Empirical Evaluation of Agile Teamwork

Paolo Ciancarini[1,2(✉)], Marcello Missiroli[1], and Sofia Zani[1]

[1] University of Bologna, Bologna, Italy
paolo.ciancarini@unibo.it
[2] Innopolis University, Innopolis, Russia

Abstract. During the fall 2020 we observed and tracked several student teams working remotely and independently to develop a non-trivial software product as the capstone project for a course of Software Engineering in our university. The teams used an integrated open-source development environment that we designed to be useful to support and measure Agile development efforts, storing all artifacts and logging productivity and interaction data. Moreover, teams were required to use the Essence visual language during the retrospectives in order to analyze and improve their Scrum-like process. The tools used by the teams were used to store and collect several process data, that post-mortem were also integrated by the answers given by the students to some questionnaires. This paper proposes an empirical evaluation of the process followed by the teams, using a teamwork quality model and an Agile maturity model. The two models highlight different facets of the teamwork. We have studied and compared the development and interaction activities of the teams, and found a correlation between the results of the two models.

Keywords: Agile · Essence · Teamwork quality

1 Introduction

Traditionally, undergraduate student projects are either personal or group-based. It is possible to assign a project to a whole class seen as a unique group, but it requires a quite complex organization [2,32].

In Software Engineering courses, student projects are usually team-based; the evaluation of the result is often based on the quality of the process enacted by the team, including the quality of the teamwork [21].

If there are several teams developing independently and in parallel, the task of the instructors is complex, because they need to track and compare the progress of all the teams. This problem is well known, and in literature some tool-based solutions have been proposed, see for instance TeamScope [10].

During the fall 2020 we observed and tracked 21 student teams working remotely and independently to develop a non-trivial software product as the capstone project for our course of Software Engineering at the University of Bologna. The teams used the Compositional Agile System (CAS for short), an

A. C. R. Paiva et al. (Eds.): QUATIC 2021, CCIS 1439, pp. 141–155, 2021.
https://doi.org/10.1007/978-3-030-85347-1_11

integrated open-source development environment we built and useful to support and measure team-based Agile development efforts [4]. The Agile Manifesto minimized the importance of both processes and tools, in fact the first of the four Agile values says: "*(...we have come to value...) Individuals and interactions over processes and tools*". The issue of effective interactions is especially important for a successful teamwork [18]. However most Agile developments in COVID-19 pandemic required the usage of tools to organize remote work and to support interactions of team members, who all worked at home. It was also important to stress the process rules - we used Scrum - as a way of making the teams more cohesive and self-organizing.

In order to support the teamwork, the teams practiced some team building activities before and during the project. In fact, we devoted some care to the topic of team building, as there was no chance of face-to-face, colocated collaboration due to the pandemic sanitary rules applied in our country. We have chosen to train students to the Scrum process model using a serious game called Scrumble[1], that could be played remotely and self-evaluated by the students. Moreover, teams were required to use the Essence visual language during the retrospectives in order to analyze and improve their Scrum-like process [9]. We hoped that the use of Essence cards could support the recording of retrospective analysis. In fact, each retrospective had to prepare an Essence-based report discussing eventual problems met during the sprint and which remedies should be introduced in the next sprint.

This paper proposes an empirical evaluation of the process followed by the teams, using a teamwork quality model and an Agile maturity model. The two models highlight different facets of the teamwork. We discuss the development and interaction activities of the teams and study the correlation between the results of the two models.

The structure of this paper is the following: in Sect. 2 we summarize some works related to our approach; in Sect. 3 we present the experiment we set up with our students, describing the most important activities and tools used; in Sect. 4 we give an overview of the results we obtained; finally, in Sect. 5 we draw our conclusions and describe some future plans.

2 Related Works

The problem of evaluation is central in education, and in case of problem-based learning (PBL) it is particularly challenging, as it requires the integration of several different dimensions.

Agile teamwork is especially complex to evaluate, as there are several possible viewpoints and approaches. For example, Poženel [22] and Wedemann evaluated teamwork interactions [30], whereas Mahnic [16] and Baham [1] evaluated the students' perceptions when using Agile practices and tools. Scott [26] and Lang [13], instead, focused on the evaluation of Scrum learning outcomes related to

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

different learning styles of students. Surprisingly, not much research exists on software process quality education, one of the few exceptions is [14]. Sussy and others present a study on the Team Software Process enacted by graduate students [29]. Instead, here we focus on undergraduates using Agile practices and Scrum.

In 2008 Dingsøyr et al. [6] wrote that Scrum was not enough studied: it was already popular in companies, but not used in university course. Since then, studies have progressed, and some focus on the approach to Agile practices in an educational context.

Kropp and Meier [12] described a course organization and some tools similar to our case. There are important differences though, such as the intensive use of Essence cards [9] and the evaluation of both process and product. Another recent paper with an approach similar to ours is [23], which exploited a number of tools for Agile collaboration and surveyed students by questionnaires. Two important differences are that we insisted that students should use only open source tools, and use Essence to guide their retrospectives.

Retrospectives are in fact a crucial practice in Agile developments and team building. Recently, Steghöfer et al. [27] completely re-imagined their course in order to teach Agile methods - and Scrum in particular - introducing student reflection in the evaluation process. Their practice has been followed by others, such as [17] and ourselves.

Traditionally, Agile developers have frowned upon tools usage [11], and though the tendency has changed through the years [5], the current pandemic crisis has been the real game-changer. Currently, most development happens remotely, even at the educational level, and required us to use a tool-intensive approach. Previous studies pioneered remote development: [25], which describes and evaluates a virtual Scrum environment, [19], that outlines a distributed Scrum activity involving students of two different universities located in two continents, and more.

3 Methods

The Software Engineering course offered at our university is based on about 75 h of theory lectures and 150 h of project work for third year Computer Science students. The theory lectures are devoted to presenting the basics of software process models, especially Agile including some practices like test driven programming or pair/mob programming. We devote time to explain and demonstrate examples of user stories, design patterns, architecture diagrams in UML, estimation techniques of development efforts, and software quality metrics.

This year we had more than 100 students, who self organized in 21 teams. They all worked remotely, as the department rooms and laboratories were off limits for sanitary reasons.

3.1 The Project Work

The students' project work has been organized in phases as shown in Fig. 1.

Fig. 1. Work phases illustration.

1. Students profiled themselves by filling personal cards using Trello, describing their previous programming experiences and their specific knowledge and skills in software development. Trello is not open source, and was chosen because it was quite popular among students, and easy to use online.
2. Using Microsoft Teams students were able to dialog and freely move cards forming groups of five or six people. This was also the time where Scrum Masters where chosen. Usually SM where chosen either because they volunteered declaring to be less able as programmers, or because they declared a strong interest in leading the team.
3. The first meeting of each group was devoted to a specific team building activity: playing remotely (mandatory, given the pandemic) a game of Scrumble to learn with a gamified approach the Scrum process model[2]. The game was useful to improve the self confidence of Scrum Masters and clarify roles and responsibilities; at the end of this activity each team had a retrospective meeting for self evaluating its own performance using a Goal-Question-Metric approach suggested by the instructors.
   This game and the subsequent Goal-Question-Metric (GQM) self-evaluation were useful to the teams for familiarizing with Scrum and discuss some critical issues, like technical debt.
4. The first sprint started when the POs introduced software requirements. Moreover, in order to inspire the teams POs gave them access to the codebase of a previous project with similar software requirements. The code base had been developed by a single student as his graduation project. Teams were allowed to reuse this code, but only two decided to reuse and extend it. The initial product backlog was formulated by the teams using a standard user story template (*As a User I need this Goal for this Motivation*). Each team built its own product backlog containing its own user stories.
5. Project work was organized over three week sprints. Three sprint were carried out by all teams, but they had the chance to add additional sprints. At the end of the work, the teams released their final software product version and a document called "Final team report", which summarized artifacts and process documentation produced during each sprint.

[2] Game manual and materials are freely available at http://scrumble.pyxis-tech.com/.

Each sprint devoted some time to a retrospective, that the team's Scrum Master coordinated referring to the Essence cards for Agile[3].

The product to develop was a Twitter client, enriched with features for data analytics: the product should be able to capture large sets of geolocalizable tweets and: a) put them on a map b) create a word cloud with their contents c) create a temporal diagram to show the distribution of collected tweets across time, and so on. The main use cases were: a) using Twitter in an emergency, like an earthquake, to collect help messages; b) using tweets to track the movements and collect the picture of a group of travelers in a city or across a region; c) using tweets for simple diachronic sentiment analysis. The teams were instructed to create independently their own user stories, so each product backlog was different. Also, the teams were free to use any technology they prefered. Most of them used Javascript with some framework, like Vue. The other teams used Java.

Each team had to develop the software product using the open source services included in the development environment hosted by a university server, and to adopt some Agile best practices and a Scrum-like process. Teams were encouraged to gather together periodically, in daily Scrum meetings, but most teams usually performed bi-weekly meetings. Moreover they practiced pair or mob programming to increase collective code ownership.

In the beginning of the sprint each team gathered together in a sprint planning meeting using an audio and/or video calling app (either MS Teams or Discord). Each team planned independently their sprint backlog, namely the selection of user stories, from their own product backlog. Each user story was decomposed in tasks and had a story point effort estimate given by the team. The process of effort estimation was performed by Planning poker.

During the sprint, each member with the assignment to complete an user story had to update its state in Taiga virtual kanban. When an user story was put in "done" status Taiga updated the sprint burn down chart.

Each personal IDE had to be instrumented with a plug-in able to log and track productivity counting keystrokes, modified lines of code, and actions on the main tools like Taiga, GitLab, and SonarQube. The plug-in was open source but not available for commercial IDEs, so several teams replaced it with Wakatime, with the difference that it tracked working hours.

The codebase had to be organized with version control inside a GitLab repository.

Source code quality had to be checked by SonarQube, which produced a report with quality ratings for each test.

Team textual communication had to be managed with Mattermost.

In the end of the sprint each team gathered together in sprint review and sprint retrospective meetings.

They stored in Taiga Wiki the following artifacts: demo video of software increment with last functionalities implemented, SonarQube report of the final release, team diary, Essence cards arrangement produced during each sprint retrospective, UML diagrams.

[3] https://www.scruminc.com/better-scrum-with-essence/.

3.2 The Research Questions

The practices and tools used by the teams recorded a lot of data concerning the process enacted by the teams. We study how agile processes can be improved, thus we aimed to compare the teamwork performances, asking the following research questions:

RQ1: *How can we evaluate the teamwork performed during the project?*
RQ2: *How can we evaluate how Agile was the teamwork?*

We used a teamwork quality model in order to answer RQ1 and an Agile maturity model to answer RQ2. We also compared the two evaluations.

The teamwork quality model is inspired from [8], thus we name it the *Hoegl-Gemuenden model*. It is based on the assumption that any human behaviour in a team can be summarized in two major areas: activities and interactions. Inspired by the Hackman model [7], which sustained that interactions influence product, performance, and satisfaction, Hoegl and Gemuenden created a survey to analyze the aspects which followed the conceptual model shown in Fig. 2.

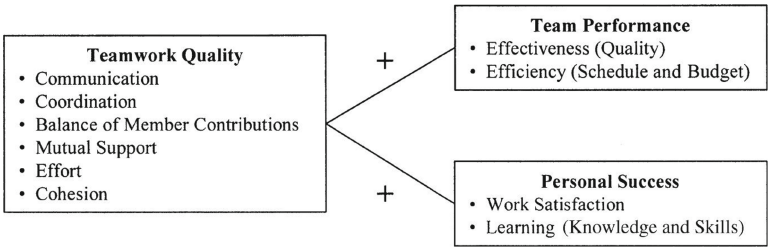

Fig. 2. Conceptual representation of Hoegl-Gemuenden's quality model [8].

The paper [15] reformulated the Hoegl-Gemuenden approach to apply it to Agile teams. Similarly, our analysis is also inspired by the Hoegl-Gemuenden model, but we applied some changes due to our educational context. Our evaluation constructs are the following:

- interaction analysis, with all the six constructs of Hoegl-Gemuenden's teamwork quality category;
- effectiveness analysis about software quality;
- work efficiency, which only considers schedule efficiency, because there was no budget;
- satisfaction analysis, which considers team satisfaction about learning, product, and process of Hoegl-Gemuenden's "personal success" category. Since our study was conducted in a short time period, questions about future personal success of team members were excluded.

Since the data collection involved different evaluation metrics (1 to 5 Likert scale for students' opinions from questionnaire about team interactions, decimal scale for instructors' evaluation of process and product, marks of SonarQube for

the product internal quality ratings, and percentages of completing user stories and tasks), in the data processing they were all converted in percentages.

The Agile maturity model we used is inspired by the Yin model presented in [31]. It includes five maturity levels and explores seven inner categories of analyses. The conceptual model is shown in Fig. 3.

Level 1 - Initial. This maturity level represents the lack of achievement of level 2 goals. This level of Agile maturity does not have a clearly defined process for Agile development, and the possible project success depends solely on the competence of individuals.

Level 2 - Managed. This maturity level represents fulfillment of the two main goals: basic Scrum management and software requirements engineering. The first area of analysis ensures the minimum acceptable usage of the Scrum methodology and structure: Scrum roles, artifacts, and meetings are used by the team. The second area of analysis comprises product backlog management and successful sprint planning meetings.

Level 3 - Defined. This maturity level focuses on the relationship with clients and on timely delivery of software products (which includes best Agile practices associated with the technical programming aspects of engineering software). The first area of analysis requires the establishment of a *Definition of Done* (DoD), frequent meetings with PO, and systematic sprint review meetings. The second area of analysis requires sprint backlog management, continuous product deliveries, verifying software quality in each delivery, usage of pair or mob programming sessions.

Level 4 – Quantitatively Managed. This maturity level includes the achievement of a standardized - repeatable - software development process aided by the management of the process performance through measurement and analysis practices.

Level 5 – Optimizing. This level focuses on the achievement of continuous self-improvement and high levels of satisfaction of both the client and the development team. The main goal for this level is: performance management through daily Scrum meetings, successful sprint retrospective meetings, causal risk analysis and mitigation and resolution.

Fig. 3. Conceptual representation of Yin's Scrum maturity model.

Yin based his model on Patel's Agile Maturity Model [20]. The paper [24] using Yin's Model proposes a evaluation system based on Patel's Key Process

Area (KPA) formula. Following this approach, we set a GQM evaluation schema in which answers to questions are: "Yes", "No", "Partially", "Non applicable".

The Key Process Area (KPA) formula is:

$$\frac{\sum Y_n + \frac{1}{2}\sum P_m}{t - \sum NA_f} \times 100$$

where n is the number of "Yes" (Y) answers, m is the number of "Partially" (P) answers, t is the total number of questions in the GQM schema, f is the number of "Non applicable" (NA) answers.

To assess the maturity level in the software development process, all its internal categories have to be fully achieved so equal or above 86%.

3.3 Collecting the Data

As depicted in Fig. 4 data were collected by surveys and observing team projects and reports stored inside in two available services: gitlab and SonarQube.

"Survey 1" and "Survey 2" were addressed to Scrum Masters. Both surveys included questions about team efficiency in each sprint, which was analyzed by counting completed tasks and user stories done in each sprint backlog. Moreover "Survey 2" included questions about ratings of maintainability, reliability, and security as obtained in SonarQube. These ratings were confirmed by a direct observation of the values reported by each team inside their specific SonarQube data repository.

The "Final individual report" was a survey addressed to all the members of each team at the end of the last sprint. It included several questions, which mostly required an answer from 1 to 5 Likert scale. The questions investigated about interactions, productivity data (lines of code personally written and working hours personally spent, either individually or in teamwork) and open-ended questions about describing the personal IDE and logger tool usage, process and practices, personal satisfaction, strengths and weaknesses of the team.

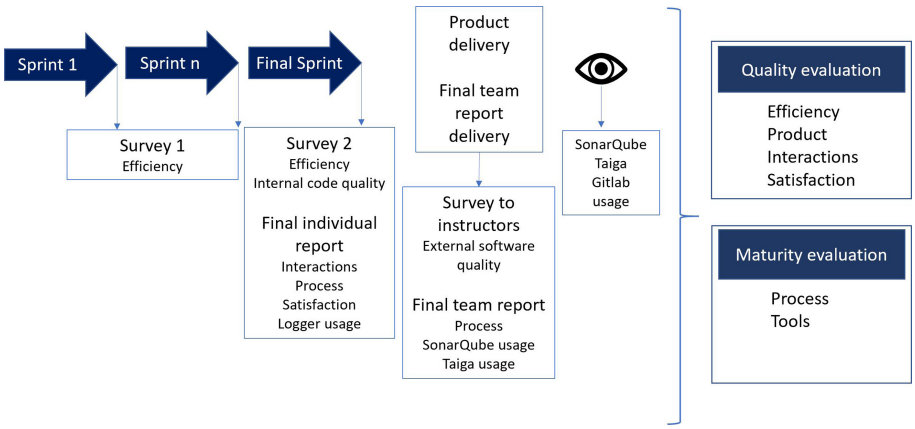

Fig. 4. Data collection process.

We performed a systematic analysis of data stored as Taiga artifacts (kanban, Wiki, tasks recorded by total power points), SonarQube reports (if each team used it in all sprints), Gitlab artifacts (if each team applied version control, updating the code throughout all work period observing contributors' graphs of commits).

4 Results

The radar graphs in Fig. 5 show the percentages obtained in each category of quality and maturity model, respectively.

Hackman's theory [7] about influence of internal interactions on satisfaction, effectiveness and efficiency is confirmed. Indeed teams which, on average, obtained the best evaluation in the teamwork quality model obtained high evaluations in these three aspects, while teams which, on average, obtained the worst evaluation resulted highly variable in these aspects.

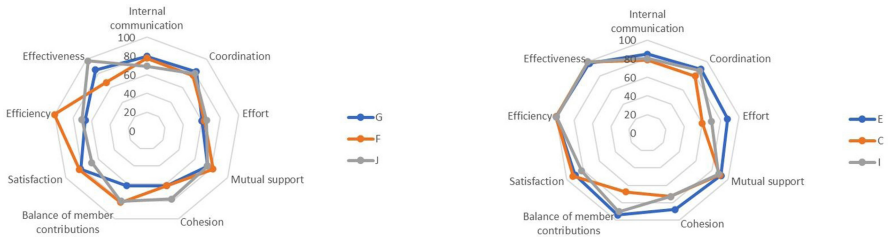

Fig. 5. Radar graphs of worst teams (G, F, J) and best teams (E, C, I) of areas analysis in team work quality

The teams which performed worst were characterized by:

- low quality of internal communication,
- low perception of effort spent in the project,
- unbalance of members' contribution to the project.

Moreover, these teams exposed often a conflict of opinions about team interactions, clearly indicating different perceptions and attitudes about teamwork.

We reported only internal communication because all teams said to not have consulted with experts or other teams. This is a behavior typical of our Computer Science students, who rarely ask for help outside their team.

Concerning the maturity model we can make two different observations about two aspects of the maturity model: one about each category of analysis, the other about levels reached by the teams.

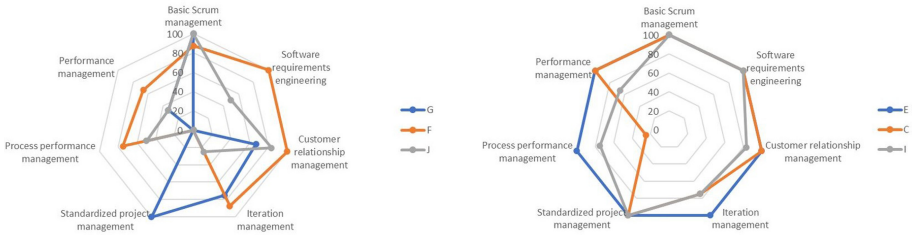

Fig. 6. Radar graphs of worst teams (G, F, J) and best teams (E, C, I) of areas analysis in Scrum maturity model

Considering the first aspect, let us consider the radar graphs in Fig. 6. Except for "basic Scrum management" category, the majority of worst teams obtained low evaluations in all categories. A common aspect of these teams is the absence of using Gitlab to apply a version control strategy, which corresponds to "Standardized project management" category.

Furthermore we observed some common difficulties in most teams in the categories of process performance management and iteration management.

Concerning process performance management, students were not constant in collecting personal productivity data neither using the logger tool which counted lines of code nor in updating burn down charts after each sprint.

Concerning iteration management, this included sprint backlog management in Taiga kanban, continuous delivery, use of SonarQube in each sprint to check and improve codebase quality, peer and mob programming sessions.

The only problematic aspect is the sprint backlog management in Taiga kanban: several teams forgot updating it most of the time, so many teams were evaluated with a "Partially" in this evaluation point. However, the majority of teams applied the other development techniques included in this category, in particular pair programming. Indeed several open-ended answers from students refer to pair programming as a valuable and useful technique. The answers underline it as a technique which encourages mutual improvement of knowledge, fast resolution of issues, a way to acquire more self-esteem about own code production, increased productivity, and creativity and fun.

Considering the second aspect of the maturity model, the majority of teams reached level two or higher, proving a basic Scrum management and good software requirements management.

In the leftmost image of Fig. 7 each point represents a team with its average KPA value in x-axis and average value of quality model evaluations in y-axis.

Data show a 0.8 Pearson correlation coefficient: this value suggests the possible presence of not perfect linear relationship among the two evaluations. Indeed the rightmost image in Fig. 7 shows a regression line not so far from the points. The average error of point distance from the line is 3.5.

We can assert that when there is a frequent use of Agile practices and control tools, there are also a good satisfaction, efficiency, product, and team work quality.

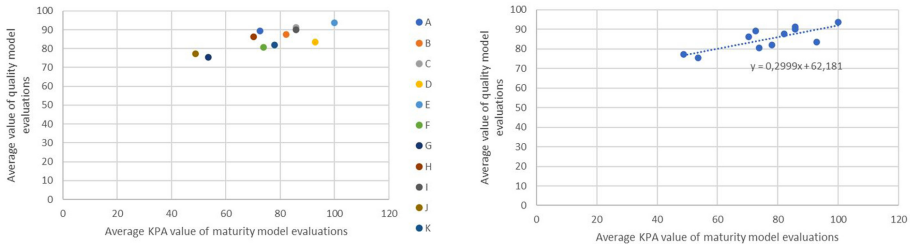

Fig. 7. Scatter graphs with average evaluations of the two models

Indeed in the analysis the worst and best teams were the same for both models. We remark that he students were not aware of any of the quality models we have used.

4.1 Students' Perceptions of Agile Practices

On the basis of students' answers to the open-ended question "List Agile practices used", as shown in Fig. 8 the top three practices used were sprint planning, pair programming, and daily scrum.

Sprint retrospective and sprint review, code refactoring, Essence and backlog usage were Agile practices quoted as useful and effective by at least 20% of students.

It is instead a strange result that kanban, burndown chart, versioning and continuous delivery were not quoted, because they were actually practices used by the majority of the teams. Our hypothesis is that since for all students this was the first experience with Agile and with Scrum, these practices were not perceived as crucial, but in some sense they were "part of the game" required by the project rules.

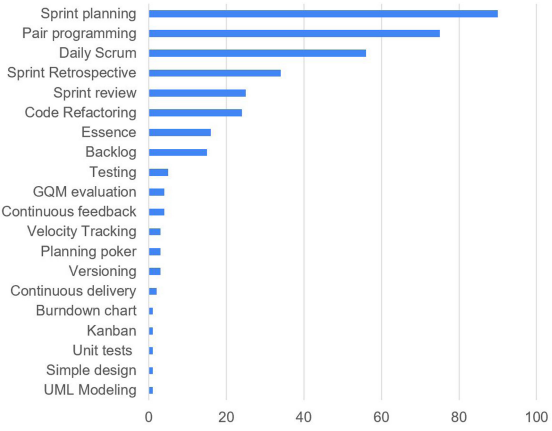

Fig. 8. Histogram of Agile practices

The uses of the kanban and versioning repository were actually widespread and effective. Instead, the idea of tracking the team's effort using the automatic burndown diagram capability of Taiga was neglected by some teams.

4.2 Threats to Validity

In this paragraph we discuss some threats to validity concerning this study. We focus first on issues connected with the pandemic, that impacted strongly the activities of this project.

First, we have chosen a specific set of open source tools, that not necessarily are the best ones for Agile developments and record data useful for quality evaluations. A different set of tools could result in different outcomes: an example is Jira, that however is not open source. Interestingly, although we insisted that the teams use the open source tools included in the environment we offered them, teams additionally used MS Teams and Discord, that are not open source but used daily for lectures and at no charge for the students. Some tools, e.g. the productivity logger, were used only partially because had some defects.

Second, the evaluation models we have used are taken and adapted from the literature of Agile developments in the industry, for teams working face to face. We found that most constructs and questions in questionnaires make sense in an academic context working remotely, however we had to delete or adapt a number of questions. We did not investigate for instance in this case study the impact of pandemic on non verbal interactions, that in Agile are quite relevant and we investigated in a recent work [3].

The students received a personal questionnaire and had to answer to questions concerning their teamwork and companions. We took measures to ensure the anonymity of the answers.

However, the same questionnaires may contain biased questions. For instance we avoided all questions connected to gender issues, but other bias could have remained unnoticed. In order to limit this aspect before sending the questionnaire to the students, we asked the opinion of two colleagues from another university expert in Agile development.

Finally, the case study has been applied once. We are replicating it in another academic context, with the same product but different students, to verify and possibly confirm or modify our findings.

5 Conclusions and Future Work

This research involved 107 students grouped in 21 teams, who worked for approximately three months at the end of 2020 and in the first weeks of 2021. All teams passed the final exam at the end of this period, delivering a product that was more or less usable and complete. In this paper we have discussed the quality of the teamwork. We remark that the data collected by the tools are available for further analyses. The experiment itself is also repeatable, as the CAS environment is all made of open source services and components.

Analyzing the results of the quality and maturity models we observed a linear relation. This relation is not perfect, indeed it has a medium error of 3.5, but it shows a possible relationship between teamwork quality and following closely a work methodology.

We argue that initiating the process with a team building game and exploiting Essence cards for guiding the teams during the retrospectives were activities very helpful for improving the quality of the teamwork observed during the project.

The main goal of this experiment was to exploit open source tools working remotely on a project developed with an Agile method and related practices, in order to collect data for quality evaluations. The teams were able to use some tools quite effectively, like Taiga, GitLab and SonarQube. Some teams decided also spontaneously to add Jenkins for automating their testing. In a future edition of our course we intend to suggest to use a tool for animating requirements [28].

Some other tools were less appreciated by our students, like Mattermost and the productivity logger. Mattermost, like Slack, is apparently superseded by MS Teams; moreover, teams based their communications on Telegram and Discord. These however are neither open source services, nor make available their data for inspections. We decided not to insist too much on using open source, trackable tools, in order to not increase the burden managed by the teams.

We believe that open source collaboration platforms, like Mattermost, need specific training when used for software development. We have also found that educating developers to self-tracking their own developing activity is quite difficult, and that the data recorded concerning productivity are not very reliable, as the developers tend to conceal or even to manipulate them. In the future we plan to improve the dashboard for self-tracking productivity data.

Acknowledgments. We wish to thank the students who participated to the project as developers.

References

1. Baham, C.: Teaching tip: implementing scrum wholesale in the classroom. J. Inf. Syst. Educ. (JISE) **30**, 141–159 (2019)
2. Blake, M.: Integrating large-scale group projects and software engineering approaches for early computer science courses. IEEE Trans. Educ. **48**(1), 63–72 (2005)
3. Ciancarini, P., Farina, M., Succi, G., Yermolaieva, S., Zagvozkina, N.: Non verbal communication in software engineering - an empirical study. IEEE Access (2021, to appear)
4. Ciancarini, P., Missiroli, M., Poggi, F., Russo, D.: An open source environment for an agile development model. In: Ivanov, V., Kruglov, A., Masyagin, S., Sillitti, A., Succi, G. (eds.) OSS 2020. IAICT, vol. 582, pp. 148–162. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-47240-5_15
5. Ciancarini, P., Missiroli, M., Sillitti, A.: Preferred tools for agile development: a sociocultural perspective. In: Mazzara, M., Bruel, J.-M., Meyer, B., Petrenko, A. (eds.) TOOLS 2019. LNCS, vol. 11771, pp. 43–58. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-29852-4_3

6. Dingsøyr, T., Dybå, T., Abrahamsson, P.: A preliminary roadmap for empirical research on agile software development. In: Agile Conference, pp. 83–94 (2008)
7. Hackman, J.: The design of work teams. In: Lorsch, W. (ed.) Handbook of Organizational Behavior, pp. 67–102. Prentice Hall (1987)
8. Hoegl, M., Gemuenden, H.G.: Teamwork quality and the success of innovative projects: a theoretical concept and empirical evidence. Organ. Sci. **12**(4), 435–449 (2001)
9. Jacobson, I., et al.: The Essentials of Modern Software Engineering: Free the Practices from the Method Prisons! Association for Computing Machinery and Morgan & Claypool (2019)
10. Ju, A., Fox, A.: TEAMSCOPE: measuring software engineering processes with teamwork telemetry. In: Proceedings of the 23rd ACM Conference on Innovation and Technology in Computer Science Education, pp. 123–128 (2018)
11. Kelter, U., Monecke, M., Schild, M.: Do we need 'Agile' software development tools? In: Aksit, M., Mezini, M., Unland, R. (eds.) NODe 2002. LNCS, vol. 2591, pp. 412–430. Springer, Heidelberg (2003). https://doi.org/10.1007/3-540-36557-5_29
12. Kropp, M., Meier, A.: Teaching agile software development at university level: values, management, and craftsmanship. In: Proceedings of the 26th International Conference on Software Engineering Education and Training (CSEE & T), pp. 179–188 (2013)
13. Lang, G.: Agile learning: sprinting through the semester. Inf. Syst. Educ. J. **15**, 14–21 (2017)
14. Lee, M., Barta, B.-Z., Juliff, P.: Software Quality and Productivity: Theory, Practice, Education and Training. Springer, Heidelberg (2013)
15. Lindsjørn, Y., et al.: Teamwork quality and project success in software development: a survey of agile development teams. J. Syst. Softw. **122**, 274–286 (2016)
16. Mahnic, V., Rožanc, I.: Students' perceptions of scrum practices. In: Proceedings of the 35th International Convention MIPRO, pp. 1178–1183 (2012)
17. Masood, Z., Hoda, R., Blincoe, K.: Adapting agile practices in university contexts. J. Syst. Softw. **144**, 501–510 (2018)
18. McEwan, D., et al.: The effectiveness of teamwork training on teamwork behaviors and team performance: a systematic review and meta-analysis of controlled interventions. PLoS One, **12**(1) (2017)
19. Paasivaara, M., et al.: Teaching students global software engineering skills using distributed Scrum. In: Proceedings of the 35th International Conference on Software Engineering (ICSE), pp. 1128–1137 (2013)
20. Patel, C., Ramachandran, M.: Agile maturity model (AMM): a software process improvement framework for agile software development practices. Int. J. Softw. Eng. (IJSE) **2**(1), 3–28 (2009)
21. Poth, A., Kottke, M., Riel, A.: Evaluation of agile team work quality. In: Paasivaara, M., Kruchten, P. (eds.) XP 2020. LNBIP, vol. 396, pp. 101–110. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-58858-8_11
22. Poženel, M.: Assessing teamwork in a software engineering capstone course. World Trans. Eng. Technol. Educ. **11**(1), 6–12 (2013)
23. Raibulet, C., Fontana, F.A.: Collaborative and teamwork software development in an undergraduate software engineering course. J. Syst. Softw. **144**, 409–422 (2018)
24. Ridha, F., Hegarini, E.: Analysis of maturity level project management of software development in scrum framework: case research on tribe enterprise PT. XYZ. IT J. Res. Dev. **5**, 87–97 (2020)

25. Rodríguez, G., Soria, A., Campo, M.: Teaching scrum to software engineering students with virtual reality support. In: Cipolla-Ficarra, F., Veltman, K., Verber, D., Cipolla-Ficarra, M., Kammüller, F. (eds.) ADNTIIC 2011. LNCS, vol. 7547, pp. 140–150. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-34010-9_14
26. Scott, E., et al.: Are learning styles useful indicators to discover how students use Scrum for the first time? Comput. Hum. Behav. **36**, 56–64 (2014)
27. Steghöfer, J., Knauss, E., Alégroth, E., Hammouda, I., Burden, H., Ericsson, M.: Teaching agile - addressing the conflict between project delivery and application of agile methods. In: IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C), pp. 303–312 (2016)
28. Sterling, L., Ciancarini, P., Turnidge, T.: On the animation of "not executable" specifications by prolog. Int. J. Softw. Eng. Knowl. Eng. **6**(01), 63–87 (1996)
29. Sussy, B.O., Calvo-Manzano, J.A., Gonzalo, C., et al.: Teaching team software process in graduate courses to increase productivity and improve software quality. In: Proceedings of the 32nd International Computer Software and Applications Conference, pp. 440–446. IEEE (2008)
30. Wedemann, G.: Scrum as a method of teaching software architecture. In: Proceedings of the 3rd European Conference on Software Engineering Education, pp. 108–112. ACM (2018)
31. Yin, A., et al.: Scrum maturity model: validation for IT organizations' roadmap to develop software centered on the client role. In: The Sixth International Conference on Software Engineering Advances, ICSEA 2011 (2011)
32. Young, P.E., Needham, D.M.: Using a class-wide, semester-long project to teach software engineering principles. GSTF J. Comput. (JoC) **3**(3) (2014)