2017 IEEE/ACM 1st International Workshop on Software Engineering Curricula for Millennials (SECM)

# Agile for millennials: a comparative study

Marcello Missiroli, Daniel Russo, Paolo Ciancarini
Department of Computer Science & Engineering
University of Bologna
Mura Anteo Zamboni, 7, 40126, Bologna, Italy
{marcello.missiroli, daniel.russo, paolo.ciancarini}@unibo.it

***Abstract*—*Context*: Comparative study of software development methodologies in millenials high school students.**
***Objective*: This paper compares the performance and satisfaction of both students and teachers in using two development strategies in a K-12 Computer Science teaching practice.**
***Method*: This study includes an experiment, administered in a laboratory controlled setting to measure students' performances.**
***Results*: Scrum produces less usable software with better working features compared to Waterfall; it is also more appreciated by students. Teachers are more experienced with Waterfall, which also offers more control on the development process.**
***Conclusions*: From an educational perspective, the two methods are equivalent. Since we noted an overall positive effect on the learning environment, we recommend that at least one method be taught in practice.**

***Index Terms*—Software engineering education; K-12 education; Scrum; Waterfall.**

## I. Motivation

At the turn of the century, the Agile movement [10] made its appearance on the Software Engineering scene. It proposed a radically new development process, offering a compromise between not enough process and too much process, being adaptive rather than predictive and more people-oriented rather than plan-oriented [17], [23]. It was a response to the limitations of software engineering practices of the time and now, 15 years later, it is not a curiosity limited to few enthusiasts anymore, but a well-established practice [27]. Studies show that Agile brings distinctive advantages [1], especially in case of small projects and when practices are correctly executed [2].

However, Agile is not taught as part of standard software development syllabi in most universities, although it is sometimes introduced - and rarely tested firsthand - in graduate courses. As a result, Agile development practices are learned "on the field", often after attending expensive seminars. In addition, teaching Agile methods in high school is rare at best. Since Agile methodologies have been successful in the professional world, we decided to test if applying them in a K-12 environment would improve the learning process, especially given that several key Agile values are very important in the light of modern teaching approaches, based on constructionism [20] and constructivism [30].

We show the results of a study conducted on young programmers and their teachers in a high school context, focusing on the final part of the Software Development Life Cycle [SDLC]. Following the path of evidence-based education [5], we designed an experiment in which the same software project is realized by two teams of the same class, each using a different development methodology. We performed the experiment in seven classes in different schools. This duel-like comparison is an opportunity to observe the potential of different methods, limiting the influence of environmental aspects such as class level, teacher influence, and similar. We considered several different factors, such as grades received.

Results show some advantage for Scrum, but not enough to recommend a change. Given the overall positive effects on the learning environment, we recommend that at least one development method be taught to millennial students, provided it is integrated in a coherent learning structure.

This paper has the following structure: section II summarizes the research on the subject; section III describes the experiment we conducted; then in section IV we perform an analysis on the quantitative data, noting the statistical significance of the experiment results. in section V we propose an overall interpretation of our investigation; finally, in section VI we draw our conclusions and indicate the future direction of our research.

## II. Related work

There are some studies related to the effectiveness of Agile methodologies in a professional environment. Results are not decisive, ranging from undecided results [28] to noting a small but perceivable positive effect [7]. Some studies tried a direct comparison and detected more pronounced advantages in certain contexts, such as startups [8], [33] and small scale projects [34]. Christelle's [3] is one of the few studies not centered on raw productivity, in this case stress; a different study discovered better empowering awareness in case of small teams [26] .

In an educational context we note that, on average, Agile teaching is not that widespread [14]. However, interest in this field is rising, and curricula are being updated according to this [15], [25], also prompting for more research. Even in this case, results are contradictory [11], [22].

There is some research on Agile in high school. One of the most complete examples is [16]. The paper presents the "Agile Constructionist Mentoring Methodology" and its yearlong implementation in high school. It considers all aspects of software development, with a strong pedagogical support. Apart from that, we have evidence of studies in European cases such as the application of the Eduscrum method [6], Extreme

978-1-5386-2795-2/17 $31.00 © 2017 IEEE
DOI 10.1109/SECM.2017.7

apprenticeship [29] and a German research group [21]. None of these, however, offer rigorous quantitative analysis of the results. The meta-approach of [9] is also noteworthy: it applies Agile principles to the learning process itself, rather to single subjects.

In our previous works [18], [19], we considered the impact of Agile *coding practices* in a K-12 environment, determining how pair composition and environmental conditions impact the learning outcomes. It this paper, however, we consider the effects of the development strategy on the overall learning environment. To the best of our knowledge, this is the first paper addressing the issue in a rigorous way.

## III. Experiment

In this section, we describe in detail the experiment we performed. We are interested to perform the following Research Question (RQ): *Is the Scrum development methodology better that Waterfall in an educational context?* According to that we developed the following Research Hyphotesis:

- $H0_{PROG/GRA/SCORE/SCORE-*}$: There is no difference in the mean value of any project metrics described above between the software development projects using either the Waterfall or the Scrum development methodology. SCORE-* indicates all possible partial values of SCORE: SCORE-F, SCORE-XF, SCORE-1, SCORE-2.
- $HA_{PROG/GRA/SCORE/SCORE-*}$: There is a difference in the mean value of any project metrics described above between the software development projects using either the Waterfall or Scrum development methodology.

The research is summarized as follows:

**Object of study**. The object of the study is the outcome of a programming project of about 25 coding hours.
**Purpose**. The purpose is to compare the effect of Waterfall and Scrum development in a high school project-based learning environment.
**Quality focus**. The quality focus is the students' results.
**Perspective**. The perspective is the teachers' one.
**Context**. The experiment is conducted using high school students in their 5ánd final year, having at least two years of programming experience.

Summary: *The analysis of the outcome of a programming project for the purpose of evaluating the impact of different development methodologies as a learning tool in high school with product evaluation from the point of view of teacher in the context of complex programming development performed by school students*.

We follow the experiment methodology proposed by Wohlin, et al. [32]: context selection, hypotheses formulation, variables selection, selection of subjects, experiment design, instrumentation, and validity evaluation.

### A. Context selection

The context of the experiment was the students of the 5ánd last year of Computer Science course in Italian high schools featuring CS programming courses. Compared to many other developed countries, CS teaching shows some interesting peculiarities after the latest reforms.

For example, programming is *not* a widespread subject. The concept of major or minor discipline does not exist, as the curriculum is linked to the type of school chosen and determined by the Ministry of Education. This strongly centralized organization of disciplines restricts the teaching of computer programming to the following types of schools: Applied Science Gymnasium (2 CS hours/week, for five years), Industrial Technical Schools (ITIS, 3-8 hours/week, three years) and some Economic Technical Schools (ITES, 3 hours/week, three years). As a result, only about 25% of the current school population is instructed in programming and has some experience in software development. Within the scope of our investigation, this dramatically reduces the size of potential experiment subjects (both teachers and students). After a nationwide search for partners we found seven schools in six different cities that were willing and able to participate in the experiment, of distributed in 34 teams — about 160 students overall. All the students are typical millennials, born in 1998-9, attending the final year of either ITES or ITIS and therefore facing the final "State Exam", that requires basic Software Engineering knowledge.

### B. Metrics

It is always difficult to establish the success of a project, since the meaning of the word "success" changes greatly depending on the context. In our case we are to establish the success of a school simulation of a project with respect to three different goals: building a successful software product that fulfills the requirements, successfully implementing a software development methodology, and evaluating the success of the experience as a learning tool.

Hence, we had to identify a three different "success" metrics that had to be as objective as possible and easy to evaluate. We chose the following:

- **Progress** (PROG). This is form of product verification [13].It is expressed as a percentile indicating the amount of requirements that the final product has satisfied. In the case of Waterfall development, it consists on the total of functional and non-functional requirements. In the Scrum case, it consists of the percentage of the completed user stories.
- **Grade** (GRA). This is a form of product validation [13]. In this case the "product" is not the software itself, but rather the overall impact of the learning experience on the class. As such, only the class teacher can determine the success of the experience. The grade the teacher would have given to the team will therefore be used as success indicator. If it may appear as a subjective factor, we note that in Italy grading is rarely a mechanical translation of points, checklists, and rubrics into numbers, but involves a series of other contextual and subjective considerations; the overall result is that students get a very limited grade fluctuation throughout the year (meaning that the current average grade becomes an excellent estimate of how they

will perform in similar situations) and grade distribution becomes a normal distribution centered on the more-than-sufficient grade (6.5 out of 10, or C+).
- **Score** (SCORE). This is a form of process verification [4]. It evaluates the impact of the methodology used on the development process, informally expressed by the question "Did we build it following the rules?". In this case, progression of the project was awarded **only** if obtained following the rules of the method. So, the difference between PROG and SCORE measures the impact of the methodology on the final product. For example, a user story that was indeed implemented but was not part of the current sprint does not count in the SCORE metric, but it does count in the PROG metric. To have a better understanding of said impact, we kept a separated SCORE value of functional and a combination of non-functional requirements (e.g. performance, usability, defects) and process-related indicators (e.g. timely delivery, artifacts) named *Extra-functional requirements*. These percentile indicators were labeled SCORE-F and SCORE-XF, respectively. In addition, we kept the partial score records of the first and second phases of the experience.

### C. Group formation

Each class participating in the experiment was divided in an even number of groups. Group formation was a modified stratified random selection, to obtain balanced skill level groups. However, the teacher could, if necessary, modify the group composition in case of particular situations, such as psychological incompatibilities, but limited to one person per group. In practice, these instances turned out to be very few - five cases overall. In Waterfall, roles were pre-determined, whereas in Scrum the team had to pick its Scrummaster.

### D. Design

The most common limitations of any school settings were considered and mitigated in the following ways:

- **Limited time use**. We decided that no more that 8 hours could be used on the project, about 4% of the yearly CS-allocated hours in an ITIS (twice as much in an ITES). The drawback was that students would not be able to fully complete the project within the given time.
- **Structure**. We devised a problem that could in some way be seen as a form of preparation for the State Exam. We therefore chose very specific and "classic" curriculum points.
- **Learning opportunity**. The activity was structured as form of active learning experience for *both* teachers and students. This required writing substantial documentation and offering direct (teaching) and indirect (videos, material) support to teachers.
- **Flexibility**. Every school has its own peculiarities; we allowed great flexibility in dates, communication means, tools, practices and programming languages to be used.

The product that the participants were required to build was a dynamic, database-using web application. Developing from scratch a project of this kind requires, by our estimates, about 20-30 hours. So, we had to devise a way to fit it in our limited 8 hour format, that also included initial formation. Each team would be assigned a project and randomly paired to another team of the same class: one would develop it using Waterfall, the other using Scrum - this choice was also randomly determined.

We decided to focus the experience on the *coding and testing* parts of the project, thus leaving out initiation and design. In addition, we only wanted a significant, working prototype of the product, that could give a general idea of the final results to the potential stakeholders. Note that as this will be, after all, a simulation of the real development process, the "final product" would probably be more similar to an alpha-quality prototype (in the Waterfall case) or a Potentially Shippable Product of a Sprint (in the Scrum case), but the students were not informed of this. In both cases, choosing which features to implement in the product was left to the team.

As we knew that up to 8 teams could take part to the experiment, and simultaneous execution of all classes was unlikely due to schedule, we prepared *four* different projects. All featured similar requirements: use of databases, web interface, user authentication, basic insertions and modifications of data. They differed in the application domain and the relative importance of requisites. The four projects were: a task list reminder, a casual chatting social site, a school auditorium seat reservation system, and a movie theater digital signage system.

For each of the projects, we had to prepare adequate documentation to compensate for the lack of the design phase in our project.

**Waterfall**. We wrote a complete Software Requirement Specification (SRS), which included Product Description, E-R Diagrams, Use Cases diagrams and description, each detailing input, output, process, user interfaces diagrams, and non-functional requirements. The SRS was modeled after the ones used in university courses. In addition, we prepared a list of management-decided priority objective in the MoSCoW format.

**Scrum**. We wrote a series of User Stories that formed the backlog of the project, including acceptance conditions and story point estimation. Even non-functional requirements were implemented in this way. All US were organized in three tiers (high, average, low) that simulated backlog prioritization by the Product Owner.

Even with the above constraints and plentiful available materials, finding partners was very difficult. Out of over 200 schools that were contacted, only 36 responded. At the end, only seven schools confirmed their participation. The experiment is designed to simulate a standard development process with either Scrum or Waterfall. The limitations discussed above introduce several differences when comparing process with real world ones, aside from the obvious extremely

short time at disposal. Namely:

**Waterfall**. Switching from the Coding phase to the Testing phase is determined by time. It is not a decision of the Project leader, determined by the maturity of the code. Testers and programmers are part of the same team, so overlapping of responsibilities are inevitable.

**Scrum**. The Product Owner is not a peer, but the teacher, this might have psychological effects. User Stories can be "partially accepted", if the team is willing to take the performance penalty, and are pre-evaluated.

This simulation is sufficiently close to the real process, in both cases, to allow us some interesting considerations.

### *E. Planning - Instrumentation*

The instrumentation we used included the computer labs of the schools, a standard Internet connectivity, and the usual development tools used in CS classes (e.g., IDEs, compilers).

### *F. Validity*

Concerning the validity explanation of our experiment, we are aware of statistical conclusion, internal, construct and external validity threats [24].

Threats to the **statistical conclusion** are considered to be under control. Robust statistical techniques, tools and representative sample sizes to increase statistical power are used. Moreover, measures and treatment implementation are considered reliable.

Threats to **internal** validity are considered to be under control. Random assignment of both research subjects and research tasks was designed to maximize internal validity. Test subjects were evaluated by their own teachers, so we assume continuity with their standard school performance.

Threats to **construct** validity are not considered very harmful. The inadequate explication of constructs is considered to be a an incentive for the elaboration of better educational frameworks for the education of future computer scientist generations. Even if we considered the literature background to be inadequate, we based our design on constructivism. However, we recognize that there is an urgent need to rethink the educational approaches to computer science, especially in high school. Limitations to experiments in high school were already discussed and considered to be under control.

Threats to **external** validity are not considered very harmful. Our sample was randomly defined, considering students coming from several different schools located in different cities, maximizing external validity. This study involved over 160 K-12 students, from six Italian provinces, representative of Italy. The sample covered Northern, Central and Southern Italy, with students of different backgrounds.

As we know, there is a constant trade-off between internal and external validity and our sampling strategy took this into account [12]. It was concluded that the threats were not regarded as being critical.

### *G. Operation*

The experiments were conducted from January to May 2016; team size was 4 to 6 elements, with an average of 5, so about 160 students took part to the experiment. It consisted of two sections, the instructional and the operational one. To have a standard reference base and to minimize interpretation problems, a proper protocol document was prepared and sent to all participating teachers.

*1) Instructional section:* It consisted in a 2-hour hands-on experience. Its goal was to provide the students with the theoretical knowledge and some limited experience with the two development methodologies. This was deemed necessary, since Agile methodologies are not part of the current curriculum, and even Waterfall methodologies are usually taught only in theory, and at different points of the school year. We needed to give the participants a common knowledge base.

The experience had the following schedule:

- **Waterfall lecture**. A standard 30-minute lecture was given on the Waterfall model. It defined the various phases, roles, the main artifacts, advantages and disadvantages.
- **Waterfall workshop**. The class worked in groups, each receiving the text of a State Exam of some years ago. They had 15 minutes to sketch the general structure of a SRS and detail one element (usually, the E-R diagram).
- **Scrum lecture**. A standard 35-minute lecture was given on Agile development in general and Scrum in particular. Again, it defined the various terms, roles, advantages and disadvantages.
- **Scrum workshop**. The class worked in groups, each receiving the text of a State Exam - different to the previous one. They had 20 minutes to write three user stories, at least one of which with acceptance conditions, and pick the conditions for their own Definition of Done from a list of six items.
- **Wrapping up**. All material was collected and given to the teacher for proper feedback (but not grading). The teacher used the last 5 minutes to outline how the experiment would be organized.

The lecture was delivered: in person by the first author (three cases); made by the class teacher (four cases); or using a prerecorded video (one case). Considering that the sampled schools ranged over 700km distance, this was the most cost-effective solution.

*2) Operational section:* The most important part of the experiment required a full 6-hour lab session. First, it was revealed to the students which group they belonged. Groups were then paired and randomly given either the Scrum or Waterfall methodology. Finally, they were randomly assigned to one of the four projects. They were then ready to start, and now the path of Agile and Traditional developers diverged (see Fig.1).

Waterfall developers were given a sizable quantity of material: general project description; a list of requirements in the MoSCoW format; a complete SRS, averaging 28 pages, that

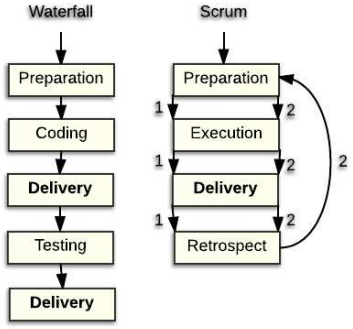


Fig. 1. Comparing the process models. Numbers indicate iterations

contained full use case coverage, ER-diagrams, user interface schemes, functional and non-functional requirements. They were allowed 5 hours for the coding section of the project, and were expected to deliver a "feature-freeze" codebase when time expired. They had one more hour for testing, but no new feature development was allowed. At the end of the morning, they had to deliver the final, hopefully functioning, product. During the experiment, the teachers assumed the role of a non-tech manager. They should refrain from giving technical advice, but were welcomed to provide assistance at a manager level (time issues, behavioral problems, etc.), even giving orders if required.

Scrum developers were provided with far less material: general project description; definition of done; user stories (prioritized in three categories). Their time allocation was structured in two $2\frac{1}{2}$-hour mini-sprints. Each consisted in Sprint preparation (15'), Sprint execution (110') and Delivery and retrospect (5'). A mandatory pause of 15' was also included. In this case, teachers assumed the role of the Product Owner of the teams. As such, they were well expected to provide some guidance on how the final product should look like, but could in no way order people around or criticize their behavior. They were also the final judges of whether a User Story was effectively "done, done, done".

*3) Post-execution activities:* All material created during the two phases of the experiment (Coding/Testing for WF, Sprint 1 and 2 for Scrum) was collected and sent to us. It included code, database dumps, and passwords. In addition, we required the teachers to fill a grading form. It allowed us to determine which feature was effectively realized (and how), the grade the group received and a subjective evaluation of the group performance and expected performance. An electronic survey was then submitted to all participating students, immediately after the experience. Toward the end of the school year, all teachers were contacted either in person or by phone to perform an extensive reflective interview.

## IV. Quantitative Analysis

In this section, we show and analyze the results of our experiment. The aim was to understand if there is a significant difference in the students' performance using different development strategies.

TABLE I
Scrum vs. Waterfall 'competition' results

| | Scrum advantage | | Draw | Waterfall advantage | |
|---|---|---|---|---|---|
| | 10+% | 5%-10% | $\Delta < 5\%$ | 5%-10% | 10+% |
| SCORE-F | 5 | 3 | 4 | | 1 |
| SCORE-XF | 1 | | 2 | | 10 |
| SCORE | 3 | 2 | | 2 | 1 |
| PROG | 2 | 5 | 2 | 1 | 3 |

### A. Direct comparison

We directly compared the scores and progress of teams of the same class that worked on the same project, but with different methodologies. This comparison is significant, because it eliminates biases due to environmental conditions, teaching methods, class level, and more. The result of this competition is shown in Table I. Unfortunately, due to organizational problems, we only have 13 direct comparison at our disposals instead of 17. In addition, usability was not tested directly by class teachers, but assessed by ourselves, using pre-prepared acceptance tests.

It is easy to see that Scrum teams concentrated on functional requirements, whereas Waterfall teams devoted themselves to other aspects (confirming the conclusions of [11] in a different context). If we consider the overall score, we see a slight advantage for the Scrum camp. If we however consider the PROG score, i.e. only evaluating the final product without penalties for not following the methodology, the advantage of Scrum becomes more relevant.

We point out that, concerning the extra-functional aspect of usability, Waterfall displayed a pleasant, sometimes innovative interface, but were unable to deliver useful functionalities. On the other hand, though Scrum projects had a variety of working functionalities, they were hidden behind a very crude user interface, and had abysmal usability problems.

### B. Performance Statistical Analysis

Experimental data were evaluated with descriptive analysis and statistical tests. What we wanted to understand is if there was a significant difference among the two methodologies in terms of performance. So, we clustered all our experiments by type per methodology (i.e., SCORE-F, SCORE-XF, TOTAL SCORE and PROGRESS for Scrum and Waterfall, respectively). Descriptive statistics are summarized in Table II. We tested our sample for each type per methodology, for the most significant descriptive value (i.e., Mean, Standard Error, Median, Mode, Standard Deviation, Kurtosis, Skewness, Minimum, Maximum, number of observations (OBS), Confidence level (CL) at 95%).

We ran our analysis to compare the different performed tasks, using Scrum and Waterfall methodologies. The performance was evaluated using several metrics, according to our experimental setting explained in Section III-B. Data taken from the experiment are summarized in Table II.

TABLE II
DESCRIPTIVE STATISTICS

| | SCORE-F | | SCORE-XF | | TOTAL SCORE | | PROGRESS | |
|---|---|---|---|---|---|---|---|---|
| Parameter | Scrum | WF | Scrum | WF | Scrum | WF | Scrum | WF |
| Mean | 17.02 | 11.51 | 15.73 | 41.56 | 16.28 | 15.79 | 20.01 | 18.49 |
| Standard Error | 2.70 | 2.81 | 2.85 | 3.81 | 2.23 | 2.61 | 2.79 | 2.32 |
| Median | 18.68 | 7.86 | 11.67 | 44.83 | 18.84 | 12.75 | 18.08 | 19.48 |
| Standard Deviation | 11.15 | 11.59 | 11.76 | 15.70 | 9.20 | 10.76 | 11.49 | 9.57 |
| Kurtosis | -1.23 | 0.13 | 1.88 | 0.91 | -1.18 | -0.43 | -0.77 | 0.12 |
| Skewness | -0.10 | 0.92 | 1.37 | -0.73 | -0.46 | 0.67 | -0.09 | 0.24 |
| Min | -0.15 | 0.00 | 3.68 | 2.96 | 1.08 | 0.92 | 1.80 | 0.92 |
| Max | 35.24 | 38.15 | 47.20 | 67.60 | 29.22 | 36.88 | 40.00 | 36.88 |
| # Observation | 17 | 17 | 17 | 17 | 17 | 17 | 17 | 17 |
| Level of confidence (95%) | 5.73 | 5.96 | 6.05 | 8.07 | 4.73 | 5.53 | 5.91 | 4.92 |

TABLE III
HYPOTHESIS ANALYSIS

| | SCORE-F | | SCORE-XF | | SCORE-F | | PROG | |
|---|---|---|---|---|---|---|---|---|
| | Scrum | WF | Scrum | WF | Scrum | WF | Scrum | WF |
| Mean | 17.02 | 11.51 | 15.73 | 41.56 | 16.28 | 15.79 | 20.01 | 18.49 |
| Variance | 124.24 | 134.40 | 138.39 | 246.43 | 84.56 | 115.88 | 132.12 | 91.50 |
| # Observations | 17 | 17 | 17 | 17 | 17 | 17 | 17 | 17 |
| Pearson Correlation | -0.35 | | 0.17 | | -0.35 | | -0.12 | |
| Hypothesized mean difference | 0 | | 0 | | 0 | | 0 | |
| df | 16 | | 16 | | 16 | | 16 | |
| t Stat | 1.22 | | -5.92 | | 0.12 | | 0.39 | |
| P(T<=t) one tail | 0.12 | | 0.00 | | 0.45 | | 0.35 | |
| t Critical one tail | 1.75 | | 1.75 | | 1.75 | | 1.75 | |
| P(T<=t) two tails | 0.24 | | 0.00 | | 0.90 | | 0.70 | |
| t Critical two tails | 2.12 | | 2.12 | | 2.12 | | 2.12 | |

The use of any model needs to be validated. Thus, we ran a t Test of paired two samples for mean. This is because a sample from the population is chosen and two measurements for each element in the sample are taken. So, the two samples were independent of one another.

Table III shows the aggregated t Test for each type. Only the difference between extra-functional requirements performance have a significant difference, according to the two methodologies used. So, for this case we rejected $H0$, since the t Stat in absolute value is larger than the t Critical two tails coefficient. Now, to see how they perform differently, we test the means. Since the mean of the Waterfall performance is larger than Scrum, we conclude that for the extra-functional requirements, Waterfall performs significantly better.

For all other cases, we did not see any statistical significance, since the t Stat was smaller than the t critical. From a purely statistical perspective, we cannot argue that Scrum performs better than Waterfall.

## V. INTERPRETATION

In this section, we provide answers to the research questions.

We accept the $HA_{SCORE-XF}$ hypothesis in favor of Waterfall, based on the statistical significance of results on Extra-Functional aspects. We also accept the $HA_{PROG/SCORE/SCORE-F}$ hypotheses in favor of Scrum, based on the results of the direct confrontation, corroborated by the numerical results (though not statistically significant). We however accept the $H0_{GRA}$ hypothesis, since no significant grade difference is noted in the project outcomes.

### A. Discussion

Results do not show a precise, clear-cut tendency.

Both approaches offer advantage over individual projects related to learning environment,and overall project completion. Both had no clear idea of what business value is, Waterfall overlooking the functional aspect, Waterfall disregarding extra-functional aspects.

**Waterfall** has a decisive advantage in one field (non-functional and process-oriented requirements). It is also a well know practice to the average teacher, as interviews and our previous work [18] show, making implementation easier. Waterfall is also the most common industrial practice, especially in Italy, so it is not possible to eliminate it from curricula anytime soon.

**Scrum** has more all-rounded advantages overall. In a direct comparison, Scrum "beats" Waterfall in all categories, except in Extra-functional requirements. In overall results, it performs equals or slightly better on both score and process for all Projects considered. Student display slightly more interest to this methodology.

So, *both* methodologies have their strong points, and both should be taught, explored, tested. In our view, teachers should try to introduce Agile concepts and methodologies, as they might become dominant in the next future.

There might be, however, a third possibility. It is conceivable to introduce an hybrid methodology, colloquially referred to as "Waterscrum". This methodology is what happens in the industry as the first step towards a complete acceptance of Scrum (or Agile methods). It is apparently a very widespread ([31]), though not ideal, practice.

In the case of education, however, Waterscrum could be the balancing point, addressing the need of planning, instruction and direction for still inexperienced programmers as well as the opportunity for creativity and freedom offered by Agile methodology and practices. It will allow teachers to build on their existing knowledge gradually, without having to spend resources on courses. It would also allow a certain degree of customization as teachers have different style of teaching styles. Some prefer a more formal approach to projects, where other prefer a looser, experimental approach.

One important take away of this research is that K-12 students should be exposed early to software development methodologies. Besides the performance of the experiment, both teachers and students expressed their appreciation for the use of project development in the learning process. This requires integration of SDLC in the standard curricula and practices; the teaching community should take steps in this direction. The exact method is unimportant: just pick one. Care should be taken; project-based education can become a multiplier for better learning, but must be inserted in a general, solid framework. Start 'doing' with no bases or knowledge will result in a funny, but ultimately useless activity.

## VI. CONCLUSIONS AND FURTHER WORK

In this paper we put two radically different developing methodologies against each other in a K-12 context to check whether one has significantly advantages over the other as a learning tool.

In most fields, and especially in education, it is difficult to strike a balance between "form", "content" and "creativity". In schools, it is the personal sensibility of the teachers that suggests where to draw the line. So, formal oriented educators will be more at ease within the controlled environment of Waterfall-like projects, whereas innovative teachers will prefer the "Embrace the unknown" attitude of Agile methodologies. Each teacher should get informed and decide. We also note that school and classes are not all equals, each having radically different interests and potentials - which also requires a significant degree of customization, though millennials - on average - tend to value more responsefulness that planning, as shown in feedbacks.

In this view, it seems reasonable to strike a compromise between these two different methodologies and apply the so-called "Waterscrum" approach. It might offer the best of both worlds - planning and structure along with creativity and reactivity. It will also allow teachers to customize the methodology and to experiment and change little by little, acquiring experience as they proceed. Some teacher will stop very early, while others might go all the way to Agile. Whatever the choice, we think that at least one method be taught and experienced as soon as possible in high schools, each teacher choosing the method best suited to his and his classes' peculiarities. We are aware that our conclusions are based on a relatively large sample base, but on a limited and shortened version of the "real" development methodologies. However, since the focus is not on industrial software development assessment but on education, we believe that these results are meaningful for the community.

## ACKNOWLEDGMENTS

This research has been partially supported by ISTC–CNR and CINI.

## REFERENCES

[1] S. Ambler. 2013 IT Project Success Rates Survey Results.

[2] T. Chow and D.-B. Cao. A survey study of critical success factors in agile software projects. *Journal of Systems and Software*, 81(6):961–971, 2008.

[3] S. Christelle, S. Heng, K. Vidya, et al. Stress level on global software projects using waterfall and scrum: A preliminary comparison. In *The First Asian Conference on Information Systems*, 2012.

[4] J. E. Cook and A. L. Wolf. Software process validation: Quantitatively measuring the correspondence of a process to a model. *ACM Trans. Softw. Eng. Methodol.*, 8(2):147–176, Apr. 1999.

[5] P. Davies. What is evidence-based education? *British Journal of Educational Studies*, 47(2):108–121, 1999.

[6] T. de Jager. Using eduscrum to introduce project-like features in dutch secondary computer science education. http://dspace.library.uu.nl/handle/1874/307201, 2015.

[7] E. di Bella, I. Fronza, N. Phaphoom, A. Sillitti, G. Succi, and J. Vlasenko. Pair programming and software defects–a large, industrial case study. *Software Engineering, IEEE Transactions on*, 39(7):930–953, 2013.

[8] J. Dorette Jacob. Comparing agile xp and waterfall software development processes in two start-up companies, 2011.

[9] A. El-Abbassy, R. Muawad, and A. Gaber. Evaluating agile principles in cs education. *International Journal of Computer Science and Network Security*, 10(10):19–28, 2010.

[10] J. Highsmith and M. Fowler. The agile manifesto. *Software Development Magazine*, 9(8):29–30, 2001.

[11] F. Ji and T. Sedano. Comparing extreme programming and waterfall project results. In *2011 24th IEEE-CS Conference on Software Engineering Education and Training (CSEE&T)*, pages 482–486. IEEE, 2011.

[12] C. Judd, E. Smith, and L. Kidder. *Research Methods in Social Relations*. Holt, Rinehart, and Winston, 1991.

[13] J. Knight. *Fundamentals of Dependable Computing for Software Engineers*. CRC Press, 2012.

[14] M. Kropp and A. Meier. Teaching agile software development at university level: Values, management, and craftsmanship. In *Software Engineering Education and Training (CSEE&T), 2013 IEEE 26th Conference on*, pages 179–188. IEEE, 2013.

[15] P. Maher. Weaving agile software development techniques into a traditional computer science curriculum. In *Information Technology: New Generations, 2009. ITNG'09. Sixth International Conference on*, pages 1687–1688. IEEE, 2009.

[16] O. Meerbaum-Salant and O. Hazzan. An agile constructionist mentoring methodology for software projects in the high school. *ACM Transactions on Computing Education*, 9(4):n4, 2010.

[17] A. Messina, F. Fiore, M. Ruggiero, P. Ciancarini, and D. Russo. A new agile paradigm for mission-critical software development. *CrossTalk*, 29(6):25–30, 2016.

[18] M. Missiroli, D. Russo, and P. Ciancarini. Learning agile software development in high school: an investigation. In *Proceedings of the 38th International Conference on Software Engineering Companion*, pages 293–302. ACM, 2016.

[19] M. Missiroli, D. Russo, and P. Ciancarini. Una didattica agile per la programmazione. *Mondo Digitale*, 15(64), 2016.

[20] S. Papert. *Constructionism: A new opportunity for elementary science education*. Massachusetts Institute of Technology, Media Laboratory, Epistemology and Learning Group, 1986.

[21] R. Romeike and T. Göttel. Agile projects in high school computing education: emphasizing a learners' perspective. In *Proceedings of the 7th Workshop in Primary and Secondary Computing Education*, pages 48–57. ACM, 2012.

[22] P. Rundle and R. Dewar. Using return on investment to compare agile and plan-driven practices in undergraduate group projects. In *Proceedings of the 28th international conference on Software engineering*, pages 649–654. ACM, 2006.

[23] D. Russo. Benefits of open source software in defense environments. In *Proceedings of 4th International Conference in Software Engineering for Defence Applications*, volume 422, pages 123–131. Springer, Advances in Intelligent Systems and Computing, 2016.

[24] W. R. Shadish, T. D. Cook, and D. T. Campbell. *Experimental and quasi-experimental designs for generalized causal inference*. Wadsworth Cengage learning, 2002.

[25] J.-P. Steghöfer, E. Knauss, E. Alégroth, I. Hammouda, H. Burden, and M. Ericsson. Teaching Agile: addressing the conflict between project delivery and application of Agile methods. In *Proceedings of the 38th International Conference on Software Engineering Companion*, pages 303–312. ACM, 2016.

[26] B. Tessem. Individual empowerment of agile and non-agile software developers in small teams. *Information and software technology*, 56(8):873–889, 2014.

[27] The Standish Group. Chaos manifesto 2013, 2013.

[28] D. Turk, R. France, and B. Rumpe. Limitations of agile software processes. *arXiv preprint arXiv:1409.6600*, 2014.

[29] A. Vihavainen, M. Paksula, and M. Luukkainen. Extreme apprenticeship method in teaching programming for beginners. In *Proceedings of the 42nd ACM technical symposium on Computer science education*, pages 93–98. ACM, 2011.

[30] B. J. Wadsworth. *Piaget's theory of cognitive and affective development: Foundations of constructivism*. Longman Publishing, 1996.

[31] D. West et al. Water-scrum-fall is the reality of agile for most organizations today. *Forrester Research*, 2011.

[32] C. Wohlin, P. Runeson, M. Host, and M. Ohlsson. Experimentation in software engineering, 2000.

[33] A. Yau and C. Murphy. Is a rigorous agile methodology the best development strategy for small scale tech startups?, 2013.

[34] L. Zhang, S. Akifuji, K. Kawai, and T. Morioka. Comparison between test driven development and waterfall development in a small-scale project. In *International Conference on Extreme Programming and Agile Processes in Software Engineering*, pages 211–212. Springer, 2006.