2016 IEEE/ACM 38th IEEE International Conference on Software Engineering Companion

Learning Agile Software Development in High School: an Investigation

Marcello Missiroli
DIEF, University of Modena and Reggio Emilia
Via Vivarelli, 10
Modena, Italy
marcello.missiroli@unimore.it

Daniel Russo
DISI, University of Bologna
Mura Anteo Zamboni, 7
Bologna, Italy
daniel.russo@unibo.it

Paolo Ciancarini
DISI, University of Bologna
Mura Anteo Zamboni, 7
Bologna, Italy
paolo.ciancarini@unibo.it

ABSTRACT

Context: Empirical investigations regarding using Agile programming methodologies in high schools are scarce in the literature.
Objective: This paper evaluates (i) the performance, (ii) the code quality, and (iii) the satisfaction of both students and teachers in using Agile practices in education.
Method: this study includes an experiment, administered in a laboratory controlled setting to measure students' performances and a case study to value the code quality improvements. Questionnaires were also used to evaluate qualitative aspects of Agile practices.
Results: groups of students with mixed skills performed significantly better than groups with the same skill level. Moreover, there was also a general increase in code quality along with satisfaction.
Conclusions: Agile methodologies are useful in the High School education of young software developers.

CCS Concepts

•**Social and professional topics** → **K-12 education;** *Software engineering education;* •**Software and its engineering** → Object oriented development;

Keywords

Agile; Pair Programming; High School Education

1. MOTIVATION

The early days of software development were based on plan-driven methodologies, among which the waterfall model is the most famous. As a reaction to the problems that still exist with plan-driven methods, the Agile movement emerged, offering an alternative approach to software project development [10]. Agile methods offer a compromise between not enough process and too much process, being adaptive rather than predictive and, accordingly, they welcome change, which is an integral part of any software development process.

Agile methods are also people-oriented rather than process-oriented. The Agile movement identifies several general principles, values and practices, that are coded in various software development methods, such as Scrum, FDD, XP. In this work, we focus on some of the well-defined practices described in Extreme Programming (XP) [3], which can be applied to a school environment - in particular, we concentrate on Pair Programming, Time Boxing, User Stories Development and Test-First Development.

During the last fifteen years, Agile methods and practices have been gaining popularity in the software, industrial and academic worlds. The Agile movement prompted a move from monolithic, formal, and individualistic methods to a flexible, relaxed, and social one. Though Agile methodologies are not yet widespread, studies show that they bring distinctive advantages [1], especially in case of small projects [22] and when practices are correctly executed [6]. Thus, they are used in a wide range of environments, also Mission & Security ones [20].

However, Agile is not taught as part of standard programming syllabi in most universities, although it is sometimes introduced - but not tested first-hand - in high level graduate courses. Most of the times it is learned "on the field", often after attending expensive seminars. In particular, teaching Agile methods in high school is almost unheard of. If you ask Agile experts about this, they will probably reply that Agile is "difficult" and "requires already skilled programmers". But, probably, the real reason is that no other way has been proposed or tested yet.

In fact, it seems irrational to unlearn traditional programming techniques to learn Agile ones. In this paper we ask: if Agile is the correct way to develop software, why not learn it *right* from the start? Can learning Agile practices bring significant advantages to our programmers-to-be, maybe even by teaching programming in a different, more productive and fun way?

We have attempted to address these questions. We show the results of an investigation conducted on very young programmers in a high school context. Focusing on Agile *practices*, we considered several diverse factors, such as grades received, static analysis of the code, and psychological factors in both teachers and students.

While far from decisive, results show that, with some adjustments, Agile deserves a place in High-School CS education.

ICSE '16 Companion, May 14 - 22, 2016, Austin, TX,USA

DOI: http://dx.doi.org/10.1145/2889160.2889180

This paper has the following structure: section 2 summarizes the context of the schools we targeted, outlining the main information on courses where software development is taught, and the related literature; then section 3 describes the experiments we set up with students of those high schools; in section 4 we describe which analysis tasks we perform on the code developed by the students in order to evaluate the impact of Agile practices; in section 5 we describe how the subject options on Agile were organized and collected; then in section 6 we analyze the statistical significance of positive or negative performances of students; in section 7 we show the results of the administered survey and interviews; finally, in section 8 we draw our conclusions.

2. RELATED WORK

Even though Pair Programming is a widely studied methodology in software engineering, little was done to investigate its application in K-12 environments. Several studies have discussed the effectiveness of Agile practices in an industrial environment (one of the most thorough being [8]) and in an academic environment [13, 9]. Very few studies concentrate on Agile practices in High School, excluding rare European cases such as the application of the Eduscrum method [7], Extreme apprenticeship [23] and a German research group [19]. None of these, however, offer rigorous quantitative analysis of the results.

Educational science has made tremendous progress in the last century. Thanks to illustrious practitioners and thinkers, such as Montessori [16], Papert [17], Kolb [12], we changed our perspective on students from a standardized "vessel to be filled with knowledge" to that of a unique, social and active pursuer of knowledge; the focus shifted from the teacher, once absolute ruler and custodian of knowledge, to the learner.

Cooperative learning, in particular, is one of the most famous teaching strategies proposed by Johnson and Johnson in [18], in which small teams of individuals, each with different ability levels, use a variety of learning activities to improve their understanding of a subject. The model identifies five essential elements: positive interdependence, promotive interaction, individual accountability, group processing, and social skills. The model led to more formalized techniques (such as STAD, Jigsaw) that have been successfully applied to several subjects, from Chemistry to Foreign Languages [5, 2]. Another interesting and related strategy is **project-based learning** [15, 25] (PBL); with it, students actively explore real-world, complex problems and acquire deeper knowledge by actually creating a product.

Turning the attention to computer science, and formal programming teaching in particular, we notice that these teaching strategies are rarely used. Programming is most of the time considered an individual activity, and the waterfall development model is often the only development strategy taught in schools and even universities [13].

We found a lack of didactic awareness in the computer science community. Thus, with this paper, we want to explore those aspects. In particular how constructivism, PBL and cooperative learning fit in with high school programming education, since software production is, after all, the main goal of CS, and Cooperative learning can easily be linked to Agile principles (e.g., face-to-face communication, motivated individuals, self-organizing teams) and practices (e.g., Pair Programming, Common code ownership, Retrospectives).

Our work aims to be a first attempt to study the practical effects of such practices on high school students.

3. EXPERIMENT

In this section, we describe in details the experiment, which is the basis of our research. The goals of the experiment are outlined below, as research question 1:

Object of study. The object of the study is the outcome of a programming project of varying size.
Purpose. The purpose is to evaluate the impact of Agile practices in a high school 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 with at least one year of programming experience.

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

We follow the experiment methodology proposed by Wohlin, et al. [24]: context selection, hypotheses formulation, variables selection, selection of subjects, experiment design, instrumentation, and validity evaluation.

3.1 Context selection

The context of the experiment was the students of the last two years of Computer Science course in Italian high schools featuring CS teaching.

The Italian school system has only recently reached a stable configuration after its 2011 reform. Our discussion focuses on the current situation. In contrast to many other developed countries, computer programming is *not* widely taught in Italian schools, especially in secondary schools.

In fact, when a teenager enrolls in secondary school, she cannot freely choose subjects: these are automatically determined by the type of school she has chosen according to national laws. This strongly centralized organization of disciplines restricts proper teaching of computer programming and software development to the following types of schools: the Applied Science Gymnasium (2 CS hours/week, for five years), Industrial Technical schools (3-8 hours/week, three years) and some Commercial Schools (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).

Classes were randomly selected within the population of three Italian provinces. For the experiment in Commercial Schools we set Test Case 2 in one class from the 4th year. For the experiment in Technical Schools we set Test Case 1 in two classes from the 4th year and Test Case 1 in one class from the 5th year.

3.2 Hypothesis formulation

The crucial aspect of the experiments is to know and formally state what we intend to evaluate. These are the quantifiable hypotheses to be tested:

- $H0_{CD/TDD/US/SP/PP}$: There is no difference in the mean value of Product Evaluation (PE) between the software development projects using any combination

Table 1: Experiment context - groups coded by color and type

Subjects	**ALL**	Pair Total	Solo Total	GreenP (G-G)	CyanP (G-A)	YellowP (G-P)	BlueP (A-A)	MagentaP (A-P)	RedS (P-P)	GreenS (G)	BlueS (A)	RedS (P)
– 4th year	61	50	11	8	10	6	8	10	8	4	3	3
– 5th year	23	18	5	4	4	4	2	0	4	2	1	2
–Test Case 1	44	36	8	6	8	4	6	8	6	3	2	2
–Test Case 2	40	32	8	6	6	6	4	2	6	3	2	3

of Classic Development/TDD (test-first)/US (User Stories) and Solo/Pair programming development method (CD, TDD, US, SP and PP are used to denote development methods).

- $HA_{CD/TDD/US/SP/PP}$: There is a difference in the mean value of PE between the software development projects using any combination of CD/TDD/US and Solo/Pair programming development method.

The one metric to be used in the experiment is Product Evaluation (PE). In an industrial context, this is essentially the outcome of an acceptance test. When we move to a school context, the teacher becomes the "customer", thus the grade is a direct measure of teacher satisfaction. 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, which means that the current average grade becomes an excellent estimate of how they will perform in similar situations. In our test we wanted, however, to minimize subjective variation to the minimum, so we devised a system that almost directly translates the amount of completed features to a grade value.

3.3 Selection of subjects

The entire classe took part in the experiment. To limit teacher assessment and class level differences, we grouped students in three categories, with the respect to their "Expected Performance" (EP), essentially the average of all previous grades the student had at the time of the experiment. We coded these groups using the three primary colors of the RGB color coding system:

"Good" (GREEN). These are students whose average grade is 7 or more in the Italian ten-point grading system, though in practice only grades 7 to 8 are used - grades above 8 being extremely rare. This roughly equates to US B grade and above, or 72% or more). The expected average performance will be 7.5 or 75% of the maximum score.

"Average" (BLUE). These are students that are considered "more or less sufficient", or around 6 in the Italian grading system (roughly comparable to B- to C grade, or around the 70-55%). The expected performance will be about 60% of the maximum score.

"Poor" (RED). These are students whose average grade is 5.5 or less in the Italian ten-point grading system, though in practice only grades 4 to 5.5 are used - grades below being 4 extremely rare. This roughly equates to grades C- and below, or 50% or less). The expected performance will be 50% of the maximum score.

Note that in Italy the phenomenon of grade inflation [26] is rather uncommon; as a result, grade distribution is generally a normal distribution centered on the more-than-sufficient grade (6.5 out of 10), giving Italian teachers a reputation of being "stingy".

Using a random-stratified selection, we then made up the pairs by combining all possible skill levels, resulting in 6 possible pair types - mixed groups acquire the code obtained by mixing their respective color and group type initial (Good+Average: CyanP; Good+Poor: YellowP; Average+Poor: MagentaP); three possible solo types (GreenS, BlueS, RedS), that will become the control groups were also determined. Table 1 shows the full coding used.

We can now estimate the EP of heterogeneous pairs by averaging the expected performance of group members. For example, the expected performance of the "CyanP" group would be 6.75 - average of 7.5 and 6. To better control our independent variable, the experimenters did *not* announce the grouping criteria. This because groups needed to be tested with the lowest internal bias with respect to their performance. We accordingly assigned a codename for each group/solo type.

3.4 Design

Designing the experiment as a hands-on programming experience is not easy. One problem is the varying degree of skills and interest of the students, forcing us to choose an average-difficulty task, but also considering how to reward exceptional performances. The most limiting factor is, however, time. School class schedules are usually very rigid, imposing constraints and an overall time-limit, lab use and instruction, often imposing substantial bureaucratic work. In addition, the educational curriculum of each class requires that the experiment be carried out in a specific part of the year, when students already have sufficient knowledge and experience to successfully conclude the experiment. Therefore, we are limited to but *one* experiment session per class per year. So, we used a single installment to test multiple Agile practices at the same time.

Given the above limitations, we designed two possible experiment formats, a short version and a long version.

Test Case 1. Test Case one is the short form of the experiment, and it is designed to test Test-First Development, Timeboxing, and Pair Programming, focusing on basic, desktop development. We wanted the test to be straightforward, with no complex algorithm, since time requirements were very strict. The prerequisites for the experiment are knowledge of an object-oriented programming language(in our case, Java) and experience with an advanced IDE with automated testing support (in our case, Netbeans). Previous knowledge of the testing framework (JUnit) or Agile practices was neither expected nor required.

Both prerequisites are part of the standard curriculum for all students of the 4th year of the target schools.

We modeled our test on the *Tennis Game Kata*: the final product is a software object able to convert basic tennis

"points" to human-readable "scores", and to determine the game winner. We then devised a sequence of JUnit tests that the students used both as a guide and as acceptance test; the subjects were forced to follow the sequence of 8 JUnit tests exactly. To that, we added a couple of "forced pauses" to allow refactoring time.

The final performance score was mostly determined by the number of the test, resulting in a score from 0 to 8, to which 1 was added. We reserved one bonus point to be added at the teacher's discretion, to reward exceptional performance or situations; the final range is therefore 1 to 10.

Test Case 2. The longer version of the test is in fact a mini-project, as the students are asked to build from scratch a complete web application - more specifically, a microblogging web platform. Significantly greater knowledge was required than in the previous test case; test subjects needed to know a web programming language, in particular HTML and a server-side programming language; experience with web servers, including FTP operations in case of remote hosting providers; finally, basic experience with databases and their interaction with web applications. Again, no previous knowledge of Agile practices was assumed.

All the above competences are part of the standard educational curricula and, although the experiment can be performed on a variety of platforms and languages, we chose the W/LAMP platform (Windows or Linux / Apache / Mysql / PHP), which was taught in both target classes.

We then wrote a series of User Stories that formed the "backlog" of the project. The acceptance conditions were to be tested on a running system, either local or remote, and not by looking at the code; though not completely automatic, this reduced the human factor involved in the acceptance tests. Some user stories were substantially more complicated than the others, so we assigned 2 story points to each of them, and 1 point to the others.

The final score varied from 0 to 8, to which 1 was added. Again, we reserved one extra point for out-of-the-ordinary performance (decided by the teacher), to obtain the usual 1-10 range.

3.5 Planning - Instrumentation

The instrumentation involved the computer science labs of the schools involved (about 20 networked workstations), the Netbeans IDE project framework, the requirement specification (JUnit tests or user stories).

3.6 Validity

Concerning the validity explanation of our experiment, we are aware of statistical conclusion, internal, construct and external validity threats [21].

Threats to statistical conclusion validity 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 the research subjects, with respect to their previous performance and the treatment performance, were designed to maximize internal validity. Since the major aim of the experiment was to determine causation, the greatest attention was paidto this aspect. Test subjects were evaluated according to their past school performance (highly reliable due to the relevance of the time series i.e., grades given in a relevant time frame) and the experiment 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. Since our primary goal is to validate causation (our hypothesis), external validity, i.e., the capability to generalize the conclusions is considered adequate. Our sample was randomly defined, to maximize internal validity. Thus we considered 84 students of three Italian provinces, choosing different school types and different age groups. As we know, there is a constant trade-off between internal and external validity and our sampling strategy took this into account [11].

It was concluded that the threats were not regarded as being critical. Experiment materials and anonymized data are publicly available at https://bitbucket.org/marcello_missiroli/agileschool_data.

3.7 Operation

The experiments were conducted from December 2014 to May 2015; in total, 84 students were tested in the four selected classes.

Test Case 1. Since the Kata is relatively well-known and easily found on the Internet, we decided to block Internet access during the test, in order to avoid so-called *Google-and-Paste*-style programming. After a short explanation of the task, each group was given the first (and only the first) test of the suite. We let the students work freely, and only after checking that the current test, and all previous ones, were green, were they allowed to proceed to the next step - each step was freely available, but password-protected. When they reached steps 6 and 10 they were forced to take a minimum 3-minute stop to refactor the code. The final, optional step, was conceived as a way to keep better coders busy whilst the others completed their task and possibly reward them with the bonus point. Every 15 minutes we forced pair teams to exchange roles, as dictated by Pair Programming rules; after two hours we had a longer 10' rest - which in fact coincided with school recess. The most significant problem was that someone had problems with JUnit, resulting in "green" tests even though they were not; that required "backtracking" some steps for the groups involved. In one case we found out that the students inadvertently "cheated" by modifying the JUnit code to pass the test, but the case was dismissed with no consequences. The allotted time of two hours was considered adequate for all involved.

Test Case 2. In this case, we had to grant Internet access to all students, since some of them had to access remote hosting services; moreover, this reflected the idea that this test offered far more developing possibilities than the other one, and we felt that using the Internet as a resource was part of all-round programming competence. Unfortunately, one student was missing and another one showed up late for

Figure 1: Test Case 2 in action (Dec 01, 2014)

the experiment; we were forced to reorganize pairs on the fly, resulting in a sub-optimal Pair/Solo distribution. After the 20-minute long instruction session, each group was given a printed copy of the user stories, and they could pick the one they wanted - sadly, due to space limitations, it was not possible to have real Taskboards to work with. As in the previous case, strict adherence to Pair Programming was enforced with a timer. When a story was completed, the group returned the card to the examiners (one of us and the class teacher), who checked the acceptance conditions. If the story was accepted, the team scored the number of "points" stated on the card and could proceed with the next story; otherwise, they had to go back and revise the code. Working on multiple user stories at the same time was prohibited; talking with colleagues was neither encouraged nor forbidden. The allotted time of 5 and half hours was considered adequate for all involved.

After the time expired, the "score cards" for each group were collected and filed under the appropriate label (GreenP, YellowP and so on).

4. CASE STUDY

In this section, we wanted to analyze the code produced by the test subjects, in order to determine whether Agile practices have any effect on the quality of code; in particular, as Agile values maintainability, we expected that code developed in this way to be more maintainable, simple and efficient. We ran a Case Study since the code baseline was gathered by a similar exercise that was not administered in an experimental setting by the students' teacher. Moreover, teacher evaluations were not considered completely homogenous among the selected subjects. So, in order not to not invalidate the experiment, we chose to run this as a Case Study since we needed a more open and flexible research tool to cope "with the complex and dynamic characteristics of real world phenomenon." [24].

We then formulated research question 2 as follows:

Object of study. The objects of study is the code of a programming project.
Purpose. The purpose is to evaluate the impact of agile practices in a high school environment.
Quality focus. The quality focus is on the students' code.
Perspective. The perspective is the teachers' perpective.
Context. The research is conducted by engaging high school students with at least one year of programming experience.

Summary: *The analysis of the code for the purpose of evaluation of the impact of Agile practices as a teaching method in high school with static code analysis tools from the point of view of the teacher in the context of basic and complex software development by the students.*

Once more, we followe the experiment methodology proposed in [24]: design, selection criteria, data collection, validity, plan validity.

4.1 Design

Defining and measuring software quality is very difficult in general [4], and especially so in case of elementary algorithms within small-scale school projects. Since there are several metrics that can be applied to software, we decided to separately analyze five "classical" code metrics, including the overall quality indicator SQALE [14]. To summarize, the metrics we used were:
CCOMP: Cyclomatic complexity per source code file
CL: Commented lines of code produced by the group
DL: Duplicated lines of code
NCLOC: Total number of non-commenting lines of code
ISS: Number of programming issues
SQALE: A holistic quality indicator

One advantage of these metrics is that they can be calculated by using appropriate tools on the source code, producing data that can analyzed at leisure later.

4.2 Selection Criteria

The Italian school context is the one described in 3.3. To conduct our code analysis, we need two codebases: one produced with standard methodologies (**baseline codebase**), and the other with agile techniques (**agile codebase**). Of course, we intend to use the material produced during our experiment as the agile codebase, since it uses a combination of one or more agile practices; moreover, it is partitioned according to subject skills and produced in a very controlled environment.To acquire the baseline codebase, with the help of the class teacher, we selected a suitable, recently executed lab exercise with characteristics similar to those of the experiment.

4.3 Data Collection

Collecting the Agile codebase was easy, but organizing it was not straightforward. For example, some groups refused to turn in the code, even though they had made some advancements - as certified by the teacher. As this was significant, we decided to add a special record to the data that took into account the "failure" of the pair/individual. In Test Case 1, most groups submitted two or more files, one of which being the boilerplate code for the ancillary object. This file was either empty or with a useless `main()` method, so it was excluded from calculations. In rare cases, the students wrote an additional class: a GUI interface, a secondary Object, a derived class. Since they were all pertinent to the problem goal, we decided to include them in calculations.

Collecting the baseline codebase was more difficult, since different teachers had different filing methods. In one case, the project was archived on single LibreOffice document, so we had to extract the single files by hand. We also had the problem that 2/3 of the Agile codebase was the product of a pair, but the previous codebase was produced by single individuals. We overcame this problem with the help of the teacher, by creating a fictional project containing the code of the same individuals that made up the pair, and analyzing the code as a single product.

We had to select an analyzing tool for our codebases, and we chose Sonarqube [www.sonarqube.org], because it supported both Java and PHP, it was easy to install and use, and it supported all the metrics we were interested in. In addition, it was free and open source software.

4.4 Plan Validity

We understand that plan validity has four aspects: Construct Validity, Internal Validity, External Validity, and Reliability [21].

Construct Validity is under control. We used several, well established and deterministic metrics to evaluate the code, and we considered them separately in our analysis.

Threats to Internal Validity depend on how much the tasks that originated the codebases were similar, the conditions under which the coding experiment was executed, and how the investigated metrics might be affected by some uncontrolled factor. The cooperation with the class teacher assures that both experiments were indeed similar and executed in similar psychological and environmental conditions. The fact that more than one Agile practice was tested at the same time might indeed be a threat to internal validity, but the impact would be limited. Hence, also this aspect was under control.

External validity was under control. The population was drawn from different types of schools, courses and cities; in addition, the target classes were, in the opinion of teachers, of average level.

Reliability was under control. Programming tasks were based on standard curricula, all code metrics were well-established and the tools used were open and free software.

Both codebases and raw analysis results are available at https://bitbucket.org/marcello_missiroli/agileschool_data.

5. SURVEY

As a final element of our investigation, we discuss if the experiment was *deemed* useful in terms of learning experience and if it could be used not only as a one-off event but as a regular practice in the daily teaching routine.

We formalize the research question 3 as follows:

Object of study. The object of study is the learning environment actors: students and teachers.

Purpose. The purpose is to evaluate the impact of Agile practices in a high school environment.

Quality focus. The quality focus is the perceived utility of Agile practices.

Perspective. The perspectives are the teachers' and the students' ones.

Context. The research is conducted on high school students with at least one year of programming experience and on all CS teachers in the related provinces.

Summary: *The analysis of the opinions for the purpose of evaluation of Agile practices impact as a teaching tool on high schools with respect of perceived usefulness from the point of view of teachers and students in the context of programming projects.*

5.1 Research Participants and Data Gathering Tools

First, we designed an **Experiment Feedback** questionnaire for all students who participated in the experiment, aiming at determining how the new methodologies were received at a psychological level. In particular, we wanted to determine if the programming experience was pleasant and useful. In almost all questions we used a standard 5-level Likert-scale.

We then organized a **Reflective Interview** with the teachers involved in the experiment, consisting of a series of focused questions. The goal was to determine if the experience was successful, and if the teachers foresaw Agile practices as a sustainable, day-to-day practice for CS teaching.

Finally, as we wanted also to have an idea of the general disposition of CS-high school teachers towards Agile, we prepared an **Agile Awareness Survey** to be submitted to all CS-teachers in the three selected target provinces, a group of an estimated size of 110. The goals were to determine what a teacher knows about Agile programming and related techniques and if they are interested in trying them outwith their classes. In almost all questions we used a standard 5-level Likert-scale.

5.2 Research Process

The Experiment Feedback questionnaire was proposed to the students after the experiment was over by using Google Forms, stating that it should be completed within the week. The response turnout was surprisingly high, around 70%, especially given that there was no school-related incentive to do so.

The Reflective Interview was performed much later, at the end of the school year, when it was possible to determine if the experience could have had an impact on the final grade achieved by the students.

Conducting the Agile Awareness Survey presented a major problem, in that a listing of all computer science teacher of the selected provinces is not publicly available. The coordinating "Ufficio Scolastico Regionale" (Regional School Board) proved to be uncooperative, refusing both to produce an email listing of the teachers and to forward the form-filling request to them. Therefore, we had to send the request to the school principals, asking them to forward it to the teachers - this has sadly been proven to be a really inefficient means of communication. Unsurprisingly, the turnout rate was very low, about 20% of the intended population. Questionnaires, driving questions and anonymized results are also publicly available at https://bitbucket.org/marcello_missiroli/agileschool_data.

6. ANALYSIS

In this section, we show and analyze the results of our Experiment and Case Study.

6.1 Experiment

Experimental data were evaluated with descriptive analysis and statistical tests. What we wanted to understand is if there was a significant change (positive or negative) of the students' performance using Pair Programming as an educational programming methodology. So, we clustered all our experiments by levels; descriptive statistics are summarized in Table 2. In rows we tested our sample for each pair level group for the most significant descriptive value (i.e., Mean, Standard Error, Median, Mode, Standard Deviation, Kurtosis, Skewness, Interval, Minimum, Maximum, number of observations (OBS), Confidence level (CL) at 95%). We considered the results of similar groups which run the experiment and analyzed their clustered performance.

Table 2: Descriptive Statistics

Performance	BlueP	MagentaP	CyanP	GreenP	YellowP	RedP	Solos
Mean	4	5	8.14	7.5	7.4	5	5.7
Standard Error	0.84	1.05	0.59	0.85	0.4	0.95	0.67
Median	5	4	9	7.5	7	5	6
Mode	2	4	9	7	7	5	5
Standard Deviation	1.87	2.35	1.57	2.07	0.89	2.12	2.6
Kurtosis	-2.90	3.32	-1.16	1.11	5	2	-0.58
Skewness	-0.38	1.74	-0.68	-0.81	2.24	0	-0.52
Interval	4	6	4	6	2	6	8
Min	2	3	6	4	7	2	1
Max	6	9	10	10	9	8	9
OBS #	5	5	7	6	5	5	15
CL (95,0%)	2.32	2.91	1.46	2.18	1.11	2.63	1.44

Table 3: Hypothesis Analysis

	BlueP		MagentaP		CyanP		GreenP		YellowP		RedP		Solos	
	P	EP	P	EP	P	EP	P	EP	P	EP	P	EP	P	EP
Mean	4	6	5	5.5	8.14	6.5	7.5	7	7.4	6	5	5	5.7	6.2
Variance	3.5	0	5.5	0	2.48	0	4.3	0	0.8	0	4.5	0	6.8	1.2
# Observations	5	5	5	5	7	7	6	6	5	5	5	5	15	15
df	4		4		6		5		4		4		14	
t Stat	-2.39		-0.48		2.76		0.59		3.5		0		-0.92	
P(T<=t) two tails	0.07		0.66		0.03		0.58		0.02		1		0.37	
t Critical two tails	2.78		2.78		2.45		2.57		2.78		2.78		2.14	

No clear trend emerged, so we had to perform additional analyses of hypothesis testing.

6.1.1 Hypothesis Testing

Experimental data were analyzed using models that relate the dependent variable to the factor under consideration. More in detail, we ran our analysis to compare the performed task with the expected result. The performance (P) was the 1-10 grade obtained by the pair (using the procedure described in 3.4). The expected performance (EP) was the mean of both team members according to their past marks (as described in 3.3).

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. The two samples were thus *not* independent of one another.

Table 3 shows the aggregated t Test for each group. Interestingly, only the CyanP (Good-Average) and YellowP (Good-Poor) groups have a significant difference, according to their expected performance. So, for these both cases 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 in both cases the means of their performance is larger than their expected one, we conclude that for the CyanP and YellowP case, groups using the Pair Programming methodology perform significantly better.

For all other cases, we did not see any statistical significance, since the t Stat was smaller than the t critical.

It should be pointed out that there was a significative performance boost in cases of heterogenous programming pairs programmers consisting of at least one skilled programmer ("CyanP" and "YellowP" groups). In contrast, the "BlueP" group performed significantly worse than expected, even if this is not statistically significant. We can generalize the result by saying that **heterogenous groups** perform **equal or better** that expected, whereas **homogenous groups** perform **equal or worse** than expected.

While this is in line with constructionist group formation theory and practices [18, 25], our interpretation of this fact is that Agile programming techniques, and PP in particular, foster interaction and knowledge sharing. This has a beneficial effect in cases where there is an effective knowledge gap between the team members, and particularly when one member is - even subconsciously - acknowledged as authoritative. In cases where one of these elements is missing, the advantages of Agile practices are undecided. The case of the poor Average-Average groups performance only confirms this hypothesis - we even observed one case of clear bickering.

6.2 Code Analysis

Since we were working with two different programming languages relating to different project types, we kept the result of the analysis separated (as shown in Tables 4 and 5). We then examined the different metrics, one by one.

Cyclomatic Complexity average/file - CC. For Test Case 1 (TDD), we have an average Cyclomatic Complexity value of 18.99, a distinct *increase* with respect to the codebase, in all skill groups and pairings. The fact is incontradiction with the Agile principle of Simplicity, which

Table 4: Code Analysis - Test Case 1 (Java)

Group	CC	DIFF	CL	DIFF	DL	DIFF	ISS	DIFF	NCLOC	DIFF	SQALE DIFF
GreenP	22.5	181%	10.8	169%	0.0	-100%	16.0	140%	141.3	275%	Better
CyanP	21.0	163%	8.2	85%	13.8	25%	7.5	1%	169.0	158%	Equal
YellowP	25.0	213%	21.5	1333%	0.0	NA	12.5	150%	59.0	81%	Worse
BlueP	22.5	181%	1.0	-69%	0.0	-100%	8.7	28%	88.0	130%	Better
YellowP	18.8	134%	1.8	-56%	0.0	NA	6.0	20%	81.5	110%	Equal
RedP	20.0	150%	3.5	250%	11.3	NA	7.3	-22%	67.7	122%	Worse
GreenS	15.8	96%	30.3	278%	0.0	-100%	9.7	20%	84.3	122%	Worse
BlueS	9.0	13%	4.0	300%	0.0	-100%	12.5	150%	41.5	-16%	Much Worse
RedS	16.0	100%	5.0	0%	8.0	-56%	5.7	-51%	44.0	-1%	Worse
AVG	18.9	137%	9.5	254%	3.7	NA	9.5	49%	86.3	109%	Worse

Table 5: Code Analysis - Test Case 2 (PHP)

Group	CC	DIFF	CL	DIFF	DL	DIFF	ISS	DIFF	NCLOC	DIFF	SQALE DIFF
GreenP	2.5	23%	0.1	-67%	30.3	-70%	39.0	-56%	188.0	7%	Equal
CyanP	3.5	69%	0.2	51%	90.3	-9%	47.7	-23%	160.7	-32%	Worse
YellowP	2.6	25%	1.1	342%	159.0	20%	88.3	-2%	256.3	15%	Better
BlueP	0.9	-55%	0.0	/	14.3	-66%	3.8	-95%	8.9	-93%	Equal
MagentaP	0.7	-65%	0.2	-85%	133.0	83%	35.0	-16%	30.0	-46%	Better
RedP	2.2	8%	0.0	/	76.5	-2%	35.5	-63%	74.5	-60%	Much Worse
GreenS	5.0	143%	0.0	/	67.5	-27%	83.5	258%	323.5	187%	Equal
BlueS	0.7	-65%	0.2	0%	29.0	-43%	21.5	-5%	16.0	-75%	Much Worse
RedS	1.7	-19%	0.0	/	143.0	240%	46.7	419%	181.0	298%	Worse
AVG	2.2	7%	0.2	4%	82.6	14%	44.6	46%	137.7	22%	Worse

was clearly shown and tested during instruction. We interpreted the fact by noticing that timeboxing constraints put relatively strong pressure on test subjects, who probably found the quickest, but not the simplest solution to the problem. Though refactoring time was given, the students had no incentives in effectively refining the code, as the only stated objective was to have the code work, i.e. pass the test. In Test Case 2 (US), we did notice a significant drop in Complexity; however, the average value was of just 2.2 indicating that the structure of the program was very easy to begin with, and Cyclomatic Complexity might not be a good metric in this case.

Comment Lines average/file - CL. Results show that Agile practices bring no advantage in this field. However, the most significant fact is that the CL/NCLOC ratio was terribly low in all cases considered. Commenting programs is a good habit that comes with experience, and for reasons similar to what has been discussed above, this test is not well suited to promote it.

Duplicated lines average/project - DL. Test Case 1 shows a very small number of duplicated lines (an average of 3 per project) with no statistical difference between the two codebases. On the other hand, Test Case 2 shows a marked reduction of duplicated lines, a tendency which is more pronounced in case of heterogenous pairs, and in case of pair programmers.

Issues/project - ISS. Though both test cases show an overall increase in programming issues of the same order (around +50%), in Test Case 2 we notice that pair programmers actually *reduce* the number of issues (-43%) whereas solo programmers display a strong increase (+223%)

Number of non-commenting lines of code total - NCLOC. Both test cases show a general increase in the number of lines produced. It might however be debatable whether this is an indication of either productivity or inefficiency. Given that failed projects were considered to contribute zero lines, we tend to value the first hypothesis.

SQALE index average/project. While it is complicated to compare sets of different size, we notice that in both cases there is a small decline in grades. However, pair programmers show overall no decrease in the ratings, whereas solo programmers do show a moderate increase.

It is difficult to extrapolate from this data possible advantages of Agile practices. Nevertheless, we notice that in the case of medium-sized school projects, there are some advantages in using agile practices in terms of duplicated lines and overall number of issues, especially when pair programming is used. In small-sized projects, however, Agile practices do not appear to be useful, and can even be detrimental to code quality. In any case, code refactoring should be handled using different instruction and testing methods.

7. SURVEY INTERPRETATION

7.1 Experiment Feedback

Feedback from students shows that Pair Programming is regarded as a useful practice (see Table 6), allowing both more code and fewer bugs. This point of view is shared even by the control group.

Opinions diverge on other aspects. In particular, students participating in PP feel they have learned something during the programming experience, where solo programmers do not. Pairs also noted that PP has been a positive experience, and, to an even larger degree, that the pair worked well in

this situation. The response is quite surprising, since the positive interaction of quasi-random working groups is not assured.

Other Agile techniques did not fare so well. As shown in Table 6, students show **no love for timeboxing** - it is the worst-rated question. students probably felt too pressed for the time; this is most unfortunate, since handling of the time variable is a decisive aspect in software development. User stories and acceptance conditions did not convince our subjects, either. TFD fared a little better, being acknowledged to be a means to enforce adherence to specification and produce correct code. But, overall, students were **unimpressed**.

One possible explanation is that, in order to have a one-day experiment, both user stories and tests were *imposed* on groups, whereas in a real context they are self-produced and agreed upon. This might also indicate that it is hard to fully perceive the advantage of Agile practices without some involvement in the analysis and design steps of software development - an issue that will be addressed in future experiments.

We also noted an interesting phenomenon: first, some students answered questions they were not supposed to (for example, they answered the TDD section, even though they took part in Test Case 2 that did not include TDD) - we had to correct the data taking this into account. In addition, many mistook the formalized "Pair programming" practice for the unstructured "Programming in couple on the same workstation", indicating that more precise tuition is required before proceeding with the practical part of the experiment.

7.2 Reflective Interviews

In general, all teachers agreed on the overall **beneficial** effect of the experience on **motivation and the learning environment**, for all skill levels. In particular, Pair Programming is considered to be a very useful technique, even as a tool to be used in the professional world.

"Everybody was happy to do pair programming. In fact, we had some grumbling from students selected for the control group."

They acknowledged the didactic value of Test Case 2 (longer experiment), but felt a little uncertain of that of Test Case 1 (shorter). The opinion of other practices was ambivalent. In particular, they thought that TFD had a bad influence on studente performance, as well as timeboxing restraints - to a lesser extent. User Stories and Acceptance condition fared a little better, but they had some reservations. One teacher noted:

"Some individuals were not interested in the experiment, and did not like the fact that task directives were strict and immutable; so they were not 100% committed. And the results show it."

However, when asked to consider the overall effect of the experiment on the course, including final grades, results were somewhat surprising. They perceived a slight overall improvement in grades, regardless of skill levels, type of test and pair/solo programming. This might suggest the possibility of using agile methodologies as one of many teacher tools in day-to-day teaching, as they pointed out:

"They are interesting methodologies to use from time to time. Not all the time, though, as they require considerable time and effort to be carried out."

7.3 Agile Awareness Survey

Even given the low (20%) turnout rate, teachers' responses deserve consideration, as they are clearly polarized. For instance, 58% of the teachers declared that they knew absolutely nothing about Agile methodologies, while the remaining teachers had only some theoretical knowledge (limited to XP, Agile RUP, TDD, PP, and timeboxing). Their stance on teaching using such techniques is somewhat puzzling: they strongly think that agile techniques would not contribute to the students' software development techniques (68% of responses), but are uncertain of their effectiveness as a teaching methodology; finally, most teachers (73.2%) declare they would like to learn Agile methodologies and how to apply them in a school context. Those few that have practiced Agile techniques declare that they had some positive effects on teaching.

We also note that 53% of the students programmed in pairs. In most cases, students were able to work on a school project, but only a few had the chance to develop more than a single project during a school year.

8. CONCLUSIONS AND FURTHER WORK

While not decisive, our investigation shows that there is a definite advantage to early exposure of inexperienced programmers to Agile, not only as a project development process, but also as a teaching tool. Pair programming is the obvious winner, bringing clear advantages in motivation and "classroom mood", code quality and, depending on pair composition, grades. Furthermore, since most studente are forced to program in pairs anyway, it is very easy to implement. User stories, tied to longer programming experience, offer less clear advantages, but are deemed useful by teachers. TDD and Timeboxing are more controversial, especially taking pupils' opinions into account. We believe that the young people involved felt the task requirements given to them to be too much "imposed from above", and teenagers do not like imposition, far less than undergraduates and professionals do. As a lesson learned, we should have included some elements of interaction and bargaining with the students before starting the experiment, building a more constructionist-oriented learning environment or, one might say, a more *Agile* approach, whilst trying to come to terms with the school's institutional constraints.

Considering the favorable response of most CS-teachers toward the object of our study, we are planning further investigation consisting in confirmatory experiments on the above results (with some due adjustments) and exploratory studies on other Agile techniques (such as refactoring, timeboxing decisions, collective code ownership, etc.).

9. REFERENCES

[1] S. Ambler. 2013 it project success rates survey results.

[2] K. H. Árnadóttir. Cooperative learning in foreign language teaching: A study of the use of group work in language studies in icelandic secondary schools. 2014.

[3] K. Beck. *Extreme programming explained: embrace change*. Addison-Wesley Professional, 2000.

[4] B. W. Boehm, J. R. Brown, and M. Lipow. Quantitative evaluation of software quality. In *Proceedings of the 2nd international conference on Software engineering*, pages 592–605. IEEE Computer Society Press, 1976.

Table 6: Experiment Feedback

Pair Programming	Results PP (AVG)	Results SP (AVG)	Test-First Development	Results (AVG)	User Stories & Timeboxing	Results (AVG)
PP has been a positive experience	Agree (2.71)	NA	*TFD allows for faster development times*	Undecided (2.20)	*US is a valid programming tool*	Disagree (1.18)
My pair worked well during the experience	Agree (2.98)	NA	*TFD enforces adherence to specifications*	Agree (2.50)	*Acceptance conditions are useful*	Disagree (1.25)
Solo programmers produce BETTER code	Disagree (1.48)	Disagree (1.39)	*TFD produces more correct code*	Agree (2.52)	*Timeboxing is a guiding tool, not a source of stress*	Disagree (1.12)
Solo programmers produce MORE code	Disagree (1.35)	Undecided (1.67)	*TFD allows me to write code faster*	Undecided (2.37)		
I learned something from this experience	Agree (2.76)	Undecided (1.88)	*I will use TFD in the future.*	Undecided (2.00)		
			TFD helps find bugs in my code	Undecided (2.32)		

[5] C. W. Bowen. A quantitative literature review of cooperative learning effects on high school and college chemistry achievement. *Journal of Chemical education*, 77(1):116, 2000.

[6] 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.

[7] T. de Jager. Using eduscrum to introduce project-like features in dutch secondary computer science education. 2015.

[8] 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.

[9] T. Flohr and T. Schneider. Lessons learned from an xp experiment with students: Test-first needs more teachings. In *Product-Focused Software Process Improvement*, pages 305–318. Springer, 2006.

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

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

[12] D. Kolb. Learning styles inventory. *The Power of the 2 2 Matrix*, page 267, 1985.

[13] 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.

[14] J.-L. Letouzey. The sqale method for evaluating technical debt. In *Proceedings of the Third International Workshop on Managing Technical Debt*, pages 31–36. IEEE Press, 2012.

[15] T. Markham. Project based learning a bridge just far enough. *Teacher Librarian*, 39(2):38, 2011.

[16] M. Montessori. *The montessori method*. Transaction Publishers, 2013.

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

[18] T. Roger and D. W. Johnson. Cooperative learning, 1994.

[19] 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.

[20] D. Russo. *Proceedings of 4th International Conference in Software Engineering for Defence Applications: SEDA 2015*, chapter Benefits of Open Source Software in Defense Environments, pages 123–131. Springer International Publishing, Cham, 2016.

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

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

[23] 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.

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

[25] E. Zecchi. Project based learning (pbl) activities using the "lepida scuola" method.

[26] R. L. Ziomek and J. C. Svec. High school grades and achievement: Evidence of grade inflation. act research report series 95-3. 1995.