and Mob Programming

Pair Programming is one of the core practices of XP methodology. In Pair Programming, two people use a single computer to write and test computer programs. Since only one keyboard is available, the programmers change roles often. Globalization has introduced some new ways to do Pair Programming through technological platforms (da Silva Estácio & Prikladnicki, 2015); for example, when a team is scattered across different countries, they can use specialized software to implement the pair programming practice remotely. Another phenomenon studied with regards to Pair Programming is the knowledge transfer between programmers, and at least six different types of transference have been identified (Plonka, Sharp, van der Linden, & Dittrich, 2015). In research, Pair Programming in the academic field has not been addressed as much as in the industry (Prabu & Duraisami, 2015), and there is still a lack of consensus about its best use and its benefits (Coman, Robillard, Silliti, & Succi, 2014).

Mob Programming (Zuill, 2014) is a concept that recently emerged from practice as an evolution of pair programming. It consists of a whole team working on the same project, at the same time/space allocation, and using a single computer for coding. A projector or big screen is needed to amplify the image coming from the computer where the coding is performed. Some practices that have been reported in the implementation of Mob programming are: 1) treating each other with respect, 2) the driver and navigators roles when programming–the driver types the code, the navigators discuss and guide the driver, 3) frequent rotation of drivers, 4) communication is made as a team, and 5) involves periodical reflection on how to improve as a team.

The implementation of agile methodologies in the development of mobile applications

While software development in general can be more efficient with agile methodologies, software development for mobile devices, specifically, is an area that should be obligated to consider using agility because of the possible direct benefits from its implementation. For example, in the literature there are studies that suggest that the development of mobile applications should not be accomplished with a traditional methodology based on documentation or time-consuming processes, but should pursue the rapid attainment of a functional product considering the features of mobile applications.

(Blanco, Camarero, Fumero, Werterski, & Rodríguez, 2009; Abrahamsson, Ihme, Kolehmainen, Kyllönen, & Salo, 2005; Gasca Mantilla, Cmargo Ariza, & Medina Delgado, 2014; Ávila Domenech & Meneses Abad, 2013) Although agile software development emerged long before the mobile software development with its actual platforms, its principles can be implemented easily in the development of this particular type of software.

Methodology

A scenario was established in which several students would jointly build a fully functional, ready for delivery mobile application using an agile methodology that was assigned to them. None of the participants had any knowledge of the agile methodologies, and their experience with the tools used was scarce or nonexistent. Students had twenty effective hours to learn and develop the required application. The research questions that guided this process were: Can agile methodologies (XP, Scrum) really generate good quality working mobile applications in a short time with teams of university students? Is it possible that the members of these teams learn the basics of agile development methodologies and mobile software while producing functioning software in a short time? What is their perception of work relationship, methodology used, and motivation during this development process? How do they perceive the changing nature of the requirements, which represents a basic principle of agility? What are the differences between XP and Scrum in the software development that is conducted inside the classroom?

Sample

Work was performed separately with two groups of students. Each developed its own version of the same Android application using Java with Eclipse (Eclipse, 2015; Vogel, 2013). One team with seven students followed the agile methodology called Extreme Programming, and the other, with eight participants used SCRUM. They were university students from two different undergraduate programs related to computer systems that offer programming courses as part of their curricula.

The invitation to take part in a course outside the regular class schedule was extended to programming students who: 1) were close to graduating from undergraduate studies in computer science, 2) had a beginner or intermediate level of programming experience, 3) were able to code using at least one language, preferably Java or C, and 4) had enough knowledge about software design to work on the project.

Once the students responded to the invitation, they answered a questionnaire and a knowledge test of programming; it served for the selection of participants. Both the questionnaire and knowledge test gave an overview of the aptitudes and attitudes of the participants. Selection criteria for this study included good theoretical knowledge, ability to propose a design, and basic to intermediate coding skills. Both teams were randomly chosen from the selected participants. Prior to selection, participants did not know that this study was taking place. They were not given any financial compensation, but they were offered a certificate of participation, provided they had perfect attendance and punctuality.

Instruments

In order to get an approximation to the answers of our research questions, perceptions from participants and researchers were studied. At the end of software development, a five point Likert scale-based questionnaire was answered by the students (see Table 2), in which some statements were presented about team dynamics, methodology used, learning and motivation, and the personal perception of the development and changing requirements. The scale was numerically coded as follows: strongly disagree (0), disagree (1), neutral position (2), agree (3), strongly agree (4). Students were asked to provide responses about the approach as a whole and not only about specific practices performed during the development process.

Procedure

The process began with ten hours of training on Java for Android and agile standards. Each team, separately, received specific training on the basics of the methodology they were to use. The sessions were interactive and focused on this project's needs. After this training process, a second ten-hour period began during which the software was developed following the methodology assigned to each team. Programmers took on their specific roles. A member of the research team took on the role of a client with the teams and remained with the developers at their workplace.

The students listened to the client, who voiced the requirements as previously established by the research team (see Table 1). The goal was the development of a mobile Android application to help exercise mathematical reasoning. To get the final version of the software, each team completed two iterations. In the first iteration, the programmers were faced with a situation which had the requirements of Table 1 with the following exceptions: only two levels would be taken into account: beginner and advanced, they were not asked to include a help function for using the program and were asked that the program had a single screen to interact with the user. In the second iteration, the requirements were intentionally modified: now the app should show a different menu screen; it should also include a help function to guide the user, and it should have an intermediate level to generate arithmetic problems.

Students had to get the user stories and estimate times, discuss with the client the stories that would be implemented in each iteration, divide the stories into specific tasks, assign responsibilities and make adjustments, all the while adhering to the principles of the assigned methodology. XP team worked in pairs the entire time; they coded with a pair programming approach. SCRUM team worked in a self-organized way using an approach inspired by Mob Programming.

Table 1. Overview of App Requirements

1. It will have three levels of difficulty: beginner, intermediate and advanced.
2. The beginner level will generate numbers between 1 and 200
3. The intermediate level will generate numbers between 1 and 600
4. The advanced level will generate numbers between 1 and 1000
5. For each problem, users can select from one of three operations: addition, subtraction or multiplication
6. With the selected operation, the application will generate two numbers (operands).
7. The application will tell the users if they answer correctly.
8. When answering correctly, the application will emit a distinctive sound of success and display a suitable image.
9. If the user answers incorrectly, the application will tell the user that a mistake was made.
10. When the user makes a mistake, the application emits a distinctive error sound and display a suitable image.
11. The application must validate empty inputs by the user.
12. The application should count and display the number of attempts.
13. The application must allow the user to generate a new problem at any time.
14. The application must allow the user the ability to surrender to the problem posed.
15. If the user gives up, the application should display the correct result of the current operation.
16. The application should allow the user to start a new game at any time. Note: Start a new game means starting from scratch and resetting the current values of the correct answers and mistakes.
17. The application must give information about its programmers and date of development.
18. The application should have a help feature on how to use the program.

Source: Prepared by the authors, 2015.

At the end of software development, the questionnaire presented in Table 2 was answered by the students. Subsequently, these responses were entered and analyzed in SPSS (Field, 2013), in which a nonparametric Mann-Whitney test (Kuanli, Pavur, & Keeling, 2006) was performed for the difference between the responses of both teams. For this test, the following hypotheses were proposed for the responses to the 18 questions:

H_0: There is no significant difference between the perception of XP team and SCRUM team regarding the i*th* sentence.

H_a: There is significant difference between the perception of XP team and SCRUM team regarding the i*th* sentence.

A confidence level of 95% was used. If the PValue shown in Table 2 was less than .05, H_0 was rejected and a significant difference between the responses of both teams was assumed. Distributions of the scores for both groups were not assumed to be similar; therefore, mean ranks are presented in Table 2. A methodology having higher mean ranks in a statement exhibits a team with an attitude closer to the "strongly agree" value in regards to that sentence.

Unstructured interviews were also conducted with both students and researchers. Finally, the researcher who had adopted the role of client evaluated the final version of the software. Another user who was completely oblivious to the project team per-

Table 2. Survey results.

Statement	XP	Scrum	MW Test
	Mean rank	Mean rank	PValue
Team dynamics			
1. Problems were caused by the relationship between team members	7.79	8.19	.843
2. There was poor communication between project members	6.79	9.06	.282
3. We had difficulty making decisions	8.07	7.94	.951
4. We had trouble organizing	8.86	7.25	.471
Methodology			
5. The methodology favored the participation of all its members	9.93	6.31	.066
6. The methodology certainly favors the rapid development of mobile applications	6.86	9.00	.118
7. The methodology contributed to achieving a good quality program	8.14	7.88	.889
Learning and motivation			
8. I learned new things in the development of this software	8.50	7.56	.350
9. I'm motivated to keep learning more about this methodology	9.00	7.13	.170
10. I will use this methodology in future projects	9.00	7.13	.171
11. I was motivated at all time during the development	6.79	9.06	.200
Changing requirements			
12. When changing the requirements, I felt I had much work to redo.	5.64	10.06	.041
13. I would have preferred that the requirements didn´t change	5.71	10.00	.042
14. It is discouraging that the work has to be modified per a customer´s request	7.14	8.75	.428
Quantitative evaluation			
15. Team members motivation	8.50	7.56	.562
16. Communication among members	8.50	7.56	.625
17. Organization to work	8.00	8.00	1.0
18. Methodology's general efficiency	8.50	7.56	.625

Source: Prepared by the authors, 2015.

formed a usability test to the final version of the teams' software and shared his views on the app.

Results

Quantitative results

The Mann Whitney test that was run to determine differences in responses between XP and Scrum participants revealed a difference of opinion and feelings when it comes to having to revise work by modifying the requirements.

Scores for question 12 ("When changing the requirements, I felt I had much work to redo") in Scrum team (mean rank=10.06) were statistically significantly higher than in XP team (mean

rank=5.64), U=11.50, z=-2.04, p=.041. Also, scores for question 13 ("I would have rather that the requirements didn't change") in Scrum team (mean rank=10.00) were statistically significantly higher than in XP team (mean rank=5.71), U=12.00, z=-2.03, p=.042.

Some other important facts derived from survey results are:

a. The members of both teams unanimously agreed that they learned new things in the project, and that they were motivated to continue learning more about their agile methodology. They also agreed to continue using this methodology in future projects and to having felt motivated during the development process.
b. The minimum scores registered for communication, motivation, organization and overall efficiency of working with the SCRUM methodology were lower than with the XP methodology.
c. All XP team members disagreed with the raised statement that changes in the requirements involve too much rework for them.
d. Although SCRUM team members manifested more an opposition to the change in the requirements, both they and the members of the XP group expressed not feeling discouraged because users asked for changes in software development.
e. SCRUM team did not accept having difficulty organizing.
f. Contrary to the XP team, SCRUM team members agreed that the methodology contributed to developing a good quality software.

Qualitative results

During interviews, XP team members reported having worked in an agreeable atmosphere and being highly motivated to continue learning about XP. The celebrations held by the team at the end of iterations were very important, for it allowed them to relax and recognize their effort. The practice of not taking work home with them seemed very convenient, attractive and contrasting to the work requirements of many domestic software development companies today. The coding practices were perceived by the team as convenient and relevant to the app development. Pair programming made them feel very confident throughout the process. Nevertheless, they perceived that the different approaches to design and programming of each participant could have been a problem meeting the deadlines.

On the other hand, SCRUM participants mentioned that new aspects of software development were acquired by them in an interactive way. They felt that the project was completed quickly because of their joint participation that allowed them to do several activities simultaneously. SCRUM team also reported the lack

of patience of some members, and the different levels of participation among them.

While recognizing that this development was their first encounter with agile methodologies and mobile development, XP and SCRUM members were open to work with these methodologies in the future. They also reported not having troubles when managing the artifacts or activities related to their methodology.

The researchers reported that the implementation of XP and SCRUM managed to timely produce software that met the specified requirements. They realized that participants were involved in their work and that they were committed to meet the specified requirements of the software being developed. The researchers perceived that the SCRUM team was concerned with managing carefully method, tools and artifacts while the XP team had a more relaxed attitude about the process. On the other hand, although both teams assigned the complex programming tasks to their most proficient members, XP participants were more involved in every programming task due to the interaction promoted by the pair programming practice.

The final programs delivered by both teams were approved by the client. In addition, an outsider conducted a usability test to both applications and reported no trouble in using them. He referred that the software developed with XP was intuitive and easy to use, but the app developed with SCRUM was not very user friendly and could have been more intuitive.

Discussion

In this study, a comparison between SCRUM and XP development approaches was sought in a university scenario. The responses provided by the participants through the questionnaire showed that the SCRUM team had a higher preference for permanent requirements than the XP team. They also felt that they would have to redo much work to comply with the changes in requirements to a greater extent than the XP team. There were no statistically significant differences in the rest of the research aspects.

The observed differences may be influenced by the coding practices (pair programing, test first, refactoring, small deliveries, continuous integration, collective code ownership, simple design, and coding conventions) which are emphasized in XP, but not in SCRUM. XP team may have perceived that implementing changing requirements in the application was easier because of the benefits provided by these practices. Differences may also be due to the way in which the SCRUM methodology is organized. Meetings are held and artifacts have to be administered and monitored. SCRUM team with a tight deadline to meet may have perceived that changing requirements would need extra time to organize

the process. It is important to highlight that XP team reported having perceived the coding practices as convenient and relevant, and researchers reported having perceived that methodology, tools and artifacts were matters of concern for the SCRUM team.

XP teams work in pairs because pair programming is a core practice of this methodology; SCRUM methodology allows teams to choose their own organization for coding. In this study, SCRUM team decided to use an approach influenced by "Mob Programming", and XP team used the pair programming practice. Nevertheless, it is important to note that Scrum and Mob Programming are not attached to each other. In this study, the Mob Programming influenced approach was used as a result of SCRUM team's decision. To prevent confusion in the SCRUM team, participants were instructed to focus their answers on the SCRUM approach instead of on Mob Programming practices. Differences and boundaries were specifically stated before they provided their responses. XP team was also instructed to report on the complete methodology and not on isolated practices.

According to researcher's perceptions, XP participants were more relaxed about the development process than SCRUM participants. This attitude could have been promoted by the practices of not taking work home, working in pairs most of the time, and holding celebrations after each iteration. XP team members provided feedback and explanations to each other as often as it was necessary. This collaborative work led to embrace the change with the confidence that the rest of the team was available to implement any changes or to provide useful information to update the code.

It must be taken into consideration that not all participants knew each other, and they each had a particular way of conceptualizing the collaborative software development. This is a common situation in regular programming courses; nevertheless, students still need to communicate with others when dealing with changing requirements in order to modify the application. The methodology could help them in this process by promoting their participation. Although it was not a statistically significant result, SCRUM received the lowest scores when participants rated the level of participation that the methodology favored.

This study focused on a preliminary comparison between XP and SCRUM approaches with regular university students who were available and willing to be trained outside their regular class schedule. Participants had standardized skills in software development, and were able to attend four hour sessions daily. It is also important to point out that this was a small project with few participants and a short development time. Further research should be conducted to replicate this study; also, students, sessions, and projects with different attributes should be analyzed.

From the findings derived from this study, we believe that XP was the most appropriate methodology for promoting participation

among students. Nevertheless, SCRUM proved to be an effective and easy-to-follow methodology as well. XP may be suitable for shorter course sessions, and SCRUM may be suitable for longer sessions. XP can be used in pure programming courses; meanwhile SCRUM can be used in courses focused on methodological approaches. If changing requirements are expected, XP could be a better approach for students to accept and update their applications easily; this empirical evidence supports the claim "XP adapts to vague or rapidly changing requirements" in the work of Beck & Andres (2004). In using both methodologies in the classroom, scope of projects, number of students, type of course, session schedules, and available facilities should be considered. For example, as stated in the work of Rodriguez, Soria, & Ocampo (2015), we identified that using a single room for several teams working with SCRUM at the same time in a face-to-face course may not be feasible if the room is not properly equipped. On the other hand, from these empirical findings we also confirmed that expected competencies in students can be met through team projects based on real scenarios that stimulate technical and human aspects of software development. A holistic approach including theory, practice, content, methodology, and experiences should be used in the learning of agility. This will promote the skills and values needed to complete software projects and will help students to take an active role in their education. Teamwork and commitment from students were relevant elements that were brought to our attention during this study. These views are consistent with those derived by Kropp & Meier (2014), Sancho-Thomas, Fuentes-Fernández & Fernández-Manjón (2009), Hazzan & Dubinsky (2007).

Conclusions

This article reported on a case study in which XP and SCRUM were evaluated from educational perspectives in a university setting. Two groups of students developed their own version of the same mobile application in Java for Android; one group used XP, and the other group implemented SCRUM.

Results revealed that both Extreme Programming and SCRUM can produce good quality software in a short period of time and can be implemented easily with university students with little experience in mobile application development. Students learned the basics of agile development and mobile programming and built a useful application for a client; they were motivated to use these methodologies in future projects. They perceived a pleasant relationship among the team members and reported a good perception towards both methodologies. In regards to the changing of requirements, XP team accepted the changes easier than the SCRUM team.

In university courses, students need to develop working software without disregarding good practices; they also need to be exposed to experiences that lead them to pertinent learning experiences for present and future challenges. Students should be helped by their teachers in achieving these goals. Since agile software development is extensively used in industry, it needs to be addressed in university courses, and teachers should evaluate different approaches to present the most suitable scenarios for learning agile practices during the course. This evaluation must be based on empirical evidence and the concrete attributes of the learning environment in which agility will be taught. These evaluations will help to succeed in the teaching and learning processes of software development, a field that has been identified as complex because of the several competencies students have to develop.

References

Abrahamsson, P., Ihme, T., Kolehmainen, K., Kyllönen, P., & Salo, O. (2005). *Mobile-D for Mobile Software: How to Use Agile Approaches for the Efficient Development of Mobile Applications.* Finland: VTT Technical Research Centre of Finland.

Arman, N., & Daghameen, K. (2007). Teaching Software Engineering Courses: When? *The International Conference on Information Technology .*

Ávila Domenech, E., & Meneses Abad, A. (2013). DelfDroid y su comparación evaluativa con XP y Scrum mediante el método 4-DAT. *Revista Cubana de Ciencias Informáticas*, 16-23.

Beck, K., & Andres, C. (2005). *eXtreme Programming explained. Embrace Change.* United States: Addison Wesley.

Beck, K., & et., a. (2001). *Manifesto for Agile Software Development*. Retrieved on May 1st, 2016 from http://agilemanifesto.org/

Blanco, P., Camarero, J., Fumero, A., Werterski, A., & Rodríguez, P. (2009). *Metodología de desarrollo ágil para sistemas móviles. Introducción al desarrollo con Android y el iPhone.* Madrid: Universidad Politécnica de Madrid.

Coman, I. D., Robillard, P. N., Silliti, A., & Succi, G. (2014). Cooperation, collaboration and pair-programming: Field studies on backup behavior. *The Journal of Systems and Software*, 124-134.

Copeland, L. (2001). *Extreme Programming.* Retrieved on January 15th, 2016 from Computer World: http://www.computerworld.com/s/article/66192/Extreme_Programming

da Silva Estácio, B. J., & Prikladnicki, R. (2015). Distributed Pair Programming: A Systematic Literature Review. *Information and Software Technology*, 1-10.

Devedzic, V., & Milenkovic, S. R. (2011). Teaching Agile Software Development: A Case Study. *IEEE Transactions on Education, 54*(2), 273-278.

Eclipse. (2015). *Eclipse*. Retrieved on January 15th, 2016 from http://www.eclipse.org

Enríquez, C., & Gómez, P. (2015). A Model for Improving Training of Software Developers in Small Companies. *IEEE Latin America Transactions, 13*(5), 1454-1461.

Field, A. (2013). *Discovering Statistics using IBM SPSS Statistics.* SAGE Publications Ltd.

Friesen, J. (2014). *Learn Java for Android Development.* United States: Apress.

Fuertes A., Y., & Sepúlveda C., J. (2016). Scrum, Kanban and Canvas in the commercial, industrial and educational sector - A literature review. *Revista Antioqueña De Las Ciencias Computacionales*, *6*(1), 46-50.

Gasca Mantilla, M. C., Cmargo Ariza, L. L., & Medina Delgado, B. (2014). Metodología para el desarrollo de aplicaciones móviles. *Tecnura*, 20-35.

Gironés, J. (2014). *El gran libro de Android.* España: Marcombo.

Hazzan, O., & Dubinsky, Y. (2007). Why Software Engineering Programs Should Teach Agile Software Development. *ACM SIGSOFT Software Engineering Notes, 32*(2), 1-3.

J. Faria, E. S., Yamanaka, K., & Tavares, J. A. (2012). eXtreme Learning of Programming - A Methodology Based in eXtreme Programming to Programming Learning. *IEEE Latin America Transactions, 10*(2), 1589-1594.

Juristo, N., & M., A. (2001). *Basics of Software Engineering Experimentation.* Springer.

Kendall, &Kendall. (2013). *Systems Analysis and Design*. United States: Prentice Hall.

Kofune, Y., & Koita, T. (2014). Empirical Studies of Agile Software Development to Learn Programming Skills. *Systemics, Cybernetics and Informatics, 12*(3), 34-37.

Kropp, M., & Meier, A. (2013). Teaching Agile Software Development at University Level. *IMVS Fokus Report*, 15-20.

Kropp, M., & Meier, A. (2014). New Sustainable Teaching Approaches in Software Engineering Education. *2014 IEEE Global Engineering Education Conference (EDUCON).* Istanbul, Turkey.

Kuanli, Pavur, & Keeling. (2006). *Introduction to Business Statistics*.South-Western College Pub.

Letelier, P., & Penadés, M. (2006). Metodologías ágiles para el desarrollo de software: eXtreme Programming (XP). *Técnica Administrativa, 5*(26).

McHugh, O., Conboy, K., & Lang, M. (2011). Using Agile Practices to Influence Motivation within IT Project Teams. *Scandinavian Journal of Information Systems*, 85-110.

McHugh, O., Conboy, K., & Lang, M. (2012). Agile Practices: The Impact on Trust in Software Project Teams. *IEEE Software*, 71-76.

Papatheocharus, E., & Andreou, A. S. (2014). Empirical evidence and state of practice of software agile teams. *Journal of Software: Evolution and Process*, 855-866.

Plonka, L., Sharp, H., van der Linden, J., & Dittrich, Y. (2015). Knowledge transfer in pair programming: An in-depth analysis. *Int. J. Human-Computer Studies*, 66-78.

Prabu, P., & Duraisami, S. (2015). Impact of Pair Programming for Effective Software Development Process. *International Journal of Applied Engineering Research, 10*(8), 18969-18986.

Pressman, R. (2014). *Software Engineering: A Practitioner's approach* (6 ed.). United States: McGrawHill.

Rigby, D., Sutherland, J., & Takeuchi, H. (2016). Embracing Agile. *Harvard Business Review*, 40-50.

Rizwan, M., & Qureshi, J. (2011). Agile software development methodology for medium and large projects. *IET Software*, 6(4), 358-363.

Rodriguez, G., Soria, Á., & Ocampo, M. (2015). Virtual Scrum: A Teaching Aid to Introduce Undergraduate Software Engineering Students to Scrum. *Computer Applications in Engineering Education, 23*(1), 147-156.

Salleh, N., Mendes, E., & Grundy, J. (2014). Investigating the effects of personality traits on pair programming in a higher education setting through a family of experiments. *Empirical Software Engineering, 19*(3), 714-752.

Sancho-Thomas, P., Fuentes-Fernández, R., & Fernández-Manjón, B. (2009). Learning teamwork skills in university programming courses. *Computers & Education, 53*(2), 517-531.

Schroeder, A., Klarl, A., Mayer, P., & Kroiss, C. (2012). Teaching agile software development through lab courses. *Global Engineering Education Conference (EDUCON), 2012 IEEE.* Marrakech.

Serrador, P., & Pinto, J. K. (2015). Does Agile work? - A quantitative analysis of agile project success. *ScienceDirect*, 1040-1051.

Sims, C., & Johnson, H. L. (2012). *SCRUM: Abreathtakinly Brief and Agile Introduction.* Lexington, KY, EU: Dymaxicon.

Singh, G., Mishra, A., Singh, H., & Upadhyay, P. (2015). Empirical study of agile software development methodologies: a comparative analysis. *ACM SIGSOFT Software Engineering Notes, 40(*1).

Soria, Á., Campo, M. R., & Rodríguez, G. (2012). Improving Software Engineering Teaching by Introducing Agile Management. *13th Argentine Symposium on Software Engineering.*

Vogel, L. (2013). *Eclipse 4 Application Development: The complete guide to Eclipse 4 RCP development.* Vogel.

von Wangenheim, C. G., Savi, R., & Ferreti Borgatto, A. (2013). SCRUMIA - An educational game for teaching SCRUM in computing courses. *The Journal of Systems and Software*, 2675-2687.

Yetunde, A., & Kolade, W. (2015). Integrating usability work into a large inter-organisational agile development project: Tactics developed by usability designers. *The Journal of Systems and Software*, 54-66.

Zuill, W. (2014). *Mob Programming: A whole Team Approach.* Retrieved on February 2nd, 2016 from http://www.agilealliance.org/files/6214/0509/9357/ExperienceReport.2014.Zuill.pdf