Faculty of Engineering and Technology

Master of Software Engineering (SWEN)

Master Thesis

Empirical Study: Teaching Agile Software Development

By:

Alaa Hantoli

Supervised by:

Dr. Abdel Salam Sayyad

This Thesis was submitted in fulfillment of the requirements for the Master's Degree in Software Engineering from the Faculty of Engineering and Technology at Birzeit University, Palestine

June 25, 2020

BIRZEIT UNIVERSITY

Empirical Study: Teaching Agile Software Development

Author: Alaa Hantoli

This thesis was prepared under the supervision of Dr. Abdel Salam Sayyad and has been approved by all members of the examination committee:

Dr. Abdel Salam Sayyad, Birzeit University

[illegible]

____

Dr. Yousef Hassouneh, Birzeit University

[illegible]

____

Dr. Sobhi Ahmed, Birzeit University

[illegible]

____

Date approved:
June 25, 2020

Declaration of Authorship

I, Alaa Hantoli, declare that this thesis titled, "Empirical Study: Teaching Agile Software Development" and the work presented in it are my own.

I confirm that:

- This work was done wholly or mainly while in candidature for a master degree at Birzeit University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: Alaa Hantoli

Date: June 25, 2020

Abstract

Agile has been successfully adopted by many software companies and it is the most popular methodology in industry nowadays. However, our universities give more attention to teaching Waterfall model in related courses with a bit coverage of Agile main characteristics. In this thesis, we work on the setup, execution, and results of teaching a Software Engineering course to undergraduate students with a specific focus on Agile practices, through official re-constructed lectures besides open workshops with a senior engineer from industry to follow up with students in parallel. In addition to improve the students' technical, management and social skills, and compared to other related works, this research investigates many factors affected or have been affected by Agile and hold many significant comparisons, also it gives additional focus on some agile practices –not covered yet- as non-functional requirements. This research was designed to study the impact of adopting the Agile Software Development Methodology in teaching, on students understanding and practicing of software engineering. And then to overcome the potential problems and highlight any raised side effects. The results show the high satisfaction of the students through the experiment, also show a sufficient evidence to conclude that there is a significant difference in the means of improvements between the experimental and control groups in understanding and applying software engineering and Agile methodology in specific.

Keywords: Agile Software Development; Scrum; Software Engineering; Undergraduate.

الملخص

تم تبني منهجية (آجايل) لتطوير البرمجيات من قبل العديد من شركات البرمجيات، وهي المنهجية الأكثر شيوعاً في الأيام الحالية. ومع ذلك، فإن جامعاتنا تولي الاهتمام الأكبر في مساقاتها المتعلقة بهندسة البرمجيات لتدريس النموذج التقليدي الذي يعتمد بدوره على التسلسل في عملية تطوير المنتج البرمجي، مقابل تغطية محدودة للخصائص الرئيسية للـ(آجايل).

في هذه الأطروحة، نعمل على إعداد وتنفيذ وتحليل نتائج تدريس مساق هندسة البرمجيات للطلاب الجامعيين، من خلال مشاريع برمجية ينفذها الطلاب اعتماداً على مبادئ وقيم وخطوات منهجية (آجايل)، بالإضافة للمحاضرات الرسمية، وبالتوازي مع ورش العمل مع مختص من سوق العمل لمتابعة أداء الطلاب.

بالإضافة إلى تحسين المهارات التقنية والإدارية والاجتماعية للطلاب، ومقارنةً بالأبحاث الأخرى ذات الصلة، يبحث هذا العمل في العديد من العوامل المؤثرة أو المتأثرة بمنهجية (آجايل)، ويجري عدد من المقارنات. كما تم تصميم هذا البحث لدراسة تأثير اعتماد منهجية تطوير البرمجيات هذه في التدريس، على فهم الطلاب وقدرتهم على تطبيق هندسة البرمجيات. ثم للتغلب على المشاكل المحتملة وتسليط الضوء على أي آثار جانبية بارزة.

تظهر النتائج الرضا العالي لدى الطلاب خلال التجربة ، كما تظهر أدلة كافية لاستنتاج أن هناك فرقاً معنوياً في متوسطات التحسن بين المجموعتين التجريبية والضابطة في فهم وتطبيق هندسة البرمجيات والـ (آجايل) على وجه الخصوص.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

SWE	**S**oftware **E**ngineering
ASD	**A**gile **S**oftware **D**evelopment
XP	e**X**treme **P**rograming
CC	**C**omputing **C**urricula
TDD	**T**est-**D**riven **D**evelopment
PTUK	**P**alestine **T**echnical **U**niversity
SRS	**S**ystem **R**equirements **S**pecification
CSE	**C**omputer **S**ystems **E**ngineering
ILO	**I**ntended **L**earning **O**utcomes
IDE	**I**ntegrated **D**evelopment **E**nvironment

Chapter 1 Introduction

The term Agile Software Development (ASD) has evolved opposite of plan-centric development, its agility come from being specially designed to accelerate software delivery to the client, and to be responsive and accept rapidly changing requirements and integrate them to the product, increase productivity as well as ensure software high quality and minimal development overhead [1].

1.1 *Introduction and Motivation*

Agile becomes mainstream and it is the public approach in software development nowadays. More than half software corporations are developing using an agile methodology such as XP and Scrum [6]. It is somehow considered the standard industry practice within teams. It was evolved and applied by industry [5].

Moving to our universities where the software engineers and developers come from, they primarily teach technical subjects such as programming, data structures and databases, algorithms and modeling. As they are essentials, also the human side can highly assist students in the transition into the software market and to act effectively. It is critical in our universities to learn the manner of organizing the process of development, treat with varied-skill teams, and how to output outstanding software in spite of strict deadlines and a forty hours of work weekly [2].

Although ASD is popular in industry for years, but universities only covered as a part of undergraduate or graduate courses such as Software Engineering and Software Project Management courses. Teaching classical methods like Waterfall is the main topic in these courses, and sometimes it mentions ASD presence and its common properties. Globally, few courses are devoted to ASD, as optional courses.

Teaching Agile methodologies usually covers one of them, such as Scrum or XP. Because it is hard for a comprehensive learning of set of methodologies and practices within the course period [5]. Our approach is to follow the Scrum strategy in the course project. A general view of other Agile methods: Kanban, Lean, Crystal and others; will be covered in the course outline.

Developing a project committing to a fully applied software development methodology during the course; convinces students that software engineering is useful for practical purposes, and students will believe in its effectiveness in real world, helping them in their future career [14].

Universities are a risk-free academic environment allow a fully experiencing of agile software development processes. This helps us to improve the students' technical and social skills effectively, and build an Agile mindset. Also, helps to investigate whatever factors we intend to study in this convenient educational environment.

Also, there will be additional focus on some agile practices –not covered yet- like non-functional requirements i.e. usability by developing a low and high fidelity prototypes for evaluation.

1.2 *Research Objectives and Problem Statement*

This research was designed to study the impact of adopting the Agile Software Development Methodology in teaching, on students understanding and practicing of software engineering.

In addition to improve the students' skills (social, technical and management), we intend to exploit the resources and the experiment itself as well, to compare the quality (in means of time and bugs) between agile and traditional model, in order to help student convince that is learning agile deserves their effort to learn.

Adopting agile in teaching require a shift in the pedagogical model, so we built our plan of teaching on two main actions: give more attention to agile through the course; and employ the project-based teaching and supportive workshops instead of the traditional classes.

So our research objectives are the following:

- By applying the proposed pedagogical model, this research will firstly discuss whether we can significantly improve the quality of teaching output represented by the students understanding and practicing of agile methodology in SWE?
- Also hold a comparison between Waterfall Model and Agile Model against specific criteria, including the bug rate and the percentage of implemented features.
- And to find if students are satisfied of learning the SWE course through projects with a more focus on agile and learning new tools?

1.3 *Overview of this thesis*

The report is organized after the introduction in the following chapters:

Chapter 2 gives a background about the Agile Software Development Methodology, and specifies Scrum again, with an explanation why we choose to adopt the Scrum in Teaching rather than other Agile Methods. The chapter also discusses the pertinent literature and sources available in order to implement our experiment, from where the other research have stopped.

Chapter 3 outlines the research methodology which was followed to collect and analyse the data.

Chapter 4 analysis and discuss the results of the experiment.

Chapter 5 shows a small conclusion about the research and the findings on the literature review, recommendation and future work.

Chapter 2 Background and Literature Review

In this chapter, we will review relevant background and term definitions, and related research work in the area of teaching agile software development.

2.1 *Definition and Background*

2.1.1 Agile

Agile software development (ASD) has been formally introduced by a group of software practitioners and consultants in 2001 in the “agile manifesto”, which establishes four fundamental values for agile software development: individuals and interactions over processes and tools, working software over comprehensive documentation, customer collaboration over contract negotiation, and responding to change over following a plan [1]. In addition to twelve principles behind the Agile Manifesto, Agile people follow. These principles are the following:

- The highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

- Working software is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity--the art of maximizing the amount of work not done--is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly [11].

Agile classified as a lightweight methodology in where the method Agile is incremental which releases a small software increments in short frequent iterations, it is also iterative which allows re-planning and re-estimating as needed, cooperative as it emphases the communication between the customer and developers, straightforward because it has only a few rules and practices, that are easy to follow and it is well documented, and adaptive such that let developers able to deal with last moment changes. [12]

More than half of the resources for the traditional project are spent before any development work even begins. Furthermore, requirements change before development even starts [4]. Where Agile emphasize leading concepts, helps a project team adapt fluently to the unpredictable and rapidly changing requirements, and risk is minimized by focusing on short iterations of distinctly defined deliverables.

The different Agile Software Development Methods are the following: extreme programming [26], scrum [27][28], crystal family of methodologies [29], feature driven development [30], the rational unified process [31], dynamic systems development method [32], adaptive software development [33], open source development [34], Agile modeling [35] and pragmatic programming [36]. They all share the same philosophy, characteristics and practices. However, from the implementation perspective, each method has its own combination of terminology and practices.

We are looking between them for an easy method to comprehend, which makes it ideal to introduce agility to undergraduate students. Also we focus on the management aspects of projects. And which has fast iterations and active collaboration within the team. So we chose Scrum for its ability to incorporate various overarching practices promoted by other Agile models. And due to its proven productivity, and its popularity in Palestine market. And some other details discussed in the preparation section.

2.1.2 Scrum Methodology

The most important agile approaches are: Scrum, extreme, adaptive and dynamic project management method. Of these, the most used is Scrum [4]. In terms of agile, a Scrum is simply an agile, lightweight process for managing and controlling software and product development in rapidly changing environments. For example, Scrums are intentionally iterative, incremental processes that are predicated on a team-based approach. Given that systems today are usually development in fluid and rapidly changing environments, one of the major reasons for using an iterative process is to help control the chaos that can result from conflicting interests and needs within the project team. Additionally, iterative processes are used to help enable improvement in communication, maximize cooperation, as well as protect the team from disruptions and impediments. Overall then, the goal is to deliver a more suitable product more quickly than with traditional methods.

Over the years, a number of agile frameworks, such as Extreme Programming and Scrum, have evolved and matured. The underlying philosophy of Scrum recognizes that the customers often change their mind about the product they want and that the development challenges are unpredictable by their nature [8]. Consequently, Scrum embraces the fact that the problem being solved cannot be fully understood or described from the start. Instead, Scrum focuses on maximizing the ability of the development team to quickly deliver in response to emerging requirements.

The Scrum model is built on three major components: roles, process, and artifacts [4] [12]. We will explain them here, because we will employ each detail of scrum process in our students' projects.

Scrum Roles

Roles are clearly defined and not boundaries-crossed. The Scrum Master looks a like a team leader, he is responsible for various things, most notably are enacting the Scrum values and practices (Daily Meeting, Planning Session, etc.), and removing impediments. The Scrum team typically is a cross-functional team of five to ten full-time members. The team is self-organizing, which has been interpreted in diverse ways, but most often means that the leadership role within the team changes depending on the needs of the current iteration (sprint). This change may occur between sprints only. The product owner is typically a functional unit manager who knows what needs to be constructed and the sequence of the progress because he is the person who will contact with the customer.

In the execution section we will clarify how we define each role, and how we assign them to team members.

Scrum Process

The Scrum process has five major activities: the kick-off, the sprint planning meeting, the sprint, the daily Scrum, and the sprint review meeting. The product is incrementally developed in time-framed sprints of two to four weeks, see figure 2-1 (Andreas Schroeder, 2012) that explains the whole process [2].

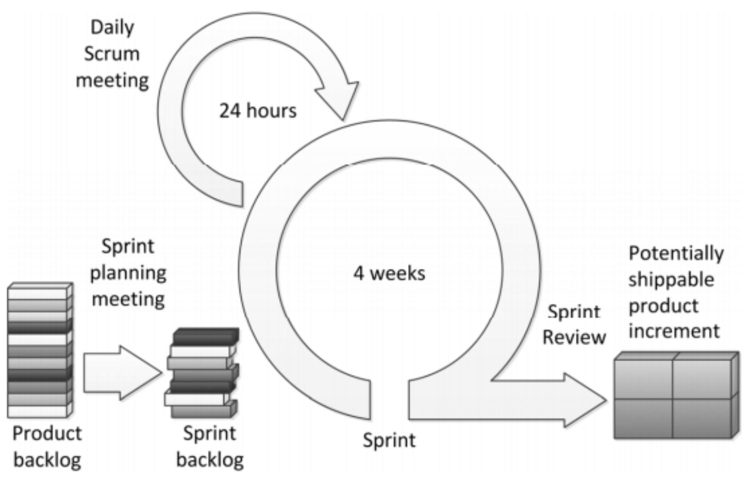


Figure 2-1. Scrum Process Loop (Andreas Schroeder, 2012)

The sprint planning meeting is a meeting of the Scrum team, the Scrum master, and the product owner at the beginning of each sprint. These meetings may require a day. In the first part of this meeting, the group defines the product backlog, which is basically a list of the project requirements. After this, the group defines its goal, which is the formal outcome(s) from this individual sprint. In the second part of the meeting, working on creating the sprint backlog. The kickoff meeting is structured similarly to the sprint planning meeting but defines the high-level backlog and the major project goals. The sprint starts after the sprint planning meeting completed. Sprints differ from phases in a traditional waterfall method in that sprints are month-long at most. Another characteristic, no outside influence should be allowed to interfere with the work of the Scrum team during a sprint. This saves the project requirements from changing during a sprint [2]. It is worth to say again Scrum still accepted changes, but they are being planned in the next sprints without affecting the current running one. Scrum master calls for daily Scrum meeting with Scrum team, which lasts about 15 minutes as maximum, every team member briefly answers three questions:

(1) What did you do since the last Scrum?

(2) What are you doing until the next Scrum?

(3) What is stopping you getting on with your work?

The daily Scrum is not a problem solving session nor individual assessment. But to track the progress of the team as well as allow team members to make commitments to each other and the Scrum master, who maintains the whole work going ahead properly. The sprint review meeting is held at the end of each sprint, to demonstrate the created functionality to the product owner. It is might be different from the traditional meeting in being informal and not distractive.

Scrum Artifacts

Scrum artifacts comprise the product backlog, the sprint backlog, and burndown charts. The product backlog is the project requirements expressed as a prioritized list of backlog items. This list formed using project management software (such MS Project) or as a spreadsheet. And it is managed and owned by the product owner. The product backlog is a prime output of the kickoff or sprint planning meetings. During the sprint planning meeting, the team performs an estimation of each product backlog item. Two methods of review are typically used, expert review or creating a work breakdown structure, using in both the story points. Then set up the team's velocity or amount of effort that can be handled during one sprint according to the estimation. Velocity is the result of the division of Agile story points delivered by the number of sprints.

Similarly, the sprint backlog is the subset of product backlog items for a particular sprint. It is created only by the Scrum team members. Ideally the sprint backlog is updated every day and contains no more than 300 tasks. The team may need to break down a task if it is determined that it will take more than 16 hours. Furthermore, the team may determine that items may need to be added or subtracted from the sprint but this is the team's decision, it is not something that is directed by the product owner.

Unlike traditional project management, Scrum intentionally focuses on work done through the use of burndown charts. Three types of burndown charts are commonly used: the sprint burndown chart documenting the progress of the sprint, the release burndown chart documenting the progress of the release, and the product burndown chart documenting the overall project progress. A goal of a burndown chart is to provide information in an easy to comprehend manner. As such, each task is typically represented in terms of time (the x-axis of the display grid) and duration (the y-axis). For example, a typical sprint burndown chart would depict the total backlog hours remaining in the sprint per day as an estimated amount of time left in the sprint. Ideally, the sprint burndown chart would “burn down” to no time remaining by the end of the sprint; however as during the sprint, setbacks could result in an increase in estimated time, not all burndown charts do burndown to zero. The release burndown chart functions in a similar way but represents the remaining time until the release will be done. Not surprisingly then, the product burndown chart is used to indicate the overall project progress.

Some warnings should be clarified before going on a Scrum or other Agile project. Periodic communication does not compensate documentation in all, without sufficient control, a project could decay because documentation is not maintained. Another point, in order to efficiently apply an agile approach, all stakeholders must be committed to the process. Also focusing on removing needless bureaucracy, makes it possible for all parties actually do productive work [4].

2.2 *Literature Review*

Our literature shows that research related to our topic of Agile teaching published in many publication sources, either journals or conferences. Regarding the years of publication, we did not find any studies related to our research topic prior to 2001, they were published in Extreme Programming and Agile Processes in Software Engineering Conference [14], and ASEE/IEEE Frontiers in Education Conference [15]. They were immature enough, and most of them was concerned XP. After few years some other studies published in Scrum and other methods. About authors, we observed that for most of them from North America and Europe (this does not mean a null participation or not significant from other locations), whether possessed by one author, or multi authors from the same university, or from different universities and colleges like in [8], or co-authors affiliated to different countries within a single study.

However, in Arab World we cannot find except Nuha's paper executed in a Jordanian university [9]. She has designed a course for undergraduate students, in terms of Problem-Based Learning (PBL), the main goals of this work are to practice one of the Agile software development methods, which is chosen to be Scrum, the second purpose is to develop the entrepreneurial skills necessary for software engineers, that “The Engineer of 2020” book mentioned, like: creativity, lifelong learning, leadership, etc. Then she intended to examine the correlation between PBL characteristics and Scrum practices.

In teaching Agile software development, the plurality of regarding work is typically group-based projects, and teaches Scrum and/or XP [5]. The researcher Nuha goes on another approach -even similar- attempting to get the best output from binding with Agile teaching. The research have been reported in related literature, differed in how effective it is in this context. This research is one of the latest we found in this specific field. The author denoted she has resulted in a successful experience of PBL as students show up. In general, the positive papers had evidence that the Problem-Based-Learning very helpful in: Enriching the skills of collaboration, communication and management; Improving criticality of learners; Enhancing deep learning; Prompting professional identity and responsibility development.

In few countries, the ASD laboratories and centers at universities have started to appear, for teaching this methodology to students. For example: "The University of Texas" in 2006 and "Bowling Green State University" in 2008. Furthermore, teaching ASD to professionals is now well developed and going to be expanded. The following are examples on those both.

An experiment of Teaching ASD through Lab Courses held in Germany [2], they presented the setup, implementation, and results of two quite succeeded Scrum labs carried out in 2010 and 2011. As all other related works, they state that the adopting of agile methods in lab courses was succeeded. Previous to this German research [2], two issues have set to be mostly preventive to the success of the labs - but authors override in their experiment- : Firstly, academic instructors tend to introduce advanced research topics into their labs. Secondly, too much time consumed on the functionality of the software itself to be built rather than benefit from it as a harmonious development approach. So authors, emphases the simplicity of the product, means not algorithmically complex, although maintaining its appealing, to not lose the student motivation, and to understand it instantly, in order to focus on Agile as software development methodology from the kickoff.

Each of the two labs over the two years was engaged just six students, who had given one project. The students found out how to achieve customer's needs tied to determined time, to overcome changes and to self-organize. Authors' main outcomes are: Agile benefits students' social skills as well as technical skills, as appear in vital collaboration and teamwork. Other key points, using a fun challenge for student motivation, and teacher should prepare an appropriate development environment- settings and plans- for a quick start. Also, Authors have found the Scrum to be perfect for introducing software engineering processes.

They faced some problems: often, the analysis phase stays longer than predicted, which reflects on the students and form their own vision about analysis that only delaying the coding phase, then should be avoided, and that project organization and infrastructure taught in the course are unable to assuring process success. Their approach was to emphasize more on continuous process during the lab, over the accomplishment the software product.

Although ASD has been applied in industry for many years, a typical case at most universities is that it is only studied within undergraduate or graduate courses. Traditional methodologies are often included in these courses, and in some cases mentions ASD existence and no more general properties. Scarce courses are devoted to ASD exclusively, and teach Scrum and\or XP of ASD several approaches, and these are usually elective.

In [5], Authors display and argue their experiences through ten years (2010-2017) of teaching a novel intensive ASD methods, the course was planned a week long, as portion of a Masters of Software Engineering program. Where students are software engineering professionals who already employed within industry and probably have a degree in a computing topic, or have substantial industry experience.

This work debate how students -experience Agile values and management practices- to foster an Agile mindset. They had a positive feedback from the participated students. The other important outcome is the course design and material they had, which may be adopted where required in other studies performed on master or bachelor degree.

The course design able to provide information to students, and to enable the Agile mindset, this be realized through integration between different manners in education, even if in formal educational settings. The outline describes the pre and post-course assignment, case studies, lecture content, group exercises. They put further emphasis on learning-by-doing with hands-on exercises and class discussion – of some case studies; presented and run by the students themselves-. Each of the exercises attempted to teach valuable aspects; for example, the estimation exercise, it put the use of abstract story points into practice on non-software artifacts to help understand that estimation is a team effort and not a formula that is uniformly applied. A pre-study assignment is an individual assignment helps teacher prepare appropriate plan, where the post-course to assess the students benefit [5].

In [1] authors pointed A new IEEE standard, P1648, that will provide a firm basis as well as directions for future computing curricula (CC), that time they said it was still under development, but when one refers now to IEEE-SA Standards Board [16] there no thing related ASD.

2.2.1 Early Studies

One of the early literatures on teaching Agile is referenced as [14], this paper reports on the practice of the ASD methodology for designing and implementing a simple Java application for graduated students in an Information Technology university department in Italy. The course is made of fifteen 3-hour weeks, the last third of weeks was of practice activity, others of classes.

The paper proposes an agile approach for teaching Software Engineering based on the strict collaboration of students and teachers in design activities and focusing on the management of groups, time-table and meetings. The paper reports positive student feedbacks and comments.

They gave care on teacher existence during meetings to monitor teams, and encourage the groups to attend each other meetings, so teachers may observe some bad practices they cannot see anywhere else, these faults are important to learn about and avoid. On the other hand, teachers may suggest a valuable ideas or behaviors.

They recommend two points, which all subsequent papers confirm. The first one, practices should consist of just one assignment consists of all software engineering aspects, where the past courses practices were based on a set of small activities as a scalable and flexible way for managing each. Also from the vision of the students, they usually had sensed software engineering just as a set of separated and not linked activities that hardly fit together to effectively handle bigger problems. Secondly, creating heterogeneous groups by the teacher, according to student capabilities, background experiences and interests. Which leads to homogeneity among groups.

As a result, which is also common result in all later studies, it is clearly observed that both students and teachers are very motivated and interested in adopting ASD in their courses.

The experiment was immature enough, they do not show a detailed plan of how to integrate the agile practices and principles through the course and project, for example they done the estimation without following any clear method or instructions. They have used technical (design and coding) tools, but no management tools. Furthermore, the evaluation was based on a written final exam with exercises and theoretical questions instead of having formative evaluation over the project period.

2.2.2 Why Teaching Agile is Necessary

In [7], based on authors' comprehensive teaching and research experience in the subject of ASD, both in the industry and in the academia, throughout five years;

Their contribution is clarifying the reasons necessitate SWE programs to teach ASD, as follows:

1. Agile was evolved and becoming utilized widespread in software industry, its teaching in the academia is just a natural response.
2. Agile deals with human aspects since two of the agile manifesto ideas: "Individuals and interactions over processes" and "Customer collaboration over contract negotiation".
3. Agile naturally taught in a teamwork-oriented environment, there is no need to introduce the topic of teamwork artificially.
4. Agile promotes diversity, with global software development, diversity distinguishes the teams' formulation, since diversity is extracted from Agile principles (e.g. customer collaboration) and is revealed by Agile practices (such as informative workspace, pair programming and planning game).
5. Agile supports learning processes. Two techniques reflect this, small releases and refactoring.
6. Agile develops mind habits. As reflection, abstraction skills and program understanding. Such skills can be enhanced by activities such as stand-up meetings, pair programming, and small releases.
7. Agile emphasizes management skills. While Agile is taught at university level, students gain some software management skills. According to the fact that all team members contribute the responsibility for the developed product, not the team leader.
8. Agile consolidates ethical norms as well: Agile manifesto adheres "The Software Engineering Code of Ethics and Professional Practice" formulated by an ACM/IEEE-CS. For example, "software engineers shall act in a manner that is in the best interests of their client and employer, consistent with the public interest." match the "Customer collaboration over contract negotiation". They both give priority to the customers' interest.
9. Agile highlights an overall image of software engineering, as it concerning various fields, like cognition and management.
10. Agile environment provides a single all-inclusive teaching framework for software engineering.

2.2.3 Other Studies

A case study accomplished in 2011 [1], describes authors' eight years of experience in teaching agile software methodologies to various groups of students at different universities, in different cultural settings, and in a number of courses and seminars. The authors provide recommendations on how to overcome potential problems in teaching agile software development and make their adoption more effective. Here are their recommendations in the light of problems they encountered -and which our study fully cared about-:

- They discover that the problems of refactoring, testing, and design are a major obstacle to fostering ASD, so it is helpful to get rid of these barriers in the beginning of the course.
- It is sufficient for the iterations to be one to two weeks' length, and this allows for more iterations.
- Agile can be attractive, but it is not a silver bullet and not the best for all cases.
- There is must not be too much theoretical argumentation, students must involve into practice to keep up the motivation. And to increase the adherence, every decision should be taken with them.
- More monitoring gets students unsatisfied.
- They advise teams to be small. So in our case we can match this testament and the group heterogeneity in [14] to conclude, if there is a team of inexperienced students, teacher should not add some skilled members to it, instead teacher should rearrange whole groups and distributed weak students among other groups, because large groups are troublesome to manage, rearrangement also helps to maintain the homogeneity between groups.

They have remarked some obstructions and were describing them unmanageable and difficult to remove: Personal relations between students, lack of knowledge, and the university setting can sometimes rise further inefficiency. As well the pairing can be a crucial problem; personal incompatibility within pairs may decelerate the team dramatically thus require special care and interference by the teacher.

2.2.4 Tools

There are some deadlock in taking the advantage of available tools, it is a clear weakness at mentioned experiments, for instance, authors in [2] did not provide any platform for communication between teams, and in this case students will begin to organize meetings to avoid the distributed team effects, depending on a group solution i.e. yahoo groups.

They also have required students to use SVN which was not accepted by the students. Teachers should carefully choose the tools for UML, version control and other processes, tools must be correctly function and in the same time easy to use, to intensify students' efforts on Agile practices. And not been exhausted or losing their time learning tools and solving its problems. For example, instead of SVN we intend to choose Git as a tool for version control, where merging is easier and straight forward.

Beside scrum practices, it is important to find an open and positive learning platform for farthest knowledge transfer that improves the programming and design skills of the students. Next are some helpful tools authors have not used, but they have advised others to use: Coding Dojos, which a meeting where a group of coders get together to complete a programming challenge within allocated time, and audience can repeat it at home by themselves. [17] [18], i.e. using them with focus on example-based learning, learning from each other.

2.2.5 Games

A rising tendency in teaching Scrum is the use of simulation games- like card games, in order to facilitate the transition from the theory to practice. Although some literatures as [25] consider the 2D games inadequate design to supply an immersion experience and factual presentation of Scrum environment.

In [2], Lego4Scrum (ex. City) activity was conducted. Lego4Scrum project can be finished in three hours, since all meetings and the sprint interval are shrink to five minutes. This game [19] teaches agile thinking and explain the Scrum framework with Lego. And cause of restricted time and powerful interaction, students in each team cognize others and learn how they may self-organized during the entire lab course, in addition to understand how to operate under pressure as a team.

Different from card games, Virtual Scrum uses a virtual world to mimic a real business environment treating 3D displays of the Scrum artifacts.

In [3], Authors devise a Virtual Scrum -a tool to enrich the learning experience of Scrum- to get rid of limitations in time, large classes, and facilities within formal education settings. The students had to develop a capstone project following Scrum as a homework of two phases. In the first phase, the students used available tools such as Jira, Excel and Microsoft Project to develop the user requirements, while through the second stage they exercised Virtual Scrum to develop different set of equivalent user requirements. Finally, authors gathered users' opinions by filling a survey- free of neutral mid-points, reported that the tool is useful to enhance the understanding of Scrum (67%), planning meetings (80%), tracking project progress and retrospective meetings (60%).

On the other hand, there was a passive feedback on traceability of the user stories with Virtual Scrum, students preferred the normal tools over the 3D representation dealing with configuration management. Negative comments also received on user interaction, authors had to improve it i.e. integration with social networks.

Other games adopted in some other papers, some are based on those from the website in [20], as an author of [8] have used. This author teaches students the core principles of Scrum using a wide range of Agile games. Students learn about the Scrum roles; sprints and their planning, reviews, and retrospectives; product backlog, user stories and their prioritization. His experiment was a part of a shared work between authors from four universities and colleges in Canada. They aimed to come on the experiences and challenges of practicing Scrum and Agile methods at a set of computer science programs. Another author gave his students a scrum-like course in students' second year, in order to anticipate the third-year software engineering course that includes an ASD project, he has adopted on a web tool (CATME) [21], for formation and evaluations of the team members. Third author teaches two courses, the first cover the basics of software engineering and the requisite tools and techniques involved in developing group projects. Then they have a second course where students apply the Scrum methodology that they have learned on a real project. The last author does not follow either Scrum or XP definitely, they developed an agile hybrid process to teaching the software engineering course.

Another game site is PlanningPoker, where students play to learn the user stories point's estimations in some works [2].

2.2.6 Agile Practices

In [10], a systematic literature review of papers on Agile requirements engineering, written based on experience or empirical studies, and have been produced between 2002 and 2013. Authors aim to address the adopted practices of agile requirements engineering in different published empirical research. And how is agile differentiate from traditional requirement engineering. Also to stand on challenges of agile.

They identified 17 practices: Face-to-face communication, Customer involvement, User stories, Iterative requirements, Requirements prioritization, Change management, Cross-functional teams, Prototyping, Testing before coding, Requirements modelling, Requirements management, Review meetings and acceptance tests, Code refactoring, Shared conceptualizations, Pairing for requirements analysis, Retrospectives and Continuous planning.

Their findings point there is a need for a more attention and extra empirical results in the field.

2.2.7 Contribution

To our best knowledge this study is considered the first study evaluating the effectiveness of adopting the Agile Software Development Methodology in teaching in Palestine, and the second in the Arab world. This work has what sets it apart from other research, it shows a complete adopting of agile in teaching the SWE course for the undergraduate students, not just in lectures, or during a week lab, or through a three-hour game; instead it goes along a full semester, then it depends on lectures, workshops, trainings, discussions, some entertainments and much practicing. The students had to find a real project to work on with a real customer, which is much closer to their future work. Furthermore, we arranged with a senior developer from the real world to help keep up with the projects, who helped us assure that all scrum aspects are adhered correctly.

Another important point that differentiate our work from a lot of others, that we have a big focus on agile values and practices, not just scrum process and artifacts.

We have almost a complete simulation of reality, while many other researchers disregard the necessary tools needed for the experimental groups who will apply agile in their projects, which are an important factor that helps the experiment succeed as they lately concluded, in our case we pay a big attention to choose the best tools by a defined criteria, which contributed a great help, this criteria based on many factors, like the cost (free time and users given), simplicity, and matching the agile needs, and as a whole we intend to find a compatible and interrelated tools to use from the same screen.

Chapter 3 Research Methodology

In this research, we focus on undergraduate-level teaching, the experiment was held over the second semester of year 2018/2019 at Palestine Technical University (PTUK). It was executed through the SWE course which is a required for students in their third year, in which other SE methodologies are taught along with ASD in the course, and other Agile methods are introduced, not exclusively Scrum.

3.1 *Scoping*

3.1.1 Goal Definition

Our goal is to analyze the level of students' understanding of SWE and Agile, and analyze the final product of student projects, including the bug rate and the percentage of implemented features, for the purpose of evaluating the impact of adopting the Agile Software Development Methodology in teaching, in the context of 3rd year SWE course students formed in teams of 4-5 members working on complicated problems. The study is conducted as a blocked subject-object study, since it involves many subjects and more than one requirements document.

3.2 *Planning*

3.2.1 Context Selection

The context of the experiment is a SWE course at the university, with undergraduates 3rd year students, hence the experiment is run off-line, not a part of an industrial development project. It runs on real problems of limited-features requirement.

It is considered as general research case in the sense that it aims to compare two software development methodologies in general, and from a research perspective. It is important to state that the support for the two inspected methodologies is comparable, and the subjects in both tracks have no prior experience with any.

3.2.2 Variables selection

The first independent variable is the software development methodology and it has two levels: Agile and Waterfall. The dependent variables are quality and completeness of the developed software.

The second independent variable is the Pedagogical model in teaching the SWE course, particularly the Agile part of it. It has two levels: the traditional way and our new model of including the agile project. The dependent variable is the level of students in understanding and practicing Agile.

3.2.3 Selection of subjects

The subjects are chosen using opportunity sampling which is a non-probability sampling, there were no other criteria to the sampling method except that people were available and willing to participate. The subjects are students taking the SWE course for that semester when we were doing the experiment. It is significant that the students still have the freedom to participate or deny participation in the experiment without any penalty. The size of the sample is: 8 groups with a total of 38 student.

3.2.4 Hypotheses formulation

As expressed in the goal definition of the research we would like to compare between the agile and waterfall models in SWE, to find its effectiveness in a simulation of real world projects, so we can dependably support our approach of replacing the traditional teaching of SWE course in universities in a manner which give more attention and focus on agile, and then do the needed shift in the pedagogical model of teaching itself, to be project based in the first place, and design whole the course to achieve a better understanding of SWE and let students be able to practice it before going to the market. Which is the main objective of this research.

The comparison will be based upon two components: quality and completeness of the product when using two different software development methodologies, Agile and Waterfall.

The quality here is the functional quality defined as the degree to which the correct software was produced, and can be evaluated by the bugs rate. Where completeness measures the percentage of developed features from the whole required features in the product backlog. And then completeness will indicate the time-to-market.

a- So the first Factor is *Software Development Methodology*

If we let:

µC Waterfall and µC Agile be the number of features completed of the project requirements applying Waterfall and Agile respectively, and

µF Waterfall and µF Agile be the number of faults logged applying Waterfall and Agile respectively,

Note: Completeness indicates Time needed.

Then, the *hypotheses* are formulated as follows:

Null hypothesis, H0: Agile needs the same time-to-market that waterfall needs. H0 : $\mu C_{\text{Waterfall}} = \mu C_{\text{Agile}}$

Alternative hypothesis, H1: Agile optimizes the time-to-market comparing to waterfall. H1 : $\mu C_{\text{Waterfall}} < \mu C_{\text{Agile}}$

Null hypothesis, H0: Agile methodology produces the same number of bugs as waterfall methodology. H0 : $\mu F_{\text{Waterfall}} = \mu F_{\text{Agile}}$

Alternative hypothesis, H1: Agile methodology produces less number of bugs than waterfall methodology. H1 : $\mu F_{\text{Waterfall}} > \mu F_{\text{Agile}}$

b- And the second Factor is *Pedagogical model in teaching*

The adopting of agile in teaching required a shift to a new model of teaching itself, where the entire outline needs updates, the topics and the lectures style. We want to make the student live a software engineering environment, where there is a real project needs management and development, in a way that enables him to understand the subject of software engineering and apply it.

In our research we will apply the new approach of teaching which is (Agile Practicing model) on agile projects with the experimental groups, where the control groups will assigned the same corresponding projects but still receive traditional education that already applied in the university.

Where R: represents the Rank (Level of students in understanding and practicing Agile). The rest of hypotheses are formulated as follows:

Null hypothesis, H0: Students from the new approach of teaching have the same understanding of software engineering and agile as the students from the traditional education.

$H0 : \mu R_{\text{Old}} = \mu R_{\text{Agile Practicing}}$

Alternative hypothesis, H1: Students from the new approach of teaching have better understanding of software engineering agile as the students from the traditional education.

$H1 : \mu R_{\text{Old}} < \mu R_{\text{Agile Practicing}}$

Measures needed: *Faults/KLOC*, the number of faults divided by the number of lines of code. *Completeness*, the number of features completed divided by the number of features required. *Students Understanding*: a pre and post quiz, and a questionnaire at the end of the semester.

3.2.5 Experiment design

When it comes to randomization, the decision of the subjects division on teams will be randomly selected of the available students. And so the assignment to each treatment (Agile or Waterfall methodology) is selected randomly. And if there is any group out of the 3rd year class would engage in the experiment we do blocking on prior experience, by a pre-test, attempting to guarantee that it does not influence the result of the research, blocking also done for differences between problems they worked on, in terms of complexity and context. Moreover the experiment uses a balanced design, which means that we have the same number of subjects per treatment in the design. (We have eight groups half of them apply Agile each at different application. The other four groups apply the Waterfall in those applications development, the applications types are Desktop or Web).

The experiment includes two factors of primary interest, the first factor is (software development methodology) with two treatments (Agile and Waterfall), with four tests with different four subjects for each treatment, each corresponding pair of the two treatments work on symmetry object. Where subjects are the different participated teams or groups, and the objects are the real problems/applications. Whereas the symmetry here means that the products are of the same context and the same level of complexity.

It also includes a second factor (pedagogical model in teaching) with two treatments (Old model and Agile Practicing model). Where subjects are the course groups, and the object is the (SWE course / Agile part of SWE course).

3.2.6 Instrumentation

The instruments for performing and monitoring the experiment are of three types, objects which are here the product and its requirements, the second instruments are guidelines and principles for the two development methodologies. And because it is important to guarantee a rightful comparison, as discussed before, we concerned comparable support of available resources and training for the two methods for the teams. Lastly, measurements instrumentation conducted via data collection in manual and online forms and discussions with students and some read from tools used.

3.3 *Operation*

3.3.1 Preparation

Here we select and inform participants, and prepare material such as forms and tools. In the first class of the SWE course we present a short session to students to describe the nature of the experiment and its aim, also how the results of the research will be used and published. And explain the global spread of Agile usage, then how they will benefit from this experience on their personal skill and in their future work even in local companies or multi-country teams. It will be made clear to the participants they are free to withdraw from the experiment, then we obtain their consent and willingness to participate.

All experiment instruments is prepared in advance, including the experiment objects, guidelines for the experiment and measurement forms and tools.

Then we ensure that the infrastructure needed is in place. This includes having a suitable room booked for addition lectures or discussion with university agreement, and a lab for practical workshops and trainings booked a day before it is needed, also a certain lab is used an hour daily to give an area for teams' daily scrum.

3.3.1.1 Which ASD method?

There is a revolutionary transmit from prescriptive approaches (waterfall, Lean, Kanban) to empirical processes (Scrum). Prescriptive approaches rely on planning and controlling the plan to success, Managers manage resources to do the work, and developers do the work, Command and control utilizes the productivity in predictive procedures. But Empirical processes regularly direct the results to the maximum as possible, Manager's work is to posture the big problems, and to help who are solving the problems as possible as they can, Creativity and collaboration are the stamp of this method.

We are looking between the Agile Software Development methods for an easy method to comprehend, which makes it ideal to introduce agility to undergraduate students. Also we focus on the management aspects of projects. And which has fast iterations and active collaboration within the team. So we chose Scrum for its ability to incorporate various overarching practices promoted by other Agile models. And due to its proven productivity, and its popularity in Palestine market. And some other details and comparisons discussed below.

Lean processes are good for complex systems and considered the most cost-effective. But it spends much time on fixing complexities than constructing new software products. Unlike Lean, Kanban do not mind much in mitigating wastes, but in utilizing the manufacturing process. Kanban is frequently used when an organization can not readily adopt Scrum [22] [23]. Agile methodologies are a well-known approach to flexibly overcome the requirements unexpected changes [12]. Despite of this, we are not exaggerated in praise of Agile. There is no method is a "silver bullet" each has its strengths and weakness as well, savvy is to know where to use each.

Teaching Agile methodologies oftentimes converges on a particular method, such XP or Scrum. It is troublesome to get into variety of Agile methods and practices in depth with fitting into the short-lived semester [5].

Our course is based on Scrum, where products are evolved incrementally in sprints enclosed to a time frame, where the team works on the sprint backlog of a requirements set and generates a running piece of the required software. It is significant that through a sprint its backlog does not change; any changes or new requirements or other not finished user stories are gathered and kept to separate backlog scheduled into next sprints. So without reminding its spread in industry, Scrum is the easy to comprehend, accessible and effective methodology, which makes it ideal to introduce agility to undergraduate students [2].

Scrum and XP are the most applicable agile methods in many studies, and they have common structures, roles, and values. There are however some fine differences, for example where XP pays more attention engineering practices such as pair programming and Test-Driven Development (TDD) [5]. Which does not match our purpose of improving students' projects management skills.

It is worthy to highlight some orientations that weaving two or more ASD approaches together is acceptable and productive [6]. To facilitate doing this, teacher should not distinct strictly between each approach practices. Also it is helpful to merge different methods terminologies [1]. In [24] authors have practiced a hybrid model, merging XP and Scrum in a single course. They argue that those two methods work well when correlated. Simply because they address different areas and supplements one the other i.e. Scrum focuses on management practices whereas XP focuses foremost on engineering practices.

3.3.1.2 Tools

Several specific software development tools have been integrated in the new pedagogical model of teaching, to be used for the experimental groups, who will applying agile in their projects, as the teacher guides them. We searched for appropriate tools needed in experiment implementation and tried them out, and examined them carefully to specify the most suitable ones. Some of them mentioned in related research. Others we have discovered online mainly to find a compatible and interrelated tools for students to use from the same screen. And in second place we focus on tools that students may understand faster and like more. A third factor that we were not able to neglect is minimizing the cost, we chose tools that allow the most possible time of free using and allow more free users to join. Lastly, we search for tools match the agile needs, so for example, we chose “Trello” over “MS Project” for management, because Trello designed especially for Scrum. Another example on this factor, we chose Git over google shared documents for code collaboration and management.

As an example, for Agile Project Management, there are many tools available for developers to use, like: Active Collab, Agile for Scrum, Jira, MS Project, Trello and others. From those all, we have chosen “Trello” according to the previous factors, it is also used in many companies and has a pretty usable interface. More about Trello is below.

Project Management Tool – Trello. A main idea in Scrum is the transparency, where the team should know the progress, and what they try to accomplish. That is also why we used Trello as a Scrum Board. This is a platform where team members organize the backlog, and tasks of the current sprint and their progress. It is compatible and powered up by many other applications in many directions; In Analytics and Reporting we find in the list for example: TimeCamp and Burndown for Trello. In Communication and Collaboration we find: Dropbox, Google Drive, Hangouts, Slack, Twitter and Calendar. In Developer Tools: Bitbucket Cloud, GitHub and Planning Poker. In Design: SmartDraw.

We mainly use from those with Trello: the Bitbucket for source code management, and Trello itself for communication related to any card/task or general conversation, and surely could insert any link from a file management repositories.

For source code management, collaborate on code, test and deploy. Students use “Bitbucket” which is a Git code management. And can be interrelated to the Trello screen.

For team work assessment. There are ClassDojo, Dapulse/Monday, Catme and others, we use “Catme” for team formation and team member evaluations by each other.

For online meetings with lecturer. We use “Webinar” which is a web-based seminar used for off-class lectures or trainings.

For online communication between team members. There is an embedded efficient platform in the “Trello” itself.

For filling questionnaires /forms and quizzes. It is executed on the University Learning Management System “Moodle” and online forms.

Course Outline. The course is designed to fulfil the Pyramid of Agile Competences described in [6]. From top: Agile Values, Management Practices, and Engineering Practices. Which the first two can be taught respectively in lectures and team project but the engineering practices (like adhering the code of ethics) is hard to achieve because teacher have to change individual attitudes.

The course runs along a semester of 15 weeks; the project is implemented during six 2-weeks sprints. The course general outline in the appendix A, the lectures differ between discussion lectures, simple gaming, and technical workshops.

The course outline was designed in the light of the plan of SWE course in the CSE department in the university, and in light of previous research, those which prepared plans for teaching agile, then we did some special modifications to fit the new experiment. And ILOs as well, they also fulfill the five levels of Bloom's taxonomy of educational objectives.

By the end of the course, the student will be able to:

1- Define the software engineering and list some software development methodologies.
2- Explain main concepts of SWE.
3- Characterize main methods for software development.
4- Sort waterfall model phases.
5- Differentiate between functional and non-functional requirements.
6- Construct low and high fidelity prototypes.
7- Comprehend and draw different UML diagrams.
8- Identify and explain Agile Manifesto of values and principles.
9- Choose the appropriate IDE to install for applications development.
10- Create a clean design for their projects and standardized code.
11- Organize their teams.
12- Collaborate with other team members and the customer.
13- Analyze the customer needs.
14- Apply ethics when dealing with others, or working their tasks.
15- Recognize and use the new tools efficiently.
16- Demonstrate good comprehension of scrum concepts.
17- Perform agile and scrum practices correctly.
18- Design and organize the documents needed.
19- Manage their projects development process.
20- Differentiate between the roles of scrum.
21- Locate the participants of each scrum activity.
22- And will be able to assess their work and test the code.

3.3.2 Execution

The research includes the Computer Systems Engineering major students; it is almost cover quite large population which we can get reliable results from. As a total we have eight teams working on eight projects running in parallel, four of them are scrum teams which are of four to six members, as most papers recommended the agile teams should be small.

The execution is through one semester fifteen-week course, which typically has two 75-min classes per week, required for 3rd year students, their age is relatively a positive factor, such it is easier to achieve best results since the social side of ASD fit with this age as mentioned in [1]. Another clear important cause of choosing this stage is that they have already passed the prerequisite courses including Java, Data Structures, Computer Networks, and Databases, so there is no need to lose some classes to teach these topics to the student to be able to achieve the product implementation depending on that skills they already had.

Students have distributed in groups from the early lectures, each of those groups have to find a real customer for a software in whatever domain; to ensure that they are taking a fully real requirement, also their motivation is enhanced by working on professional and real-world development modality.

Inserting some form of competition between the teams can be auxiliary to maintain the pace up if this have not been placed excessive emphasis on it [1]. So we intended to encourage each group to have a name represents their team as a company, each company should have a logo and may have a Facebook page.

The applications vary between Desktop and Web Applications. At the end of the course they should have the first version of their project ready and tested.

Through the course all the concepts of SWE were covered, and many SW development strategies mentioned. And some of the groups will construct their product following the waterfall model. Others will be guided to go through the Agile strategy, where the instructor follows up the whole processes and activities through the class and off hours.

In [2] authors conclude that proper tutoring and coaching of teams with respect to agile methods is a key factor for a project's success; In [13] a finding that denotes that students are often lacking critical skills and knowledge at first. So in our case tutoring have been achieved through classes -particularly the first classes by the teacher-, coaching will be done through the workshops by the external Scrum Coach.

3.3.2.1 Roles Assignment

The main scrum roles described in [4] will be assigned to team members, with their self-explanatory names, which is highly helpful in development life as discussed in [1]. These roles are: The Product Owner having the user stories and leads teams to transpire the requirement specifications, he is the person who will contact with the customer. The Scrum Master who is leading the team into enforcing the Scrum practices (Daily Meeting, Retrospective Meeting, Planning Session, etc.), helps make decisions or obtain needed resources, and guarantee Scrum process is followed as it should. The Scrum Team is responsible for developing and testing the requirements [3].

In our experiment, Scrum roles have been assigned as follows:

- Scrum Master initially assigned to the teacher until the students are ready after some sprints to take the responsibility, then it will be given to the student with the best familiarity of technology and tools utilized in a certain iteration [1], or he could be voted within the students as authors done and recommended in [6] to let the teams more self-organized.
- Scrum Coach was external to the college; he is a senior from a software company in Palestine (named: Dimensions).
- Product owner assigned to students under supervision of the teacher.
- Other students will take on the developer role,
- And some of them (one or two) be testers.

Teacher mission is tutoring and he must be existing pending teams' meetings, so that he might hint best practices on both high and low level of software design, to help students in the use of notations, and more. However, teacher existence must be prudent since members are responsible of the project at end but they are just assistants and observers [14]. Again teacher's role should be a full expressing of the "guide on the side, not a sage on the stage" principle [1].

Scrum coach was responsible for follow up with teams progress, and a consultant for any scrum issues, he was the trainer on source code management tools, also he evaluate the final projects if adhered code standards and participate in testing applications.

3.3.2.2 Self-organizing teams

Within the team we did not impose the structure, members themselves are allowed to self-organize, much like a real Agile team would be expected to do [5]. Self-organizing teams are at the core of “the Agile way”. It supposes the students’ self-control and self-management. It violates with traditional class management so it may be difficult to conduct at first, but a better interaction and partnership will be a reward [1].

3.3.2.3 Data Collection

There will be a formative and summative evaluation to assess the students understanding and skills which are built during practicing Agile over a comprehensive understanding of peculiarities of that methodology.

According to our hypotheses and measures needed, we have collected data in different ways. Data about learning outcomes to test the hypothesis mainly gathered by a quiz which is a summative evaluation method, consists of different question in SWE, Agile and Scrum. It was executed on the University Learning Management System (Moodle). During the semester we monitor how students are practicing Agile, by observations and evaluations, writing down points for each member and team according to the tables in Appendix [B, C and D] mainly to follow up with teams work and make sure they adhere each principle well. Also discussions were needed in many stages with students about the work and how to improve it. So the researcher has the possibility to communicate better with the participants through the process.

A questionnaire at the end of the semester done to investigate students satisfaction through the experiment, inner questions to be rate on a 5-points Likert scale are clear in Appendix [H]. Lastly we test the final products to find the number of faults, and the number of features completed of the whole number of features required.

3.3.2.4 Experimental Environment

Surely the experiment should not change natural environment too much thus affect its objects more than needful. However there are conditions where some interaction of the experimenter is convenient to have better execution or estimations in this research. An example of this potential interaction is the intervention by adding some changes to the requirements during the teams are working on the applications development, in order to study some related affairs in dealing with upcoming changes.

3.3.3 Data Validation

Finally, the data had been validated before it was analyzed on SPSS, if it is reasonable, and has been collected in a correct way, and that the subjects have applied the methodology with its provided practices and principles through the team monitoring tools discussed previously (Trello, Bitbucket and Catme). Or some data may be taken away. An example of a monitored practice is the daily scrum, by daily writing down the attendance and absence of students with the help of the lab supervisor (whom we already had his agreement to help in such cases). Other principles and values discussed in Appendix [B, C and D], mainly designed to follow up with teams work and make sure they adhere each principle well.

Chapter 4 Results and Discussion

We have conducted our experiment on the SWE course students in the second semester of year 2018/2019. It was executed on eight teams of four to five members each. Four of them applied Scrum, and the others Waterfall. Three pairs of teams were developing desktop apps and one pair was developing a web app, and all of those teams had no prior experience. The total number of the students were thirty eight students, with eighteen in Agile teams.

Below in Table 4-1 is a summary of teams and projects.

type\method	Agile	Waterfall	Project Context
Desktop	First team "Hash the Dash"	Fifth Team	Dental Clinic Systems
	Second team “Alpha Team”	Sixth Team	Medical Laboratory Systems
	Third Team “Rainbow Team”	Seventh Team	Cars Insurance Systems
Web	Fourth Team "*CCCare Team*"	Eighth Team	Child Care Center Systems

Table 4-1. Summary of teams and projects

It is important to remember that the projects they were developing were of the same context and the same level of complexity, but different products and details. We have used “Trello” for Scrum teams as a project management tool and the “BitBucket” as a source management tool. Screen shots of “Trello” teams boards found at Appendix F. And screen shots of “BitBucket” of first team found at Appendix G.

4.1 *Evaluation of Learning Outcomes*

For the factor: Pedagogical model in teaching, the hypotheses were formulated as follows:

Null hypothesis, H0: Students from the new approach of teaching have the same understanding of software engineering and agile as the students from the traditional education. $H0 : \mu R_{\text{Old}} = \mu R_{\text{Agile Practicing}}$

Alternative hypothesis, H1: Students from the new approach of teaching have better understanding of software engineering agile as the students from the traditional education. $H1 : \mu R_{\text{Old}} < \mu R_{\text{Agile Practicing}}$

Where, R: Rank (Level of students in understanding Agile)

The data needed to test this hypothesis was mainly gathered by a pre and post quiz consists of different question in SWE, Agile and Scrum. They were distributed into two parts with a total of 20 multiple-choices questions, first part is the general part of 10 questions, and the second is the scrum part. There marks were written down on an excel worksheet, and the difference between the pre and post results were taken to find the improvements of the students, which were about 44% for the Agile students against what about 28% for the traditional ones.

These improvements were analyzed with SPSS software. By Shapiro–Wilk test of normality, shown in table 4-2, we find that the p value equal to 0.152 which is greater than α=0.05, then the null hypothesis that the data came from a normally distributed population can't be rejected, and the data is asymptotically normal. So we were able to use the T-Test which is a parametric test.

Tests of Normality

	Kolmogorov-Smirnov[a]			Shapiro-Wilk		
	Statistic	Df	Sig.	Statistic	df	Sig.
improvement	.151	38	.028	.957	38	.152

a. Lilliefors Significance Correction

Table 4-2. Shapiro–Wilk test

According to Levene's Test for Equality of Variances, of T-Test results shown in table 4-4, we find that there is a homogeneity of variance between the two samples since sig. = 0.130. Then we read our values from the first row.

Since p-value = 0.001 which is less than α=0.05, we reject the null hypothesis (states: Students who applied the new model have the same understanding of it as the students who applied the traditional model), which means at level of significant α=0.05 the data give us a sufficient evidence to conclude that there is a significant difference between the means of the experimental and control groups, which are respectively 8.94 and 5.50, as shown in table 4-3.

Group Statistics

	method	N	Mean	Std. Deviation	Std. Error Mean
improvement	Agile	18	8.94	2.461	.580
	Waterfall	20	5.50	3.502	.783

Table 4-3. Group Statistics, T-Test

Independent Samples Test

		Levene's Test for Equality of Variances		t-test for Equality of Means						
									95% Confidence Interval of the Difference	
		F	Sig.	T	df	Sig. (2-tailed)	Mean Difference	Std. Error Difference	Lower	Upper
improvement	Equal variances assumed	2.404	.130	3.471	36	.001	3.444	.992	1.432	5.457
	Equal variances not assumed			3.535	34.097	.001	3.444	.974	1.464	5.425

Table 4-4. Independent Samples Test, T-Test

At the end of semester the doctor of the SWE course have discussed each team in their projects, after they have presented it. The new model teams have explained their projects and their work in a scientific way, while they have showed a better understanding of the SWE concepts in general, as well as Scrum, in their discussions. In spite of the control groups who were not participants at the new pedagogical model were focusing (in other words, they cut corners) to finish their products with obvious negligence of methodology principles. That is strongly support our results in this most important part of the research.

4.2 *Time to Market*

The workload required during the course or the real life, makes time a worthy resource, people have to handle minutely.

For the factor: Software Development Methodology, the first set of hypotheses were formulated as follows:

Null hypothesis, H0: Agile needs the same time-to-market that waterfall needs. $H0 : \mu C_{\text{Waterfall}} = \mu C_{\text{Agile}}$

Alternative hypothesis, H1: Agile optimizes the time-to-market comparing to waterfall. H1 : $\mu C_{\text{Waterfall}} < \mu C_{\text{Agile}}$

Completeness of the projects used to be an indicator of the time needed, in term of the number of features completed through the experiment time, divided by the number of features required from the customers. The results are stated in table 5-5.

Project Context	Team	No. feat/ req. feat	No. feat/ req. feat	Team
Dental Clinic Systems	A1	26/28	24/28	W1
Medical Laboratory Systems	A2	36/37	30/34	W2
Cars Insurance Systems	A3	31/36	27/36	W3
Child Care Center Systems	A4	29/29	26/29	W4

Table 4-5. Projects Completeness

The feature is formed in a set of user stories, which in its turn broken down into many tasks. We consider some feature as completed, if its main user stories are done, done means coded, reviewed by the product owner and integrated to the system. The data in Figure 4-1, show the results for both methodologies in our experiment, and it gave us a string indication that Agile optimizes the time-to-market comparing to waterfall.

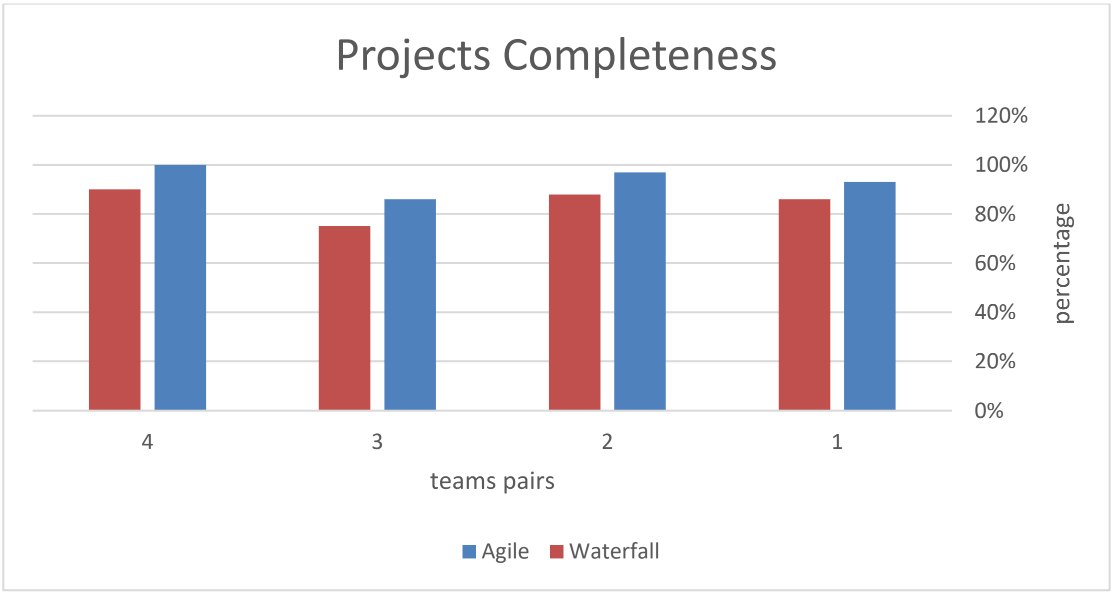


Figure 4-1. Projects Completeness Percentages

Students have tried their best to get the work finished as well as they can, and within the semester time, so they need an efficient time management, which is ordinarily easier in agile, anyone can easily find the time rhythm in sprints design, which help them be more organized.

With Scrum teams, the process of evaluation were easier than Waterfalls, because their own definition of done is clear through their design Trello lists (ex.: To Do, Doing, Testing, Done …), and backlogs content, and tasks cards.

Where with waterfall teams, they took a long time in requirement elicitation and SRS preparation, then they had dived in coding, and put off testing, where their colleagues in Scrum teams building their low and high fidelity prototypes, involving their customer, prioritizing their requirements and refine the next sprint planning.

In the execution as mentioned before we have inserting some form of competition between the agile teams, hence they have such a company and name. But when we have the results of students quizzes and projects evaluation, our vision was to not make it completely a competition, it is not the major objective from the experiment to get the best project done, but to enforce and work out ASD principles. Also if the teacher finds and honors the best team, we will motivate this team but on the other hand make other teams depressed.

The agile teams were almost self-organized as mentioned before, but about roles rotating, we prefer to not repeatedly rotate the team roles as some papers vision was, because it will lose their time and dissipate the students' efforts, and concentration, than making the development process dynamic or deepening the experience for everybody having each role himself. Instead, to save time- we have rotated the role of Scrum Master a few times only. Another reason is there will be a natural level differences between students in each group, may be one of them is not well with development, so he might choose the tester position represented the quality assurance part of the project, this does not prevent him from helping in estimation or take a design task, but not necessarily be a developer, an explanation of this decision that in the software market you will do one role in this system, not a role each morning. Another reason is what [14] have glanced at -if a novice discovers himself engaged in a working team together with witty students, he might respond by retiring out of the swarm, resigning work to others in the group, and as a result he will not be involved in any real or effective practical task. This way of distributing student within a team, taking into consideration the individuals' abilities and tendencies, and lets all members be conscious of all Agile practices and –even by observation- of all group activities.

4.3 *Product Quality*

For the factor: Software Development Methodology, the second set of hypotheses were formulated as follows:

Null hypothesis, H0: Agile methodology produces the same number of bugs as waterfall methodology. H0: $\mu F_{\text{Waterfall}} = \mu F_{\text{Agile}}$

Alternative hypothesis, H1: Agile methodology produces less number of bugs than waterfall methodology. H1: $\mu F_{\text{Waterfall}} > \mu F_{\text{Agile}}$

The product quality is indicated by the number of bugs divided by the number of lines of code – physical lines of code. The bugs included the interface and usability problems, because we gave a good attention to that non-functional requirement through the course and development.

Team	Bugs/ KLOC	Bugs/KLOC		Bugs/ KLOC	Team
A1	95/4.2	22.6	24.6	105/4.3	W1
A2	120/5.7	21	18.5	92/5	W2
A3	84/7.7	11	34.8	223/6.4	W3
A4	70/4.7	14.9	27.3	120/4.4	W4

Table 4-6. Products Quality

Figure 4-2, shows how many bugs were found during testing. It is obvious that the rates in general is lower for agile teams, so we can have a strong indication that Agile methodology produces less number of bugs than waterfall methodology.

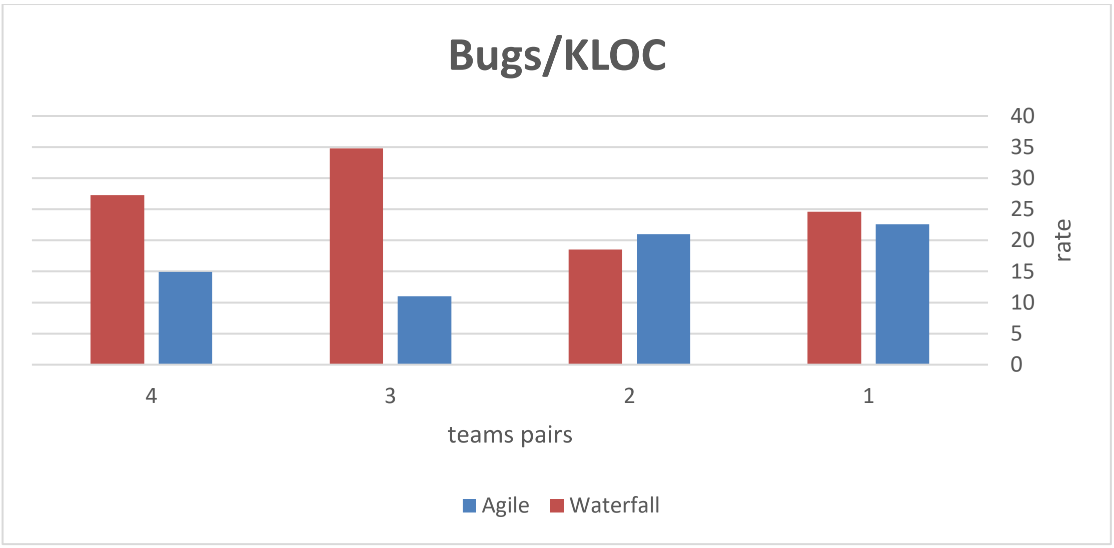


Figure 4-2. Bugs/KLOC rates

In table 4-6 if we compare teams in each pair together, we find less bugs per KLOC at the left part of the table, except the results for pair 2, perhaps we can attribute this to the team's less code, and features developed, compared to the corresponding agile team, and after we had discussed that with the team, they said that they were sometimes meet the customer if they feel a necessity and that may help them achieve this score.

Experimental groups did not see testing as last activity of the development. Frequent testing and frequent releases are very helpful in quality improvement.

If we compare bugs per KLOC within the experimental groups themselves, we find the best result is for the third team even they have the biggest code size, also it is one of the highest number of features between the projects, as Table 5-5 shows. Furthermore, their corresponding team have the highest number of faults. All of this is justified due to the type of the application, it is primarily an accounting system, where examiner or user can find a lot of errors if the product had not been tested enough.

4.4 *Student Satisfaction*

In the last week of the experiment, the eighteen student in the experimental groups filled in a satisfaction questionnaire, online using a free tool for questionnaire creation and analysis. The questionnaire consists of fifteen statements, students asked to rate them between “Very dissatisfied: 1” and “Very satisfied: 5” on Likert scale to gather their feedback. Overall feedback was positive. The second part asks if they will recommend their colleagues to take this course. All of the eighteen students have answered by “Yes”.

Figure 4-1, shows the Top/Bottom five results of the fifteen statements, with its percentage of satisfaction.

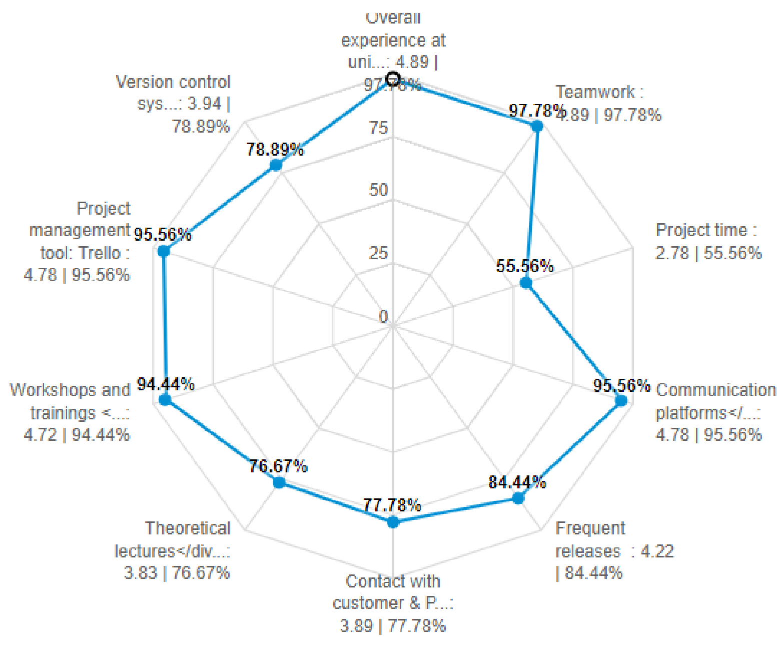


Figure 4-3. Top/Bottom five results of the fifteen statements, with its percentage of satisfaction

These results points a sense that the design and content of the course are mature enough, hence students are satisfied in general, with overall satisfaction of 87.6%. However some negative feedback arrived. These results were discussed with the students after they had all submitted the questionnaire and we had had the results analyzed. The negative feedback was especially on time offered to the experiment compared with the amount of work required throughout the project and with many new concepts introduced, learning about the source code management.

We -as authors and teachers- satisfied of the experiment significant results, and the good influence on the students, and satisfied of the high quality products delivered by the teams too. It is also worth mentioning that one of our projects become a graduation project for students.

The results in details are in this link [38]. Include a full graph of the figure 4-3. Followed by a table shows the statements of the first part, and its scores of satisfaction, and ends with an analysis for satisfaction on each statement with the number of students answered each choice. Some of them shown in Appendix H.

4.5 *Other Results*

We had conducted a trial similar partial execution of this experiment a year ago; to learn from that experience in order to help us in conducting the current main experiment.

It was executed on five teams of four to five members each. Three of them applied Scrum, others Waterfall. We had 4 teams working corresponding each other as the research balanced design states, so we had 2 web apps and 2 desktop apps in total, all of those teams had no prior experience.

But the fifth team had developed a mobile app, and they have used “Jira” and “GitHub” instead of “Trello” and “BitBucket” respectively. Also, they are differ from others, that they have passed the SWE course a year before, so they had some experience. Hence, before we engaged the fifth group in the experiment we had dine blocking on prior experience, by a pre-test, attempting to guarantee that it does not influence the research results. Table 4-7, shows a summary of teams and projects.

type\method	Agile	Waterfall	Project Context
Desktop	First team "*Breaking Point*"	Third team	Storage Management System “Cappuccino System”
Web	Second team "*Sky Geeks*"	Fourth team	Tourism and Travel Systems
Mobile (Android)	Fifth team “*Jira Team*”		

Table 4-7. Summary of trial experiment

The results were mostly in line with the current results, in addition to some dedicated observations related to its different teams, students who applied the agile process without prior experience in the waterfall process (first and second teams) perform the same as those with prior experience in the waterfall process (fifth team).

Students were able to go through the agility from the team's first project conforming full software development lifecycle. Scrum such as chess; one can pick up the principles in minutes, but it needs more time to be great at it. Theoretically, the team should be more efficient and produce more work with each new cycle [37]. And we found this somehow true because one becomes more familiar with tools and practices.

In addition, we found that the probability of finding students able to develop a mobile app –in order to maintain apps diversity- is very low. The experiment results are going well with whatever were the apps types.

The following are some noticeable results related to unlike tools used. Two different tools were used for source code management which were “BitBucket” -as we planned- and “GitHub”, they are almost the same, both are Git code management, and can be integrated in Trello. In both as we wrote before, students need intensive training on the source code management tool because it is a new concept and tool for them, which they did not use before. But any way they are both better than SVN.

Another difference, that we used “Trello” except the fifth team have used “Jira”. Trello have simpler interface, and students was satisfied with Trello more than Jira. Furthermore, free Jira gives a limited time (a month) and number of collaborators (three members), or you have to pay to get more, so we have paid for the fifth team. As a conclusion we again recommend Trello as a project management tool.

Furthermore, one of those groups has used agile methodology in the development of their new web application project as a graduation project.

4.6 ***Di*scussion of Results**

For all the above results, we found a clear agreement with the literature results and outcomes, and on the other hand, we take their recommendation -extracted from the problems they were faced- to upgrade our experiment design which improved our results.

We all agree that teaching agile is a necessary, according to its widespread and success in software development, and the universities is the best environment to do. But we may agree or not agree in design details and results.
We share that it is very necessary to balance the time granted for software functionality with the time required to work on agile as a way to develop the software projects, and this was achieved by assigning students to projects that are not complicated. Also, the simplicity of the projects maintained their appeal to students to complete the experiment. Furthermore, we found that there is no problem in helping students in some tasks like the estimation part of the project to keep up with progress.
Permanent follow-up with students, attending their meetings and guiding them had a clear effect, and the frequently workshops and discussions as well grant a great help. In exchange for the lecturer role of giving assessments only at the end. But we should keep in mind that more monitoring gets students unsatisfied.
Likewise, the usage of helping tools where necessary, give the new model students a push in understanding each aspect of the work and its needs. And naming the teams made them motivated to learn more correctly, so that they could properly implement their project, and then compete with the rest of the teams. Teams in its turn should be small, to benefit each member, and we recommend our strategy in roles assignment, which comes in light of all others advices and our own opinion, it was really fit the aim with no side effects.
Documents are important in any project, to go through logical and correct steps based on a clear objectives and charts, but as agile manifesto recommend, we also from our experiment recommend to focus on working software more than comprehensive documentation, because any expansion will become useless, delay the project and overburden the student.
The course design and material should be fully prepared before start the experiment, as well as the infrastructure needed by the students, and other instrumentations.

But from our experience we do not agree with authors conducted the experiment in time frame shorter than two to three months, which does not achieve the desired results. About the course, it should consist of just one assignment consists of all software engineering aspects, where the traditional model and some of literature experiments of adopting agile were based on a set of discrete small activities. And when come to tools, we see that researcher should not neglect any necessary tools, even its hard to learn from the students, in that case teacher have to find a good alternative or train the student.

Lastly, we applied simple games and entrainment through the course, but they were not enough to write down results and conclusions after them, but we recommend to employ more games if time permits, all previous research supports this trend.

Chapter 5 Conclusion and Future Work

5.1 *Conclusion*

Agile is the most used methodology in software companies, so it is clear that Agile more important than just has been defined with some of its characteristics in the software engineering course in the universities, where it is more effective and less risky to get this knowledge and behaviours in academic environment, paying attention to the methodology of teaching itself, teaching Agile should be Agile teaching, to improve the output quality. Teacher may be act a negative role in the learning process if he is not skilled as required, so in our study we invited an external coach to monitor the process. That helped us assure that all aspects are adhered, and give more reliability and credibility to the results we gained through the experiment in answering our research questions. Evaluations of learning outcomes and students satisfaction demonstrate that the course concepts were well received, and participant students learned much about agile and software engineering, while having fun. Also, the experiment was helpful to enrich the skills of management, collaboration and communication between students. An important contributions of this paper to light a methodology for teaching the SWE as a project based course of agile development, and to present different techniques to enhance this methodology.

5.2 *Future Work*

To accommodate more students, where all groups might be given the same project, then it will be easier to manage and follow them up. Or having a large project as well, then distribute its aspects between groups, this case will be closer to the reality. Like in process happening in outsourcing development with distributed teams and remote customers.

Next times, there will be more focus on games as a factor through teaching process, and additional treatments of non-functional requirements through product development in addition to the usability requirement. Hence, be able to analyse its appliance results.

In the coming years we intend to develop a simple course as an introduction to Agile, dedicated to second year students, so in the third year it will be easier to exploit the semester time in a better manner. Also, it would be possible to give the students a complicated project from external client at their fourth year in university.

5.3 *Recommendation*

We recommend that other courses like: Object Oriented Programing, Data Structure, Database and all other courses including a project, to adopt Agile methodology in their group projects. Not to just isolate SWE teaching in a separate course.

As a reaction to the increasing needs for SWE professionals who are understanding and introducing Agile in their work, and on consequent of this work [5], we suggest to have an experiment on SWE master students in the Construction Course, or introducing it as elective course, where the students are usually full-time employees in public or private IT sector.

Through data gathering we recommend to have a questionnaire measures customers satisfaction, which gives an indication of the validity of students work.

5.4 *Threats to Validity*

For the external validity, our sample is based on the students who happen to be most accessible to the researchers. The experiment happened in one university with thirty eight participants who eligible to our aims. There is no way to tell if the sample is representative of all Palestinian universities, we are not able to include many universities in the experiment and then have a random sample, so it might not produce generalizable results, due to under-representation of subgroups in the sample.

Another limitation, regarding the conclusion validity, we were limited to a certain number of subjects due to the population available, our population is the SWE course's students at a specific semester in our university. If we would intend to increase the sample, then we had two choices, the first is to repeat the experiment on other years, and this is hard to achieve according to duplicated time needed, but we had done a trial experiment before, it was somehow useful guiding us for a better execution of our official experiment.

Another choice was to include the Applied Computing students in the Applied Science College, who have the SWE course in their study plan. But we preferred not to do, to guarantee the unified level and background of groups, then the homogeneity among them. And make less effort on the authors in training the participants and arranging times according to students other lectures in a different college. But it still a possible choice for next iterations of the experiment.

We have another conclusion validity concerned with results analysis and two hypotheses tests, those hypotheses for the second factor are time-to-market and bugs rate, their resulted data are so small to perform a statistical test, so we can't eject the null hypotheses, we can just have an indications from the data.

Also, it is a human-oriented experiment, as regards internal validity, this implies a limitation to the control of the study, since students have distinct capabilities, skills and interests, which in itself may be an independent variable. But in our study were the students grouped in teams, they will be internally heterogeneous, but homogeneous among groups, which reduces this limitation effect.

For the construct validity, we should track our students after graduation; since to some extent it needs much effort to report they learned such valuable thing, without see them at work.

In the experiment design we have two factors to investigate, which may have affected each other, the first was the pedagogical model, where in the new model we adopting agile and designing the whole course, lectures, projects and its follow-up in a new manner dedicated to teaching agile professionally, which includes the teacher as a guide for the students, and we have a volunteer from industry to help students in the new approach of learning. This treatment was on teams who assigned projects to develop using Agile. While the other treatment, which is the old model, was applied on the rest of the students who are the control group and who are assigned the waterfall projects, and that is okay for the first factor.

But when we move to the second factor of the software development methodology to compare between two methods the first is the agile and the other is water fall, by the object which is the final product developed, in means of time needed and bugs rate, there is a difference between the attention given to the two teams according to which model of teaching they belongs, this difference act as a construct validity, that we can't avoid unless we separate the two factor into two experiments with different subjects.

Furthermore, we thought it might be better if we test null hypothesis of time-to-market, by straight forward finding the velocity of each team, velocity is the result of the division of Agile story points delivered by the number of sprints. But it has a poor fit for waterfall strategy of development. So, we found the number of the completed features from the whole required features in the limited time of the experiment. Which considered suitable for both methods.

References

1. Devedžić, V., & Milenković, S. R. (2011). Teaching Agile Software Development: A Case Study. IEEE Transactions on Education, 54(2), 273–278.

2. Schroeder, A., Klarl, A., Mayer, P., & Kroiss, C. (2012). Teaching agile software development through lab courses. IEEE Global Engineering Education Conference, EDUCON, 1–10.

3. Rodriguez, G., Soria, Á., & Campo, M. (2015). Virtual Scrum: A teaching aid to introduce undergraduate software engineering students to Scrum. Computer Applications in Engineering Education, 23(1), 147–156.

4. Cervone, H. F. (2011). Understanding agile project management methods using Scrum. OCLC Systems & Services: International Digital Library Perspectives, 27(1), 18–22.

5. Johnson, A. M. A. (2017). Teaching Agile Methods to Software Engineering Professionals: 10~Years, 1000 Release Plans.

6. Kropp, M., & Meier, A. (2013). Teaching agile software development at university level: Values, management, and craftsmanship. Software Engineering Education Conference, Proceedings, (November 2014), 179–188.

7. Hazzan, O., & Dubinsky, Y. (2007). Why Software Engineering Programs Should Teach Agile Software Development. Software Engineering Notes, 32(2), 1–3.

8. Campbell, J., Kurkovsky, S., Liew, C. W., & Tafliovich, A. (2016). Scrum and Agile Methods in Software Engineering Courses. Proceedings of the 47th ACM Technical Symposium on Computing Science Education - SIGCSE '16, 319–320.

9. El-Khalili, N. H. (2013). Teaching Agile Software Engineering Using Problem-Based Learning. International Journal of Information and Communication Technology Education, 9(3), 1–12.

10. Inayat, I., Salim, S. S., Marczak, S., Daneva, M., & Shamshirband, S. (2015). A systematic literature review on agile requirements engineering practices and challenges. Computers in Human Behavior, 51, 915–929.

11. Beck K. (2001). Manifesto for agile software development. [Online]. Available: http://agilemanifesto.org/

12. Abrahamsson, P., Salo, O., Ronkainen, J., & Warsta, J. (2002). Agile software development methods: Review and analysis. Espoo, Finland: Technical Research Centre of Finland, VTT Publications, 112.

13. Scharff C. (2011). Guiding global software development projects using scrum and agile with quality assurance. Software Engineering Education and Training Conference, IEEE.

14. Del Bianco V. & Sassaroli G. (2003). Agile Teaching of an Agile Software Process. Extreme Programming and Agile Processes in Software Engineering 4th International Conference, XP. Genova, Italy, 417-420.

15. Reichlmayr, T. (2003). The agile approach in an undergraduate software engineering course project. Proceedings - Frontiers in Education Conference, FIE, 3, S2C13-S2C18.

16. IEEE. (2009). New Standards Committee (NesCom) recommendations, IEEE-SA Standards Board. [Online]. Available: http://standards.ieee.org/

17. Sato D. T., Corbucci H., & Bravo M. V. (2008). Coding dojo: An environment for learning and sharing agile practices, IEEE. 459–464.

18. CodinDojo. [Online]. Available: http://www.codingdojo.com/

19. Lego4Scrum. [Online]. Available: https://www.lego4scrum.com/

20. Tastycupakes. [Online]. Available: tastycupcakes.org

21. Catme. [Online]. Available: www.catme.org

22. Fowler, M. (ThoughtWorks). (2009). Flaccid Scrum, 2. [Online]. Available: https://martinfowler.com/bliki/FlaccidScrum.html

23. Schwaber, K. (2010). Waterfall, Lean/Kanban, and Scrum, 5. [Online]. Available: https://kenschwaber.wordpress.com/2010/06/10/waterfall-leankanban-and-scrum-2/

24. Mushtaq, Z., & Qureshi, M. R. J. (2012). Novel Hybrid Model: Integrating Scrum and XP. International Journal of Information Technology and Computer Science, 4(6), 39–44.

25. Castronova, E. (2005). Synthetic worlds: The business and culture of online games. University of Chicago Press, United States.

26. Beck, K. (1999). Extreme programming explained. Reading, Mass., Addison-Wesley.

27. Schwaber, K. (1995). Scrum Development Process. OOPSLA'95 Workshop on Business Object Design and Implementation, Springer-Verlag.

28. Schwaber, K. and M. Beedle (2002). Agile Software Development with Scrum. Upper Saddle River, NJ, Prentice-Hall.

29. Cockburn, A. (2002). Agile Software Development. Boston, Addison-Wesley.

30. Palmer, S. R. and J. M. Felsing (2002). A Practical Guide to Feature-Driven Development.

31. Kruchten, P. (1996). A Rational Development Process. Crosstalk 9(7): 11-16.

32. Stapleton, J. (1997). Dynamic systems development method: The method in practice. Addison-Wesley.

33. Highsmith, J. A. (2000). Adaptive Software Development: A Collaborative Approach to Managing Complex Systems. New York, NY, Dorset House Publishing.

34. O'Reilly, T. (1999). Lessons from Open Source Software Development. Communications of the ACM Vol. 42(No. 4): 32-37.

35. Ambler, S. (2002a). Agile Modeling: Effective Practices for Extreme Programming and the Unified Process. New York, John Wiley & Sons, Inc. New York.

36. Hunt, A., Thomas, D. (2000). The Pragmatic Programmer. Addison-Wesley.

37. Trello. [Online]. Available: https://blog.trello.com/

38. Satisfaction Questionnaire – questions, results and analysis. Available: https://www.questionpro.com/t/ZRisXHZG2zpYD

Appendices

A. Course Outline

Week	Lecture
1	Introduction to Software Engineering Concepts
	Non-functional requirements
2	Usability, Low & high fidelity prototype
	UML (RationalRose Tool)
3	Software Development Methodologies Overview
	Waterfall Methodology
4	Pre-study Assignment
	Agile Manifesto
5	Group organization
	Specify group projects
6	Appropriate IDE installation (AndroidStudio for Mobile Applications Projects, etc)
	Case Study
7	Scrum
	Refactoring
8	Kanban Game
	eXtreme Programming
9	Code Integration & Version Control
	Github
10	Estimation
	User stories & Planning Poker
11	Quality Assurance and Testing
	Follow-up Lecture
12	Learning Stand-up Meeting
	Release and sprint Planning
13	Communication
	Follow-up Lecture
14	Follow-up Lecture
	Follow-up Lecture
15	Follow-up Lecture
	Follow-up Lecture

B. Agile Values

Agile Values	What to investigate	How to measure
Individuals and interactions over processes and tools	Which leads/drives the other, which responds?	In the case of individuals: communication happens when a need arises. In the case of process: communication is scheduled and requires specific content.
Working software over comprehensive documentation	What are the used documents?	Necessity of: • user stories (imp. to begin the task of building a new function) in form of scrum artifacts: - the product backlog, - the sprint backlog, • Class-based documentation • Estimation excel sheets. • Burn down charts. Wasting time if doing more extensive documentation.
Customer collaboration over contract negotiation	When Customer is involved and how much?	If customer was involved before development began and after it was completed>> Not Agile. But if during the process>> Agile.

		It will be measured through simple reports from the product manager to the teacher.
Responding to change over following a plan		Mentoring the team's transaction on "Trello" tool to investigate if they can benefit from the shortness of the iterations so they do shifting priorities from iteration to iteration and adding new features into the next iterations. Which is appear clearly in the modified product and sprint backlogs, and the student's behaviours through each sprint in the system log.

C. Agile Principles

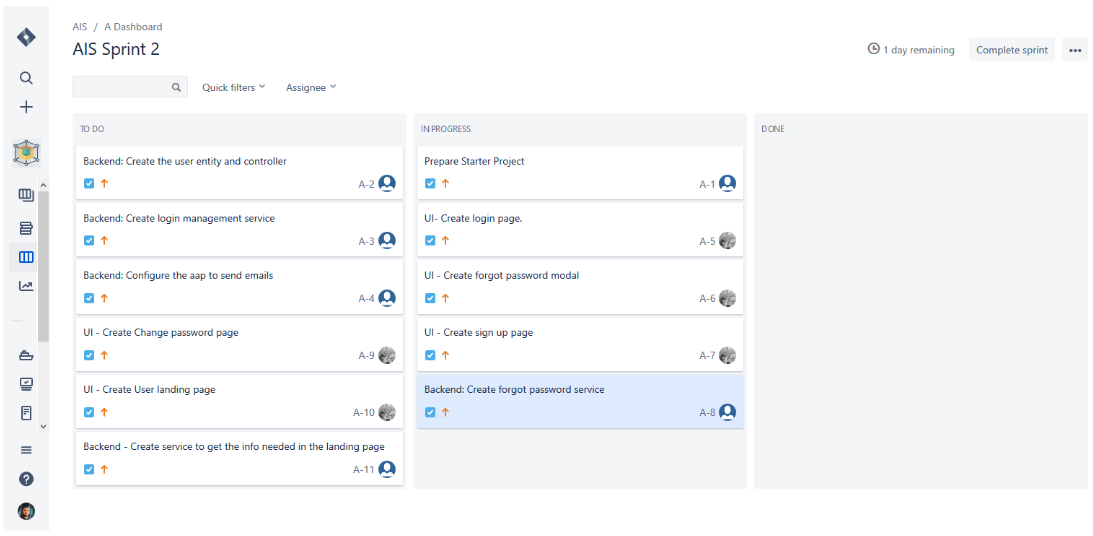
Agile Principles	How to be evaluated
Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.	By a discussion in the middle of work and at the end of the project, asking the customers if they were satisfied of receiving working software at regular intervals, or they were having to wait extended periods of time between releases.

Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.	By asking the team and the customer if requesting changes caused any/excessive delays. Or if have not been done.
Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.	Find by the "Trello" tool the percentage of unfinished features per sprints.
Business people and developers must work together daily throughout the project.	Did the customers participate in decision making? Prompt teams to show cases.
Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.	Manage the complexity of projects choosing/ Entrainments Then measure satisfaction and achievements.
The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.	Lectures Dedicated Rooms/labs and daily scrum
Working software is the primary measure of progress.	Delivering functional software to the customer is the ultimate factor that measures progress.
Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.	Monitoring teams/members progress and releases
Continuous attention to technical excellence and good design enhances agility.	Supportive learning by suggestion of MOOCs and some courses from Coding Dojo
Simplicity--the art of maximizing the amount of work not done--is essential.	If students doing prioritization And treat epics effectively.

D. Scrum Artifacts

More for Scrum	How to deal with
Daily Scrum	There is a specific hour daily from 8:00 to 9:00, each team should book a quarter hour meeting on Doodle a day before. And when coming presence. PS: if there is a problem in booking, it is returned to Scrum master.
Product backlog	Seen on Trello
Sprint backlog	Seen on Trello

E. Jira Dashboard Screenshots


AIS / A Dashboard

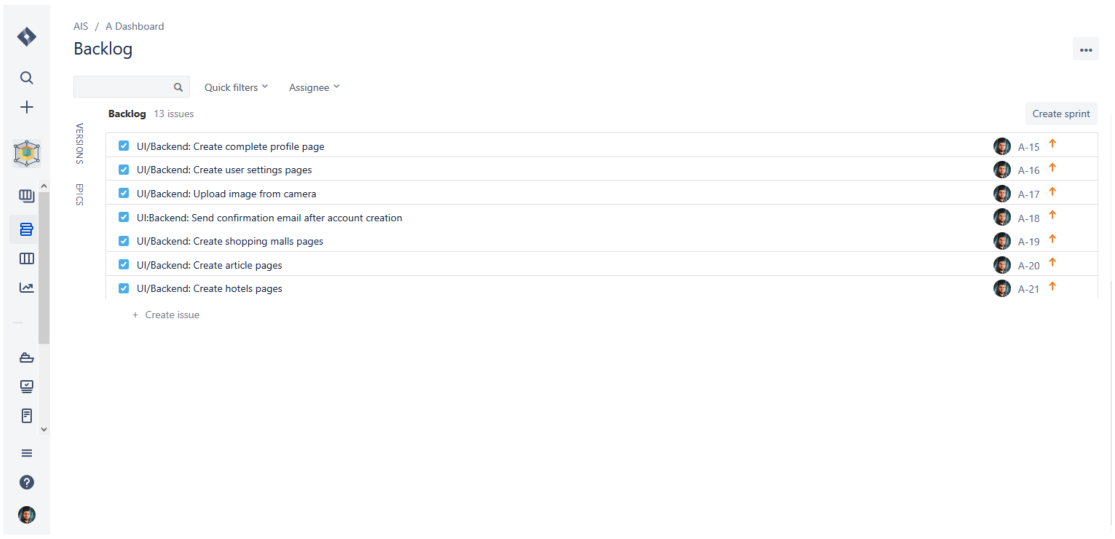
AIS Sprint 2

1 day remaining | Complete sprint | ...

Quick filters | Assignee

TO DO	IN PROGRESS	DONE
Backend: Create the user entity and controller — A-2	Prepare Starter Project — A-1	
Backend: Create login management service — A-3	UI- Create login page. — A-5	
Backend: Configure the aap to send emails — A-4	UI - Create forgot password modal — A-6	
UI - Create Change password page — A-9	UI - Create sign up page — A-7	
UI - Create User landing page — A-10	Backend: Create forgot password service — A-8	
Backend - Create service to get the info needed in the landing page — A-11		

Fig. E-1. Active Sprint


AIS / A Dashboard

Backlog

Quick filters | Assignee

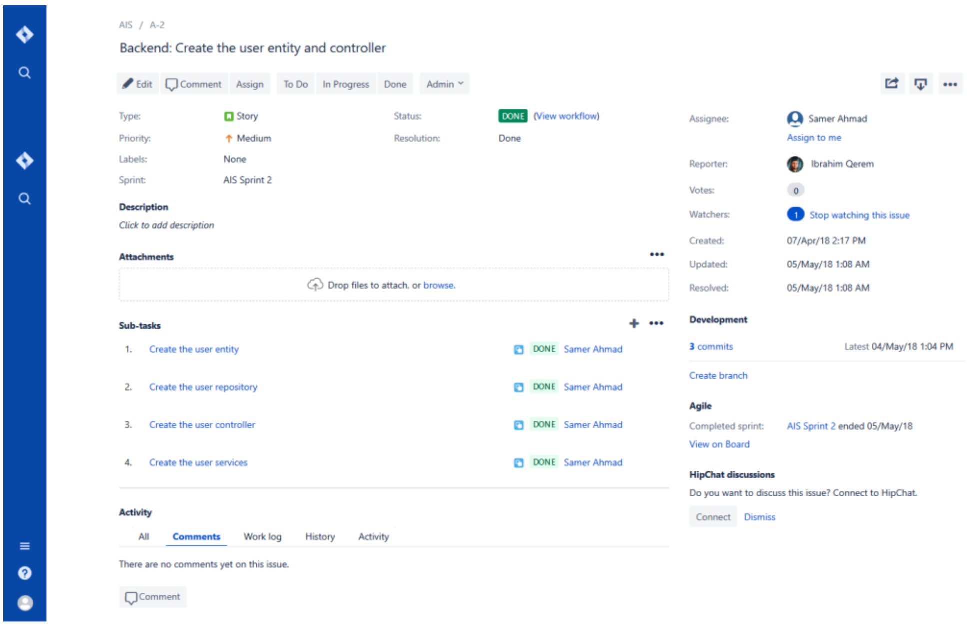
VERSIONS | EPICS

Backlog 13 issues | Create sprint

UI/Backend: Create complete profile page	A-15
UI/Backend: Create user settings pages	A-16
UI/Backend: Upload image from camera	A-17
UI:Backend: Send confirmation email after account creation	A-18
UI/Backend: Create shopping malls pages	A-19
UI/Backend: Create article pages	A-20
UI/Backend: Create hotels pages	A-21

+ Create issue

Fig. E-2. Backlog


AIS / A-2

Backend: Create the user entity and controller

Edit | Comment | Assign | To Do | In Progress | Done | Admin

Type: Story
Priority: Medium
Labels: None
Sprint: AIS Sprint 2
Status: DONE (View workflow)
Resolution: Done

Description

Click to add description

Attachments

Drop files to attach, or browse.

Sub-tasks

1. Create the user entity — DONE — Samer Ahmad
2. Create the user repository — DONE — Samer Ahmad
3. Create the user controller — DONE — Samer Ahmad
4. Create the user services — DONE — Samer Ahmad

Activity

All | Comments | Work log | History | Activity

There are no comments yet on this issue.

Comment

Assignee: Samer Ahmad
Assign to me
Reporter: Ibrahim Qerem
Votes: 0
Watchers: 1 Stop watching this issue
Created: 07/Apr/18 2:17 PM
Updated: 05/May/18 1:08 AM
Resolved: 05/May/18 1:08 AM

Development

3 commits — Latest 04/May/18 1:04 PM
Create branch

Agile

Completed sprint: AIS Sprint 2 ended 05/May/18
View on Board

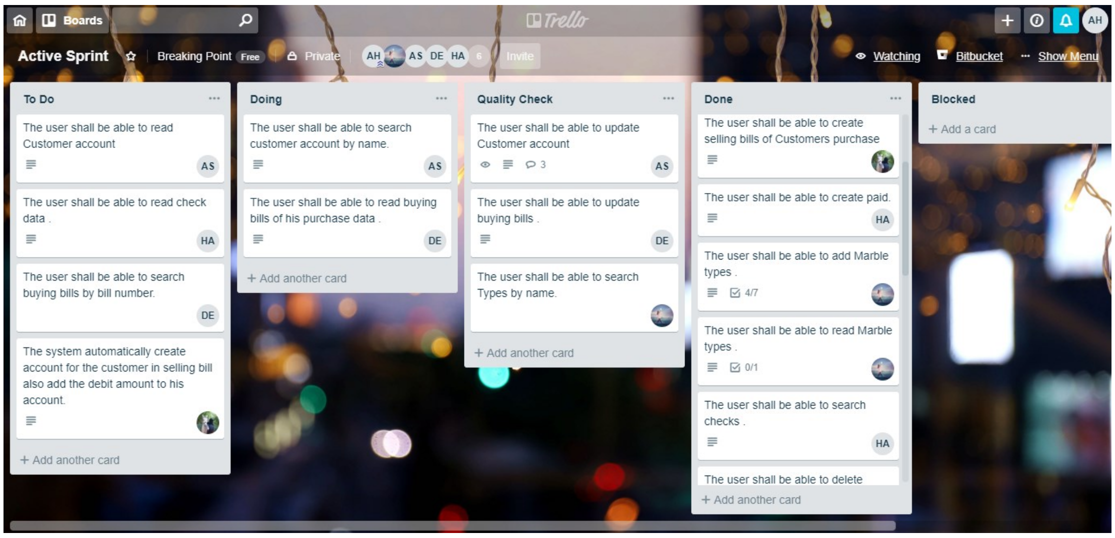
HipChat discussions

Do you want to discuss this issue? Connect to HipChat.

Connect | Dismiss

Fig. E-3. Task Details

F. Trello Boards Screenshots


Boards | Trello

Active Sprint | Breaking Point Free | Private | AH AS DE HA | Invite | Watching | Bitbucket | Show Menu

To Do
- The user shall be able to read Customer account — AS
- The user shall be able to read check data . — HA
- The user shall be able to search buying bills by bill number. — DE
- The system automatically create account for the customer in selling bill also add the debit amount to his account.
- + Add another card

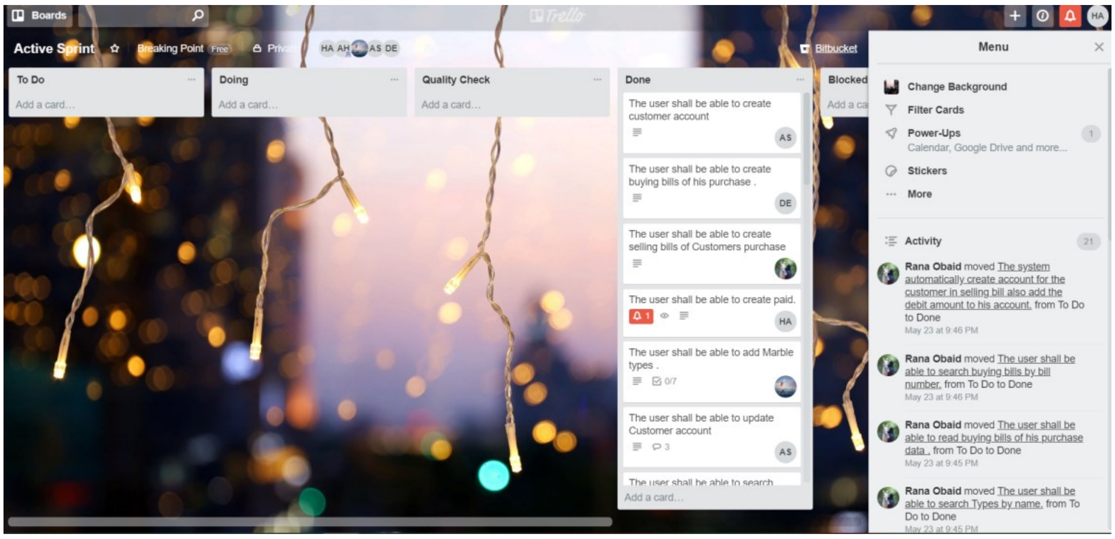
Doing
- The user shall be able to search customer account by name. — AS
- The user shall be able to read buying bills of his purchase data . — DE
- + Add another card

Quality Check
- The user shall be able to update Customer account — 3 — AS
- The user shall be able to update buying bills . — DE
- The user shall be able to search Types by name.
- + Add another card

Done
- The user shall be able to create selling bills of Customers purchase
- The user shall be able to create paid. — HA
- The user shall be able to add Marble types . — 4/7
- The user shall be able to read Marble types . — 0/1
- The user shall be able to search checks . — HA
- The user shall be able to delete
- + Add another card

Blocked
- + Add a card

Fig. F-1. Active Sprint



Done:
- The user shall be able to create customer account (AS)
- The user shall be able to create buying bills of his purchase . (DE)
- The user shall be able to create selling bills of Customers purchase
- The user shall be able to create paid. (HA)
- The user shall be able to add Marble types . 0/7
- The user shall be able to update Customer account 3 (AS)
- The user shall be able to search

Activity:

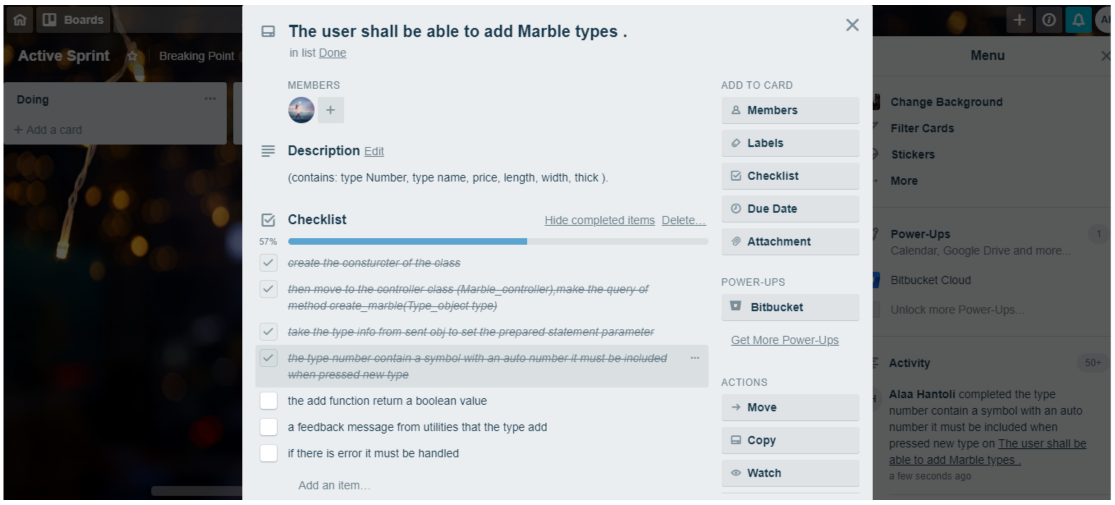
Rana Obaid moved The system automatically create account for the customer in selling bill also add the debit amount to his account. from To Do to Done
May 23 at 9:46 PM

Rana Obaid moved The user shall be able to search buying bills by bill number. from To Do to Done
May 23 at 9:46 PM

Rana Obaid moved The user shall be able to read buying bills of his purchase data . from To Do to Done
May 23 at 9:45 PM

Rana Obaid moved The user shall be able to search Types by name. from To Do to Done
May 23 at 9:45 PM

Fig. F-2. Sprint Review



The user shall be able to add Marble types .
in list Done

Description Edit

(contains: type Number, type name, price, length, width, thick).

Checklist Hide completed items Delete...

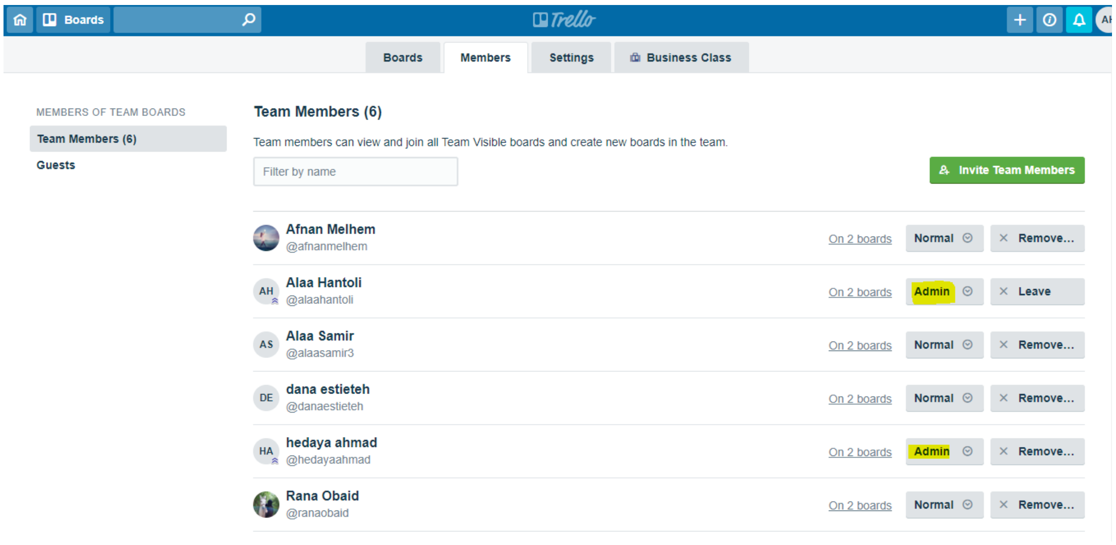
57%

- [x] ~~create the consturcter of the class~~
- [x] ~~then move to the controller class (Marble_controller),make the query of method create_marble(Type_object type)~~
- [x] ~~take the type info from sent obj to set the prepared statement parameter~~
- [x] ~~the type number contain a symbol with an auto number it must be included when pressed new type~~
- [ ] the add function return a boolean value
- [ ] a feedback message from utilities that the type add
- [ ] if there is error it must be handled

Add an item…

Alaa Hantoli completed the type number contain a symbol with an auto number it must be included when pressed new type on The user shall be able to add Marble types .
a few seconds ago

Fig. F-4. Task Details



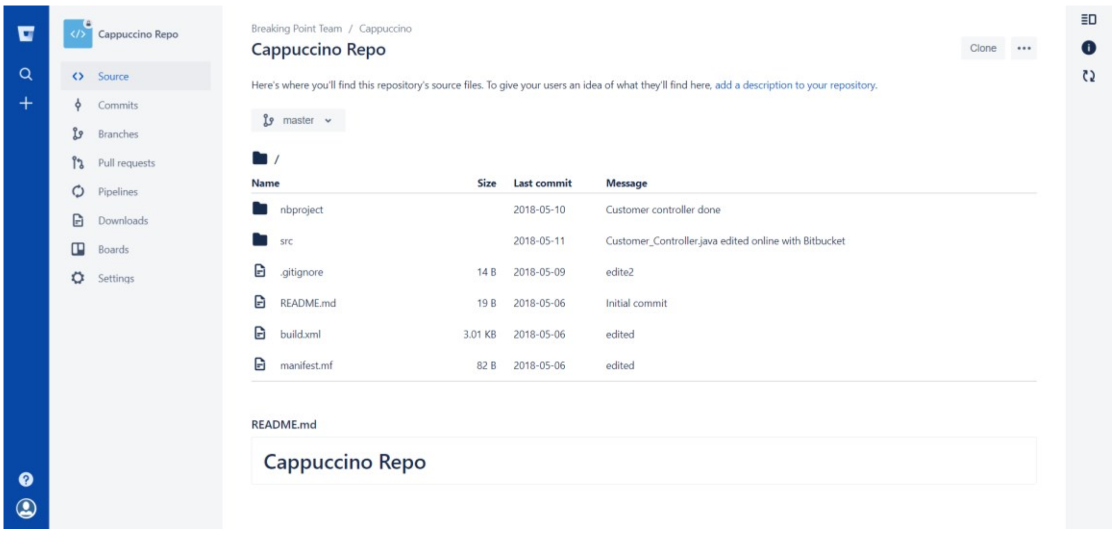
Team Members (6)

Team members can view and join all Team Visible boards and create new boards in the team.

Afnan Melhem @afnanmelhem	On 2 boards	Normal	× Remove…
AH	Alaa Hantoli @alaahantoli	On 2 boards	Admin	× Leave
AS	Alaa Samir @alaasamir3	On 2 boards	Normal	× Remove…
DE	dana estieteh @danaestieteh	On 2 boards	Normal	× Remove…
HA	hedaya ahmad @hedayaahmad	On 2 boards	Admin	× Remove…
	Rana Obaid @ranaobaid	On 2 boards	Normal	× Remove…

Fig. F-5. Team Members

G. BitBucket Screenshots



Cappuccino Repo

Here's where you'll find this repository's source files. To give your users an idea of what they'll find here, add a description to your repository.

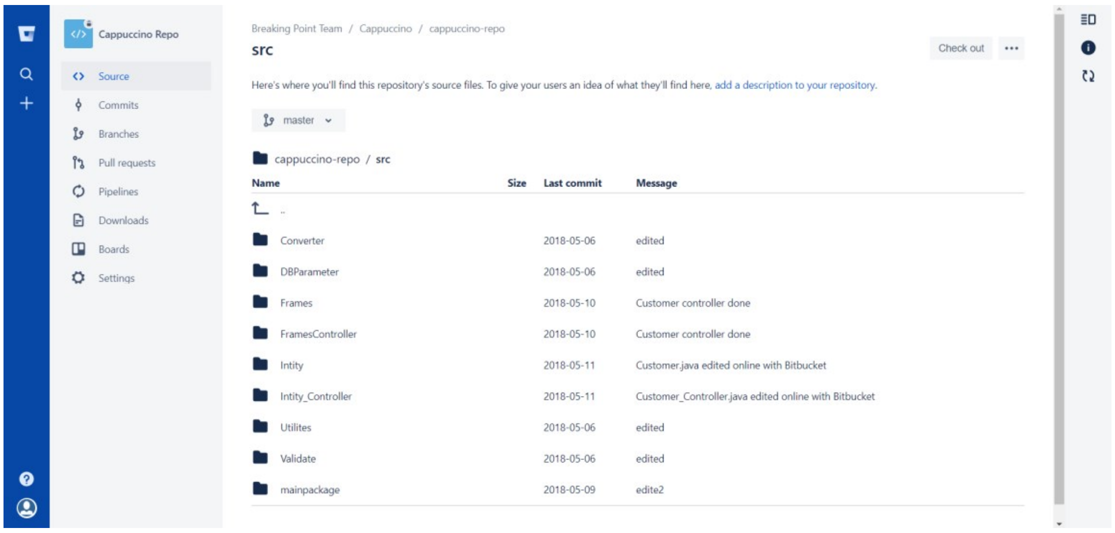
/

Name	Size	Last commit	Message
nbproject		2018-05-10	Customer controller done
src		2018-05-11	Customer_Controller.java edited online with Bitbucket
.gitignore	14 B	2018-05-09	edite2
README.md	19 B	2018-05-06	Initial commit
build.xml	3.01 KB	2018-05-06	edited
manifest.mf	82 B	2018-05-06	edited

README.md

Cappuccino Repo

Fig. G-1. Project Repository



Breaking Point Team / Cappuccino / cappuccino-repo

src

Here's where you'll find this repository's source files. To give your users an idea of what they'll find here, add a description to your repository.

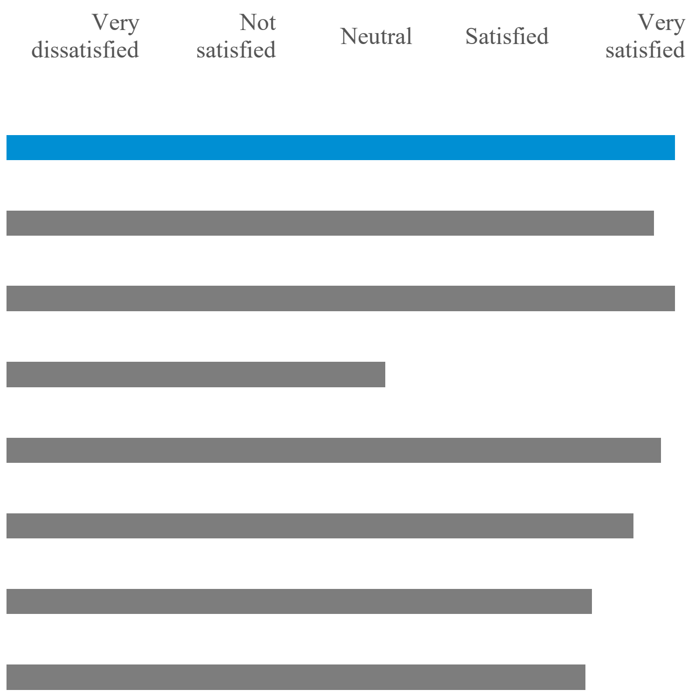
master

cappuccino-repo / src

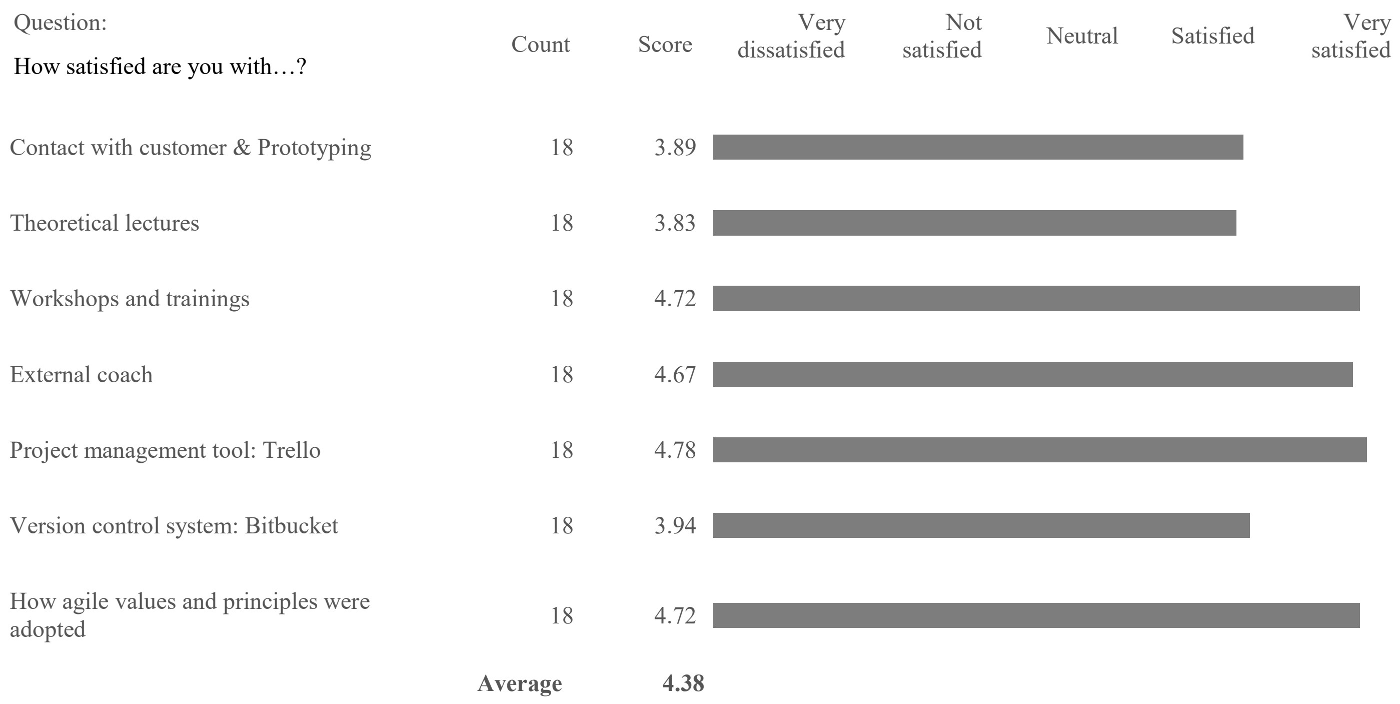
Name	Size	Last commit	Message
..			
Converter		2018-05-06	edited
DBParameter		2018-05-06	edited
Frames		2018-05-10	Customer controller done
FramesController		2018-05-10	Customer controller done
Intity		2018-05-11	Customer.java edited online with Bitbucket
Intity_Controller		2018-05-11	Customer_Controller.java edited online with Bitbucket
Utilites		2018-05-06	edited
Validate		2018-05-06	edited
mainpackage		2018-05-09	edite2

Fig. G-2. Project Source Files

H. Satisfaction Results



Question: How satisfied are you with…?	Count	Score
Overall experience at university	18	4.89
Development of a real product	18	4.72
Teamwork	18	4.89
Project time	18	2.78
Communication platforms	18	4.78
Daily scrum	18	4.56
Change management	18	4.28
Frequent releases	18	4.22



Question: How satisfied are you with…?	Count	Score
Contact with customer & Prototyping	18	3.89
Theoretical lectures	18	3.83
Workshops and trainings	18	4.72
External coach	18	4.67
Project management tool: Trello	18	4.78
Version control system: Bitbucket	18	3.94
How agile values and principles were adopted	18	4.72
	Average	**4.38**

Fig H. Whole Statements Satisfaction