Training students to choose their agile practices and tools

Paolo Ciancarini
Dept. of Computer Science
University of Bologna
Bologna, Italy
paolo.ciancarini@unibo.it

Andrea Loretti
Dept. of Computer Science
University of Bologna
Bologna, Italy
andrea.loretti2@studio.unibo.it

Marcello Missiroli
Dept. of Computer Science
University of Bologna
Bologna, Italy
marcello.missiroli@unibo.it

Andrea Schinoppi
Dept. of Computer Science
University of Bologna
Bologna, Italy
andrea.schinoppi@studio.unibo.it

***Abstract*—We present our experiences in training computer science students in agile software development over two academic years. The product to build was a specialized Twitter client, with instructors refining its requirements throughout the course. We observed over a hundred students divided into teams of approximately five members each. To support agile collaboration and self-tracking, we provided students with a comprehensive software development environment consisting solely of open-source tools. Before commencing their cooperation, we encouraged students to engage in team-building activities to foster better mutual understanding. We adopted the Essence approach to instill an agile mindset and facilitated retrospectives tailored to the students' needs. Key findings include the effective use of the adaptable Scrum framework with support from the Essence approach to agile teamwork. Although the constraint of using exclusively on-premises open-source software tools posed some challenges for the students, all produced data and documents were accessible for inspection by the instructors. Additionally, the utilization of a product quality model and team maturity model proved valuable for evaluating and comparing the teams. Notably, all teams successfully completed their tasks within the designated timeframe.**

***Index Terms*—agile, teamwork, Essence, agile tools**

I. Introduction

Agile methodologies are widely used in the software industry, where they are applied using a variety of processes and best practices. Agile methods and practices can also help students develop valuable soft skills such as communication, teamwork, and adaptability. Practicing agile, they should learn how to work effectively with a product owner and other developers, respond to changing requirements, and aiming at continuously improving their work processes. Although there are several agile methods, two are especially outstanding: Scrum [21] and Extreme Programming [3]. The former is the most popular, because it enforces the major Agile principles of iterative, value-based, incremental delivery by frequently gathering customer feedback and embracing requirements change. The latter is also well known, as it introduced a set of agile best practices that can be easily be applied and exploited by students, such as pair programming, the collective property of code, requirements as user stories, etc. An important value fostered by the agile vision is that the process is less important than individuals and interactions. We interpret this value suggesting the students that they can choose the best practices to use during the development. Planning and implementing their own Agile software process can also help students develop a sense of ownership and responsibility over their work. They will have a better understanding of what is expected of them, and they will be more motivated to achieve their goals. Overall, learning how to plan their own Agile software process and pick Agile best practices can be a valuable learning experience for students, preparing them for successful careers in software development and helping them develop important skills for any profession.

In this paper we report on our experiences with training to agile vision and selection of agile best practices our Computer science students in a software engineering class. We adopt the Essence approach [12] that is based on a visual language able to capture combinations of agile practices and the state of their enactment inside a development process. Essence is formally an international standard issued by OMG [18].

Learning how to plan their own Agile software process and choose which best practices apply can help students become more effective and efficient in their software development work. They will be able to break down complex projects into smaller, more manageable tasks and work collaboratively with their team members to complete these tasks in a timely manner. We use a team building game to help the students to select their role in a Scrum-like team. Then we ask them to use a number of state-of-the art open source tools, like Taiga, GitLab, and SonarQube, to help them during the development and to collect data useful for the analysis we will report in this paper. We also suggest them to follow the Essence approach to process enactment and retrospective analysis. Concerning the evaluation of this experience, we have developed a maturity model to analyze how student teams perform their teamwork. We observed more than one hundred students divided in teams averaging five members each.

Specifically, we observed 136 students (57 in 2021 and 79 in 2022) divided in 27 teams (11 in 2021 and 16 in 2022) with an average of 5 components each (three teams had six components and two teams had four).

The project consisted in creating a Twitter client enriched with a dashboard with several filtering and visual analytics features.

The teamwork was organized following a Scrum-like approach: each team had to work for at least three 3-week-sprints using CAS services and then write a final report to summarize the development process. Each student had then to complete an individual questionnaire to evaluate the experience.

We collected several data. The main data sources were the final reports, the individual questionnaires and, within the CAS environment, Gitlab, Taiga, and SonarQube. The final reports were a fundamental source of data since, in addition to containing all mandatory information, most teams included additional elements regarding their own process. Tools were shared with the instructors. Gitlab was used to analyze the structure of the teams' repositories and obtain data regarding the frequency of commits and the tests performed. Taiga was used by the teams to manage their sprints and their retrospectives (conducted using Essence cards); the instructors, as Product Owners, could see the progresses on the product and its documentation. SonarQube was used for extracting product quality data and technical debt data.

This paper has the following structure: Section II presents some related works; Section III overviews the approach we followed, describing the preliminary activities for team building, the product to be built by the teams, the process enacted, and the open source tools which were used. Section IV presents the results of the analysis of the data we collected; Section V discusses these results; finally, Section VI presents our conclusions and the work we are planning now.

II. Related works

There are several papers concerning teaching agile teamwork and development in a university course. For instance, in [9], the authors describe their experiences teaching agile software development to computer science students and provide some recommendations for instructors; in [15] we found the description of a course on agile methodologies taught to software engineering students analyzing their personalities and attitudes to teamwork; and in [17] there is a discussion of how the use of collaborative practices needs some maturity by the students. A paper exploiting Essence during an academic course project work is [13]; this paper discusses the difficulties of adopting Essence in the context of an undergraduate software engineering course. We searched for papers concerning team building in a university context: we found [14], which suggests an approach on how to integrate team building into university courses including agile teamwork.

We also have devoted some effort to look for papers exposing experiences on teaching agile using open source tools. The match between agile and open source seems natural, however we have found only the paper [23], which reports an experience using tools quite different from those we used, in particular they used Redmine for project management and bugzilla as issue tracker.

Documentation of process choices, tools selections, and their rationale is an important factor in software development projects in order to support product quality and future maintenance. While several research publications address this topic, systematic approaches and tools are rarely found in practice, and not well covered in software engineering education. Lack of suitable process documentation is especially an issue in agile software development, where processes and tools are often seen as less important than working products [20].

We found some experiences of using a Scrum-like approach adaptable by the students, see for instance [19; 2].

Concerning the evaluation of the product, our work has been inspired by the quality model described in the article by Hoegl and Gemuenden [11], and further developed by Lindsjørn et al. [16].

The maturity model for agile teamwork which inspired us to evaluate the process was proposed by Yin et al.[24], based on the one created by Chetankumar et al.[4]. Gren et al. have also discussed the concept of teamwork maturity in agile teams [10], and influenced our model as well.

III. Method

In this section we will present our research method. First, before the course we built and configured an experimental open source environment, inspired by Jira, to be used by the student teams and also useful to collect process data.

We asked the teams to start their project work with a team building activity, using the serious game Scrumble[1], aiming at improving the reciprocal knowledge and training themselves with a simulated form of Scrum cooperation. Fig.1 shows a selfie by a team after having played the game.

We produced also an online version of the game, so that the teams could play remotely during the pandemic.

[1]http://scrumble.pyxis-tech.com/

Figure 1. A team at the end of a game of Scrumble

After the teams completed and self-evaluated their team building we described them the product to build: a Twitter client with several capabilities for visual analytics, to be applied to specific situations: eg. an earthquake, to signal emergency situations, or a travel with some friends, to signal the positions of the participants during the visit to some city or area. We also asked the teams to follow a Scrum-like process, that could be adapted to their needs using the Essence approach. Essence allows to select and organize specific practices (eg. pair programming or retrospective with some special activities) still keeping Scrum as a reference framework. At the end of the project the teams had to produce a demo and a process report. The exam consisted in a product demo and a final retrospective conducted together with the instructors.

A. Product specifications

This section briefly illustrates the product specifications. The goal of the project was to create an application capable of gathering and organising tweets. The collection could be historical (e.g. tweets from last week) or gathered as a real time stream.

The application had to allow tweets visualization and consulting and, under certain conditions, activate specific procedures. The requirements for a Minimal Value Product were:

- Use a keyword to collect Tweets from the past;
- Collect geolocalized tweets;
- Given a geographical area, show tweets posted in that area;
- Gather tweets referring to a some Point of Interest;
- Given a specific keyword or hashtag, show related tweets;
- Show geolocalized tweets of a specific person, and follow its movements;
- (sentiment analysis) Analyze the sentiment of a series of tweets about a given topic;.

The tweets collected and classified had to be aggregated in an interactive dashboard, possibly showing coordinated views presenting different data details (e.g. locations on a map), a term cloud of the most used words, a bar chart with the number of tweets per unit of time, a pie chart with positive and negatives sentiments, plus other ways to aggregate data.

B. Process description

This section describes the software development process adopted by the teams during their work. We split the whole process in three main phases: preparatory phase, execution phase, and conclusive phase.

1) Preparatory phase: During this first phase teams were formed using Trello: the students created personal cards on the platform by inserting their nickname, a brief description of their skills and their programming preferences (for example: front-end development, back-end development, UX expert, or testing). An attempt to form teams that, while covering all the necessary roles during development, reflected as much as possible the skills and preferences of the individual components was made. Trello was used because was enough popular among students, and could be accessed using their smartphones.

After the team forming, the teams autonomously assigned the two roles of operational Product Owner (oPO) and Scrum Master (SM), while other members made up the developer team. The assignment of roles is a critical team building activity: to decide the roles, the teams were requested to play a game of Scrumble. This is a serious game in which a Scrum process is simulated, including also a concept of managing technical debt. A game of Scrumble can be played either in presence or online [5]. During each round of the game, the players took on different roles, such as Product Owner, Scrum Master, and Development Team members. The Product Owner was responsible for introducing the user stories and prioritize the work, while the Scrum Master was the coach explaining the Scrum process and ensuring that the team is following the rules. The retrospective performed at the end the game was based on a GQM evaluation which helped the team to choose which people could cover which roles in the real project.

The role of the operating PO inside the team proved to be useful to discuss with the real Product Owners (the instructors) variants of the product to build , possibly with enhancements introduced by the teams themselves.. The addition of the "operational" specification to the PO indicates that it is not an external member of the group, but a student with a double role: Product Owner and team support (and in most teams also developer). A similar observation can be made for the Scrum Masters who, in most cases, have not limited themselves only to providing support to their team but they have additionally taken on the role of developer.

2) Execution phase: Once teams were formed and roles assigned, the real code-development phase began: every team worked for at least three sprints lasting three weeks each.

Teams used Taiga to draw up a product backlog where each requirement was transformed into user stories (US) and tasks assigned to developers, whose code was then pushed on Gitlab and had to pass SonarQube's quality checks. User stories also had several acceptance criteria, including mandatory tests and an estimated difficulty level expressed in story points.

When a developer completed a US, its amount of story points was subtracted from the total allowing Taiga to create a chart to track progress.

At the beginning of the third sprint, POs created two extra user stories to be implemented in order to simulate a real, changing work environment.

Also, during this phase teams met in periodic reunions to discuss their advancement and to let everyone inform others on what they planned to work on during the remaining part of the sprint. At the end of every sprint, teams had to meet POs to present a working demo of their projects at the actual state of art and a retrospective made with Essence cards[2].

3) Final phase: When the team felt that the product was complete enough, they had to write a final report to accurately summarise their process and make a demonstrative video of their product. Eventually, every student individually answered to a questionnaire about their perception of the work. The final evaluation was assigned by POs after a meeting similar to a sprint review, including a demo and a final retrospective including a presentation of the final report.

C. The agile open source environment

We gathered and analyzed data produced by students during the development of the project work for the Software Engineering course. The teams used the services offered by the Compositional Agile System (CAS) [7]. This is an open source, open-ended development environment conceived in order to adopt an Agile approach based on a process model inspired by Scrum introduced for critical systems development. The implementation of CAS aims to provide an autonomous environment which can be deployed on a private server or a hybrid cloud, and extended with additional open source tools, while keeping complete control over the data and source code stored within. The CAS structure is composed of a server, which is a basic set of shared services, and some clients, which are used by developers to interact with the services, as shown in Fig.2.

A CAS Client is installed by the developer along with an IDE of their choice, among the supported ones, and consists of two distinct parts:

- A plug-in for the chosen IDE to monitor the development and record the metrics which will be sent to the logger service hosted on the CAS server.

- A Browser capable of interfacing with the logger dashboard on the server in order to retrieve and show all the metrics with graphs.

1) CAS Server: All the CAS services are open source and each one of them has a utility in contributing to different activities in the agile development process
2) CAS-Logger and CAS-Dashboard: CAS-Loggeris a self tracking service that allows each user to view personal statistics generated by the actions performed on an IDE, equipped with a plugin that connects to a CAS service repository.
To be able to consult the statistics in detail, users have to log into the CAS platform.

CAS-Dashboard is a web app that aggregates data from all the CAS services used either by a single developer or by the team. Each section of the dashboard interacts with the associated service through the APIs made available by the service itself.
The extrapolated data are manipulated to return tables or graphs, like those in Figure 3.

[2]https://practicelibrary.ivarjacobson.com/start/

The loggers track and record five different types of measurements, either for a single developer or for the whole team:

- **Lines of code**: the plugins determine, through the use of diff libraries in Java and JavaScript (ECMAScript), the number of lines added, modified and deleted every time a file is saved.
- **Comments**: with the same libraries for lines control, using pattern matching it is determined how many comments have been added or removed.
- **Lines of test**: the same procedure is used to identify the lines relating to testing.
- **Refactoring**: each editor emits signals for the events generated through the use of the IDE itself.
  The plugins capture metrics by determining duration and type of each individual operation and ignoring events that are not considered significant.
- **Session**: an authenticated user also generates activity metrics that include hardware details of the machine they are working on, in particular the IP and MAC addresses, which will be associated with the type of activity being carried out.

D. Data gathering and elaboration

This section describes the data collection process.
The analyzed data come from the tools present in the CAS environment, final reports, and individual questionnaires.

In particular, data on productivity, code quality and workload organisation of the teams were collected from the CAS environment, information on the process followed by each team was gathered from the final reports and, finally, the students' perceptions of various aspects of the project were collected from individual questionnaires.

The data was aggregated and analyzed using an electronic spreadsheet: some interventions were necessary to integrate or correct some answers which, depending on the case, had been omitted or entered incorrectly and would have been unusable for the analysis.

1) Data obtained from the CAS environment: The main data sources within the CAS environment were Gitlab, Taiga and SonarQube. Gitlab was used as versioning system. Taiga was used to share the product backlog and to visualize the progress of every sprint. The teams used Taiga to split user stories into tasks, the relative estimates assigned based on the perceived difficulty (expressed in story points). Taiga updates the burndown chart sprint by sprint. Students learned to appreciate the power of this representation of effort as the instructors monitored their progress.

There are other two tools in CAS, namely Mattermost for team communication and Jenkins for testing. Some teams, not all, used these tools as well.

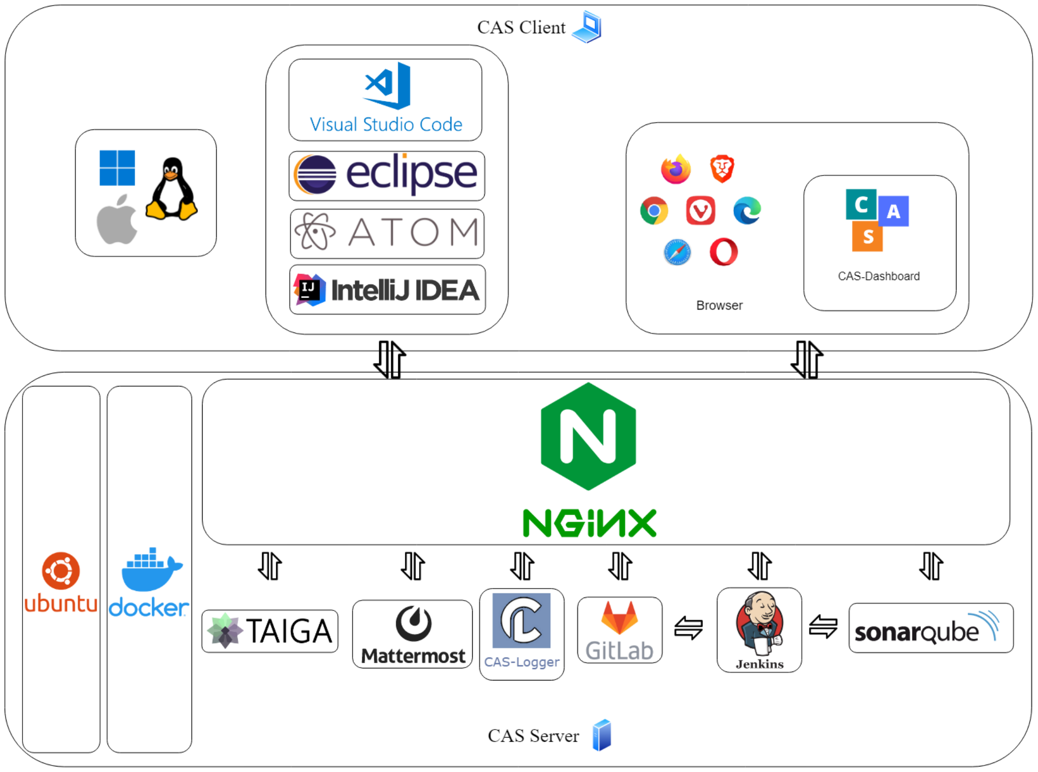


Figure 2. Structure of CAS services, which are all dockerized

In the end, data related to the number of lines of code (LoC), comments written and the general quality of the code were obtained from SonarQube in terms of scores for: reliability, security, maintainability, number of tests performed and duplicate code blocks. An example of SonarQube evaluation screen is shown in Figure 4.

2) Data obtained from the final reports: The final reports were a vital source of data as, in addition to containing several mandatory information, such as sprints descriptions and retrospectives, most teams included additional elements regarding their process. These elements were used to better understand some answers contained in the individual questionnaire or to better interpret certain data.

The requirements for the final reports included:

1) Product description, including scope, backlog and UML diagrams for use cases and architecture;
2) Description of each sprint:
   a) Sprint goal, sprint backlog, and definition of done,
   b) Implemented tests,
   c) Sprint burndown,
   d) SonarQube evaluation of the product increment,
   e) Sprint retrospective performed using Essence cards.
3) Process description:
   a) team forming with Trello and team building results of a Scrumble game,

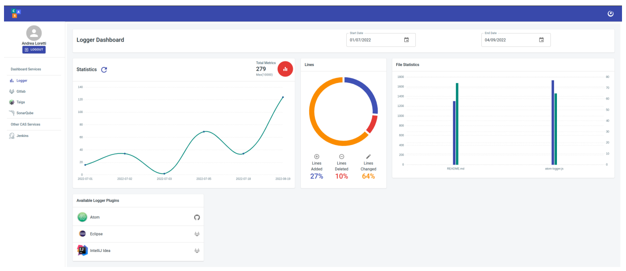
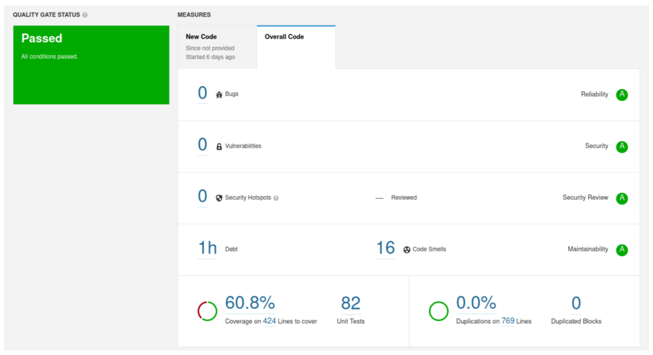


Figure 3. CAS-Dashboard


QUALITY GATE STATUS

Passed
All conditions passed.

MEASURES

New Code
Since not provided
Started 6 days ago

Overall Code

0 Bugs — Reliability A
0 Vulnerabilities — Security A
0 Security Hotspots — — Reviewed — Security Review A
1h Debt — 16 Code Smells — Maintainability A
60.8% Coverage on 424 Lines to cover — 82 Unit Tests
0.0% Duplications on 769 Lines — 0 Duplicated Blocks

Figure 4. Example of SonarQube evaluation of a release of the product at the end of a sprint: the green area says that the quality gate was passed, even if the test coverage is only 60.8%, and the remaining technical debt is 1 hour

b) Total logger and Gitinspector data,

c) Final retrospective.

3) Data obtained from individual questionnaires: Individual questionnaires were the main source of the analyzed data. The individual final questionnaire was submitted to all the students a few days before the final discussion with the POs via Google Forms. The 37 questions are described in Table I.

Table I
DESCRIPTION OF THE 37 QUESTIONS OF THE INDIVIDUAL QUESTIONNAIRE

Question number	Answer type
1	Team number
2	personal ID number
3a, 3b	PO or SM role
4	Personal IDE used
5a, 5b	Personal estimation of produced lines of code
6	Agile practices used
7, 8	Personal retrospective
9a, 9b, 9c, 9d, 9e, 9f	Anonymous evaluation of members contribution (%)
Da 10 a 25 e da 31 a 37	Evaluation of internal team interactions
26 , 27	Product and process self evaluation
28, 29, 30	Estimation of programming or team support hours

E. Teamwork quality model

Once the data was collected and the necessary modifications made, it was possible to apply a teamwork quality model. We have chosen a model developed and described by Martin Hoegl and Hans Georg Gemuenden [11], based on the analysis of internal team interactions, performance analysis and team satisfaction analysis in relation to the final product. This model was later applied to Agile development teams by Dingsør et al. [16]. The model relates to the team performance, the quality of teamwork, measured through the analysis of interactions within the team, and the personal success of team members to return an estimate of good quality.

In Table II, quality model parameters and performance data sources are listed by categories. Most of the data are taken from the answers students gave to their individual final questionnaires.

Table II
TEAMWORK QUALITY MODEL PARAMETERS AND DATA SOURCES

Parameter	Data sources
Performance analysis	
Productivity analysis	Questions 5a, 5b, 28, 29 and 30
Final product quality	SonarQube analysis and final project grade
Internal interactions analysis	
Communication	Questions 10, 11, 12, 13, 14, 28 and 29
Coordination	Questions 15 and 16
Effort prioritization	Question 20
Mutual support	Questions 17, 18 and 19
Cohesion	Questions 22 and 23
Effort balance	Questions 9 (a, b, c, d, e, f), 21 and 24
Team satisfaction	
Team satisfaction	Questions 25, 26 and 27

F. Agile maturity model

We present a teamwork maturity model following the one proposed by Yin [24], which was based on Chetankumar's one [4]. We adopted the maturity model as explained in [8]. The model provides five levels of maturity (1 lowest to 5 highest). Some objectives are associated with each level and a score is assigned for each. The average score for a level is called the Key Process Area (KPA). The calculation of KPA is based on a series of questions to be answered with "Yes," "No," "Partially," or "Not applicable"; each question has been answered by the authors of this paper after reading the final reports. The final score is calculated according to Formula 1.

$$\frac{\sum N_S + \frac{1}{2}\sum N_P}{t - \sum N_{NA}} \times 100 \tag{1}$$

where N_S represents the number of "Yes" responses, N_P the number of "Partially" responses, t the total number of questions, and N_{NA} the number of "Not applicable" responses.

If the KPA score is between 86% and 100%, the maturity level is considered completely achieved. Lower scores (between 51% and 85%) indicate sufficient achievement of the maturity level, partial achievement (up to 16%) or failure (below 15%). It is not possible to reach a maturity level if the previous one has not been fully reached.

IV. RESULTS

In this section we will present the results of our analysis of the data collected from the tools and the final reports written by the teams.

A. Students' choices

We are now presenting the results regarding the self organizing teams. In fact, as stated before, students were free to choose the tools they desired to carry out the project.

1) IDE and logger: An interesting fact we discovered from the questionnaires is the usage of IDEs and Loggers: students used different ones and sometimes more than one as presented in Table III (IDEs) and Table IV (loggers).

Table III
IDES USAGE BY YEAR

IDE	Usage in 2020/2021	Usage in 2021/2022
Atom	0	54
Visual Studio Code	26	22
IntelliJ Idea	20	10
Vim	0	2
Eclipse	5	1
Emacs	0	1

Table IV
LOGGERS USAGE BY YEAR

Logger	Usage in 2020/2021	Usage in 2021/2022
Atom logger	0	53
Logger for IntelliJ Idea	0	3
Logger for Eclipse	0	1
WakaTime	11	1
Innometrics	15	0
No logger	21	22

Also some students did not use a logger.

2) Programming languages: The nature of the product required a mix of languages, in general different for the front-end and for the back-end. Students chose a main programming language as shown in Figure 5:

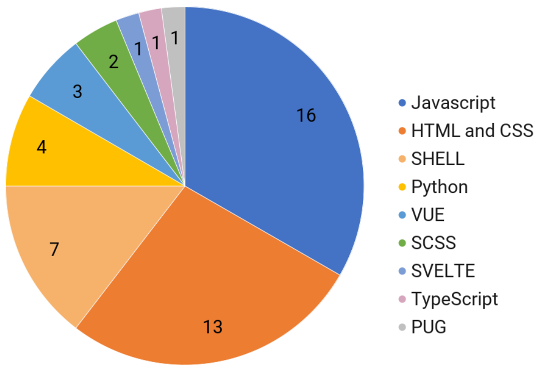


Figure 5. Programming languages used by teams in 2021/2022

From this pie chart it is possible to understand that most of the teams in 2021/2022 chose to develop a web application, but some decided to deliver other kinds of applications. The 2020/2021 teams also developed mostly web apps: Javascript was the language of choice for seven out of eleven teams (37 students), in contrast to the other four who chose Java (20 students).

B. Agile practices

There were several Agile practices to choose, some were mandatory, but some were freely selected by the students. Clearly the most reported were the mandatory ones, but several students added others. Figure 6 presents the results for year 2021/2022.

In year 2020/2021, fewer agile practices were employed. In particular, sprint planning, mob or pair programming and retrospectives were adopted by all students, while daily or weekly scrum only by 25 out of 57.

C. Retrospectives

At the end of each sprint and at the end of the entire development process, each team was required to create a retrospective using Essence cards. These cards describe typical agile development practices. During a retrospective the Scrum Master first selected some cards, then placed them inside a table. The table consisted of three columns, on which the cards

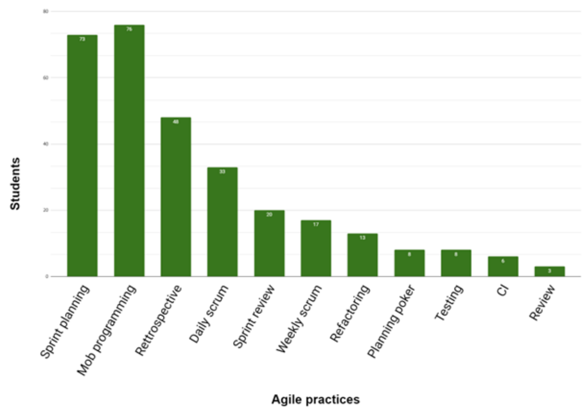


Figure 6. Agile practices used by students in 2021/2022. ”Testing” means automated testing.

were placed according to the team’s self-assessment of the performance of the activities (good, normal, bad) and three rows, which represented the priority set for that activity in the development context (high, medium, low). In addition, there was a fourth column in which the team could make notes on the aspects to be improved in the next sprint or that they would like to improve during the course of the project, again on the basis of the corresponding priority. Fig. 7 shows a table filled by a team.

The retrospectives were a way for students to reflect on their activities, to self-assess themselves and to improve their own development process as they went along.
Several teams reported in their final reports the improvements they had achieved in a recently completed sprint thanks to the retrospective conducted at the end of the previous sprint.
Finally, the students were asked to rate the perceived effectiveness of the retrospectives with essence cards and the result was 3.75 out of a maximum of 5 (StDev. $\sigma = 0,98$).

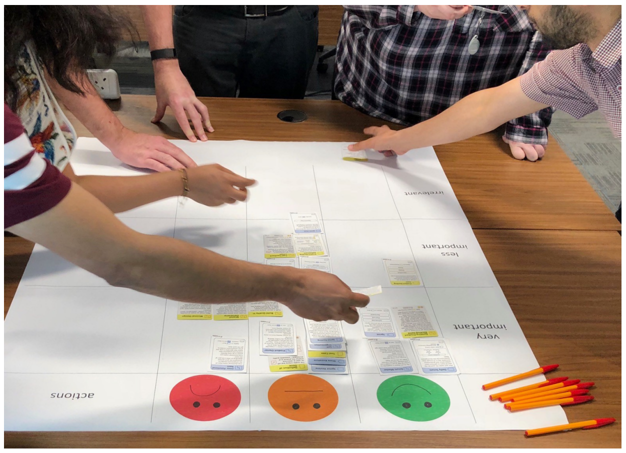


Figure 7. Using Essence during a retrospective

Table V
COMPARISON BETWEEN 2021/2022 DATA AND 2020/2021 DATA

	2021/2022	**2020/2021**
Number of teams	16	11
Involved students	79	57
Performance analysis		
Avg. error on LoC self-estimate	882	619
Avg. devs LoC	1688	721
Avg. devs working hours	83	55
Avg. SM LoC	1484	806
Avg. SM support hours	32	52
Avg. SM programming hours	91	42
Avg. product quality	90.1%	92.0%
Interactions analysis		
Avg. communication quality	74.51%	66.32%
Avg. coordination quality	83.69%	85.61%
Avg. mutual support quality	75.62%	82.60%
Avg. effort prioritization quality	79%	71.80%
Avg. cohesion quality	66%	73.20%
Avg. effort balance quality	85.02%	80.20%
Satisfaction analysis		
Avg. satisfaction	82.65%	82.20%
Avg. quality	83%	85.16%
Teamwork maturity analysis		
Avg. maturity level	3.18	2.36
Avg. KPA	88.10%	76.67%

D. Teamwork quality and maturity models

We now present the teamwork quality and maturity models' results.
The data are presented in Table V: the first column indicates the parameter being considered, the second column the data for Academic Year 2021/2022, and the third column the data for Academic Year 2020/2021.

Starting with the first two rows, it is immediately possible to see that for the year 2021/2022 the sample, both as the number of teams and the number of students was larger.

Passing to performance analysis, it can also be seen that the developers in 2021/2022 wrote more lines of code on average and at a higher rate.

The Scrum Master's role as developer was maintained: in both cases the average LoC SMs reported is very close to the average of the lines reported by developers.

Regarding meeting times, the values are similar. What varies are the total duration of work: in the case of the 6 ECTS course the hours reported by students are 29.8% more than in the 9 ECTS course. It is important to underline that the productivity data come from students self-estimates and is thus subject to large approximation, as it is possible to see in the "average error on LoC self-estimate" row. Turning to the analysis of team interactions, only minor differences can be seen in product quality, coordination quality, and average satisfaction. The overall average quality, calculated from the average values obtained by each team, also did not change much. Switching to the maturity model, we have that in the year 2021/2022 the result is better, but this is partially due to the new mandatory requirements.

Moreover it is important to clarify that even if the average KPA is greater then 86%, it does not mean that on average every team reached such a high (> 3) maturity level. In fact the average KPA was calculated from every level's KPA, even if one team did not reach a specific level.

Finally, we present Figure 8 and Figure 9 to illustrate the average teamwork quality and maturity for each team during each year. Comparing the two graphs, we observe that the second one shows a better maturity, reflecting an improved clarification of the agile practices used by the students.

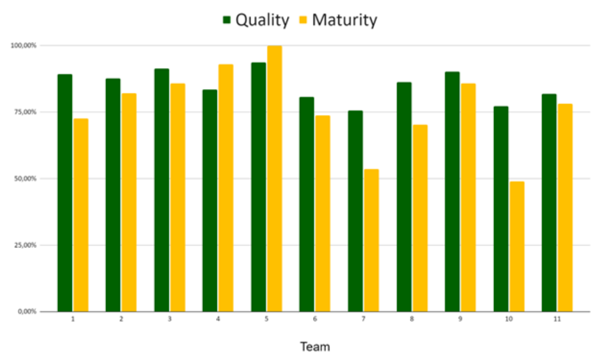


Figure 8. Comparing teamwork quality and maturity for teams (2020/2021)

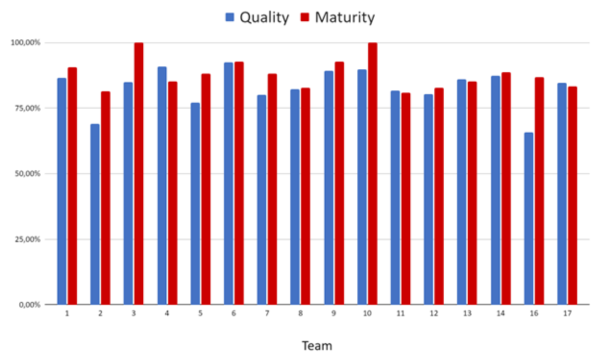


Figure 9. Comparing teamwork quality and maturity for teams (2021/2022)

V. DISCUSSION

First, we checked the projects' adherence to the 12 agile principles[3]: with the exception of the fourth principle ("Business people and developers must work together daily throughout the project") which was not strictly applied, due to the university context in which daily meetings were not affordable, all others were respected.

[3]https://agilemanifesto.org/iso/en/principles.html

Following the sixth agile principle ("The most efficient and effective method of conveying information to and within a development team is face-to-face conversation"), most of the meetings took place online because of the restrictions due to the pandemic. This necessity enhanced not only the usage of collaborative tools (the students used MS Teams for most video calls, as it was made available for lectures and they could use it for their meetings) but also of development tools like GitLab and SonarQube.

Another important point of discussion concerns the students' difficulty in estimating their own code production in terms of lines of code written: in fact, it appears that in both academic years, lines of code counted by SonarQube and those estimated by students differ in excess by a significant percentage as shown in Table V.

Starting from this observation, it is possible to make another regarding the massive hourly production of lines of code. In fact, by overestimating the lines of code written and possibly underestimating the actual working hours, students in both years achieved very high writing rates.

Another issue concerns Scrum Masters: in fact, those in this role are not usually asked to write code, but in our context each SM was first and foremost a computer science student and was both a support figure and a programmer, so the majority wrote code as developers.

Turning to the choice of agile practices, it is noticeable that several students were reluctant to apply practices that were not strictly compulsory.

Comparing analyses of internal team interactions over the two years, it is possible to see an improvement in criteria strictly related to project execution, i.e. organisation of effort and work-related communications, and a deterioration in those strictly related to social relations between developers, especially cohesion.

Finally, the correlation between the results of teamwork quality and maturity models was verified for both years.
The Pearson correlation ρ index was used for this purpose: it returns a value between -1 and 1, where -1 represents inverse linear correlation and 1 direct linear correlation.
In the year 2020/21 was obtained $\rho = 0,8$, while in 2021/22 it was $\rho = 0,436$.
This difference is probably due to the fact that there were more compulsory requirements in the project guidelines for the year 2021/2022. The compulsory requirements influenced the assessments in the two models, especially the maturity model.

VI. Conclusions and future work

We have presented the results of a two-year long observational study which consisted of training several student teams to use an open source environment for agile developments in order to develop a Twitter client with several capabilities for visual analytics. Teams were asked to follow a Scrum-like process and produce a demo and a process report. Teams were also requested to use during their retrospectives a choice of Essence cards, and perform some activities to analyze how good was their process, to be recorded on their final reports.

Several product and process data were gathered from the tools used in the open source development environment, the final reports, and individual questionnaires. A teamwork quality model and an agile maturity model were also used to analyze the data and compare the performance of the teams (we remark that these analyses were not used for the final individual grading of the students).

We found that the students were engaged by the agile approach, improved their cooperation attitude thanks to team build activities. Notably, all teams were able to complete their projects working during the course and delivering the final product just after the end of the lectures.

Students were not happy to measure themselves, being afraid that the use of tracking tools could be used to grade their work. Instead, they felt that their ability to work in a team was greatly improved after this experience.

We are now preparing the next round of this experiments, in which we would like to extend the set of tools, for instance in order to support retrospectives with improved tools [6]. We also plan to suggest the usage of tools for automating integration and delivering following a DevOps approach, to be applied applied to a product to be deployed in the cloud.

Another approach we are considering consists of including a Large Language Model like OpenAI's GPT as team member. In combination with specification animation [22] and automated testing [1], these offer a powerful method of requirement analysis. Our approach should help students to understand the reasons for selecting a specific agile practice, and to explore the various roles of Product Owner, Scrum Master, or developer in a different, possibly more effective way using a low code approach.

Acknowledgments. We thank CINI and ISTC-CNR for their support. We also ack the support of CN-HPC under PNRR.

References

[1] A. Bacchelli, P. Ciancarini, and D. Rossi. On the Effectiveness of Manual and Automatic Unit Test Generation. In *Proc. 3rd Int. Conf. on Software Engineering Advances*, pages 252–257, Sliema, Malta, 2008. IEEE CS.

[2] R. B. Bass, B. Pejcinovic, and J. Grant. Applying Scrum project management in ECE curriculum. In *Proc. IEEE Frontiers in Education Conference (FIE)*, pages 1–5. IEEE, 2016.

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

[4] P. Chetankumar and M. Ramachandran. Agile maturity model (amm): A software process improvement framework for agile software development practices. *International Journal of Software Engineering*, 2, 01 2009.

[5] P. Ciancarini and M. Missiroli. Training Students as Agile Developers: Team and Role Building Games. In G. J. et al., editor, *Proc. 17th KES Int. Conf. on Agents and Multi-Agent Systems AMSTA*, volume 354 of *Smart Innovation, Systems and Technologies*, pages 289–300. Springer, 2023.

[6] P. Ciancarini and M. Missiroli. Education to Agile: fostering team awareness with Essence. In A. C. et al., editor, *Proc. 2nd Int. Workshop on Frontiers in Software Engineering Education FISEE*, volume to appear of *LNCS/LASER*, page 15. Springer, 2023.

[7] P. Ciancarini, M. Missiroli, F. Poggi, and D. Russo. An open source environment for an agile development model. In *IFIP International Conference on Open Source Systems*, pages 148–162. Springer, 2020.

[8] P. Ciancarini, M. Missiroli, and S. Zani. Empirical evaluation of agile teamwork. In A. Paiva, A. Cavalli, P. V. Martins, and R. Pérez-Castillo, editors, *Proc. 14th Int. Conf. on Quality of Information and Communications Technology QUATIC*, volume 1439 of *Communications in Computer and Information Science*, pages 141–155. Springer, 2021.

[9] V. Devedžić and S. Milenkovic. Teaching agile software development: A case study. *IEEE Transactions on Education*, 54(2):273–278, 2010.

[10] L. Gren, A. Goldman, and C. Jacobsson. Agile ways of working: a team maturity perspective. *Journal of Software: Evolution and Process*, 32(6):e2244, 2020.

[11] M. Hoegl and H. G. Gemuenden. Teamwork quality and the success of innovative projects: A theoretical concept and empirical evidence. *Organization science*, 12(4): 435–449, 2001.

[12] I. Jacobson, J. Sutherland, B. Kerr, and B. Buhnova. Better Scrum through Essence. *Software: Practice and Experience*, 52(6):1531–1540, 2022.

[13] K.-K. Kemell et al. The Essence Theory of Software Engineering – Large-Scale Classroom Experiences from 450+ Software Engineering BSc Students. In M. Kuhrmann et al., editors, *Product-Focused Software Process Improvement*, volume 11271 of *LNPSE*, pages 123–138. Springer, 2018.

[14] M. Kropp, A. Meier, M. Mateescu, and C. Zahn. Teaching and learning agile collaboration. In *2014 IEEE 27th conference on software engineering education and training (CSEE&T)*, pages 139–148. IEEE, 2014.

[15] L. Layman, T. Cornwell, and L. Williams. Personality types, learning styles, and an agile approach to software engineering education. In *Proc. 37th SIGCSE Technical Symposium on Computer science education*, pages 428–432, 2006.

[16] Y. Lindsjørn, D. I. Sjøberg, T. Dingsøyr, G. R. Bergersen, and T. Dybå. Teamwork quality and project success in software development: A survey of agile development teams. *Journal of Systems and Software*, 122:274–286, 2016.

[17] A. Meier, M. Kropp, and G. Perellano. Experience report of teaching agile collaboration and values: agile software development in large student teams. In *Proc. 29th Int. Conf. on Software Engineering Education and Training (CSEET)*, pages 76–80. IEEE, 2016.

[18] OMG. Essence – Kernel and Language for Software Engineering Methods. Version 1.2. Technical Report 18-10-02, Object Management Group, Milford, Massachusetts, 2018. URL https://www.omg.org/spec/Essence/1.2/PDF.

[19] D. E. Rush and A. J. Connolly. An agile framework for teaching with Scrum in the IT project management classroom. *Journal of Information Systems Education*, 31(3):196–207, 2020.

[20] M. Schubanz and C. Lewerentz. What matters to students-a rationale management case study in agile software development. In *Proc. SEUH Software Engineering im Unterricht der Hochschulen*, volume 2531 of *CEUR Workshops Proceedings*, pages 17–26, Innsbruck, Austria, 2020.

[21] K. Schwaber. Scrum development process. In *Proc. OOPSLA Workshop on Business Object Design and Implementation*, pages 117–134, Austin, Texas, 1995. Springer.

[22] L. Sterling, P. Ciancarini, and T. Turnidge. On the Animation of Not Executable Specifications by Prolog. *Int. Journal of Software Engineering and Knowledge Engineering*, 6(1):63–88, 1996.

[23] S. Teel, D. Schweitzer, and S. Fulton. Teaching undergraduate software engineering using open source development tools. *Issues in Informing Science and Information Technology*, 9:63–73, 2012.

[24] A. Yin, S. Figueiredo, and M. M. da Silva. Scrum maturity model. *Proceedings of the ICSEA*, pages 20–29, 2011.