"Extreme Development" as a Means for Learning Agile

Paolo Marzolo[1], Matteo Guazzaloca[1], and Paolo Ciancarini[1,2](✉)

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

Abstract. During the 2020 pandemic a new modality for the capstone project in Software Engineering was introduced to our third-year students in Computer Science. They have been tasked with the development of a non trivial software product - a Twitter client capable of visual analytics - using some Agile practices, exploiting a Scrum-like process model, and using only open source tools. Due to circumstances that were either planned (in the selection of tools and requirements) or unintended (the pandemic forbade any physical meeting), the project had some interesting outcomes. The project was not easy to enact, neither for the students nor for the instructors. The main problems were two: the students were not ready to practice agile teamwork, and the open source tools they had to use were demanding and only partly suitable for the goal they were chosen for. We term this experience - where students applied an agile discipline and were required to use only open source tools - an "extreme" agile development project. This paper - written by two students together with their instructor, summarises some lessons learnt: characteristics and features of the tools and practices used, the evolution of product artifacts and some difficulties encountered, along with the solutions we adopted. An important lesson learnt is that an agile project developed by Computer Science students requires specific training in communicating correct information at the right moment, and avoiding telling "social lies" concerning the status of both the product and its development process.

1 Introduction

Agile software development has been introduced more than twenty years ago, and it is now considered mainstream in the industry.

Countless higher education institutions have adopted agile as a way of introducing students to teamwork during software development. There are several approaches to teaching agile practices, most of them including some kind of teamwork training, like pair or mob programming.

However, Computer Science students notoriously do not like and are scarcely trained to teamwork [24], so the adoption of agile process models inside undergraduate courses suffers from a number of impediments. One of the most important impediments is that students have to work in groups with scarce or zero

G. Succi et al. (Eds.): ICFSE 2021, CCIS 1523, pp. 158–175, 2021.
https://doi.org/10.1007/978-3-030-93135-3_11

training to build "self-organizing teams". In fact, the topic of team building is a crucial one for agile developments, and student teams are no exception [20]. Another issue is that the students are not used to self tracking their productivity, and even less to "team tracking", namely the act of measuring the effectiveness of their teamwork.

Yet another impediment derives from the necessity of face to face cooperation between a "customer", who is often the instructor of the course, and the developing teams. The agile principles highlight the importance of face to face communication and cooperation over process and tools. This kind of communication is intended to increase the trust and collaboration spirit between the product owner and the developers. However, in an educational setting there is a specific problem: the students tend to develop the project as an effort necessary to pass the exam, so it is natural for them to minimize efforts and possibly lie about the real status of their process.

This situation changed dramatically in the spring of 2020 as the pandemic compelled most teaching to be offered online. The project has been introduced to overcome the limitations connected to a traditional exam based on written exercises. Students were given specific training concerning the process and the tools to use during the development. The tools were all open source and made available online on a departmental server, in a form downloadable and deployable on a cloud using Docker.

Students were introduced to some team build activity with the serious game Scrumble[1], that is a Scrum simulation. Students were also instructed to conduct their retrospectives with the help of Essence cards [16].

In this paper we will describe a project-based course with 21 teams including each five or six students as developers, and two people (one instructor and a teaching assistant) playing the role of product owners. The teams were requested to use a set of open source tools especially tailored for agile development. This condition was especially burdensome for the students, transforming the project in a sort of "extreme development" experience, as it combined an agile discipline, that was new for the students, with the mandatory use of open source tools, which were also new for most students.

The main research questions are the following:

RQ1: Can an agile development discipline (e.g. Scrum) and open source software tools be effectively combined when training novice developers?
RQ2: How can we evaluate the teamwork and agility of a team of novice developers who use open source tools?

The first question concerns the ability of students of using only specific development tools, exiting from their comfort zone of well known IDEs and limiting or even forbidding the use of commercial tools. We call "extreme development" this combination of agile and open source, plus the requirement of self tracking their productivity. The second question concerns the evaluation of teamwork in such an agile setting.

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

After this introduction, this paper is organized in the following sections. Section 2 summarizes some related works, aiming at describing how novice students adopt agile development practices and tools. Section 3 introduces the course structure. Section 4 describes the tools and practices integrated in the development process. Section 5 analyzes how process artifacts evolved, and Sect. 6 describes some issues in the evaluation of the students and their teamwork. Finally, Sect. 7 discusses some issues and gives suggestions for future editions of the course.

2 Related Works

Recent surveys have shown how widespread the adoption of agile software engineering practices has become [8]; although some surveys report only a portion of manufacturing companies rely strongly on Agile, the majority of them rely on a combination of agile methodologies [10], and a large percentage of software developers use Agile in their work [18].

Although Agile is, at its core, a series of principles and guidelines [2], multiple frameworks provide actionable plans and activities. One of such frameworks is Scrum [19], that is reported as one of the most used process models for software development, but it is not limited to this field: its stated objective is to help "generate value through adaptive solutions for complex problems" [22]. Scrum is not the only agile framework, and each of its practices has many variants [1].

Scrum has been used with reported success in high schools [15] and in several different university settings, both as the main learning goal, in its same-site and distributed versions, or as a method for teaching [25]. In contrast to what has been possible until now, last year's pandemic has made impossible students working together in the same room or in close proximity. Moreover, in our case, relevant government regulations changed considerably the rules to enter university labs between the first and the last sprints, making quick adaptations unavoidable. Few publications have investigated the effects of the sudden move to remote working [6].

Due to its novel nature, past research on university projects during the COVID-19 pandemic are scarce; at the same time, industry has already proved that Scrum (and agile workflows in general) can still be effective in remote contexts [17,23] the same was true during the pandemic [13]. The additional challenges presented by the changing environment were exacerbated by the documented difficulty of evaluating agile processes in university [9,21]: past research propose various metrics [14] to grade students on their application of agile process, but mostly fail to capture the ability of students to be agile instead of following any given formula. In this paper, we will outline the challenges one group faced, the support given by instructors, the adaptations they implemented and why their freedom of choice helped them learn the importance of adapting. In order to better represent their viewpoints, some parts of this article will be written from the point of view of the students.

3 Course Structure and Project Description

The course in Software Engineering at the Department of Computer Science of our University was reconfigured to face the challenges offered by the 2020 pandemic caused by COVID-19. The main novelty has been the introduction of a project to be executed using a Scrum-like process, to be enacted using several open source tools supporting remote collaboration.

In the past, the course had covered agile processes and XP/Scrum like best practices in the last few years, but only from a theoretical viewpoint. The students were tested individually by a written exam and an oral presentation. Since physical interactions were strongly limited, and written exams in presence were forbidden, the instructors decided to redefine the final exam as a team project, to be performed online to avoid unnecessary personal contacts.

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 uses case 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.

In the last few years a research project has developed an open source agile development environment deployable on a private cloud, thus avoiding any external dependence for privacy and security reasons. The environment is called Compositional Agile System (CAS) [4]. The main idea behind this environment is to offer a customizable environment, including powerful albeit free services for collaborating and managing agile development activities.

Thus, having this resource, that will be described in the next Sect. 4, the students could exploit a powerful, fully open source environment to start with. The environment can be deployed either on personal workstations, or on a departmental server, or in a public cloud.

4 Tools and Practices

The CAS environment in the version we used includes the following tools:

- Taiga for project management;
- GitLab for versioning;
- bugzilla for issue tracking;
- Mattermost for team communication;
- SonarQube for software analyses;
- open source productivity dashboard.

All the tools that were given to the students (including the productivity dashboard, discussed later) are open-source and available in a self-hosted instance.

This constraint transforms the project in an experience of “extreme development”, where all interactions, all artifacts, all data produced by the tools can be saved locally and later examined, without any dependency from external services, for instance on commercial clouds.

4.1 Proposed Tools

The instructors introduced us to the basics of Agile Development and Scrum, but they were confident in both our ability to pick it up as we went and the importance of practice. Because of the nature of the course, we did not approach this in the most focused or comprehensive way, but instead considered various alternatives and recent advances: a clear example of this is the use of Essence Cards. We will outline this and other tools and practices in this section.

Taiga. Taiga, as its website says, is “an open source project management software that supports teams that work Agile across both Scrum and Kanban frameworks”. Clearly, this software is one of the two main tools we used, together with GitLab, to manage and organize the teamwork during the software development process. Taiga’s capabilities are vast; so vast, in fact, that we found some of them useless for a project of our size, and we ignored them. They range from the basic Kanban board, to a sprint task board for each sprint, the availability of a point breakdown of each user story and task, an issue tracking system, a comment system, different roles with varying responsibilities and a lot more. For us, Taiga was a most sensible alternative to Jira, that was out of the question as fully closed-source and mostly enterprise oriented. The use of Taiga will be documented in the next section.

GitLab. GitLab needs no introduction: as the most popular open-source alternative to GitHub, it was suggested by the instructors and we quickly adopted it as our (only) version control software.

BugZilla. BugZilla was initially proposed as a complementary service to Taiga for issue tracking, thanks to its integration possibilities and historical relevancy. We decided not to use it, as our organizational overhead was already too large to include one more tool we were not likely to use.

Mattermost. Mattermost was the solution of choice by the instructors for day-to-day communication and light issue tracking (which would then be moved to either BugZilla or GitLab, integrated with Taiga). Mattermost works well: it is light, the self-hosted instance is simple to set up, it has a well-working mobile app and an appealing interface. Unfortunately, all of us had been using Telegram for the longest time (partly open-source), so even though we completed the setup we jointly agreed we would favor Telegram.

SonarQube. SonarQube was new to the students: it is a code analysis tool that runs static analysis on source code and brings vulnerabilities or possible future problems to the attention of developers. Then, it generates a report on the current position, and proposes changes to improve the code. The students used SonarQube starting from the second sprint, and it helped to find some hard to spot vulnerabilities.

Productivity Logger. The Instructors were also interested in our productivity data, for future research purposes. The environment has a logging facility which can record any keystroke pressed when using an IDE like Eclipse. This logging function was not fully set up at the start of the project. This resulted in some misunderstandings on our part and moreover raised privacy concerns that we later brought up with the Instructors. We had two main problems with the logger: its scarce availability constrained us to a single IDE we were not comfortable with and we were not given a self-hosted version to install on our own server. We will now tackle those issues and describe the steps we took to resolve them.

The plugin availability was a direct consequence of the nature of the software: this logger is the result of a mentioned research project [4]. The plugins that were made available were for the following IDEs: Atom, Eclipse, and IntelliJ. Most of our development team uses VS Code, or alternatively vim for remote development, so we were not thrilled to be requested to move to and learn a completely new (and sizable) IDE as IntelliJ WebStorm.

Some teams investigated for alternative ways to track development activity and the related productivity measures. Since the productivity data were required in the final report for documenting the process, some teams settled for using Wakatime, as described in Sect. 4.2.

Essence. Essence is an open standard for the creation, use and improvement of software engineering practices and methods. In order to classify, explain, apply and evaluate such practices, the Essence Kernel was created as part of the SEMAT initiative [12]. The Essence kernel and the Essence Language as embedded in the cards for Agile allow teams to describe, discuss, evaluate, and improve both their product and their process. Essence is now an OMG standard [16]; because of its agnostic viewpoint, independent from any software process model, it is well suited to the educational setting [3].

Essence allowed our students to avoid to study the classic Scrum documentation: they used the Essence cards for Scrum, instead. Condensing information to the size of a card is a great way to keep a reader interested and give a bird's eye view that aids understanding, without getting distracted by specific details and missing the complete picture or, on the opposite side, skipping key parts of the process. The cards themselves proved to be very useful as well, since the students used them in two "serious games" which helped us approach the Review and Retrospective activities.

The first game we learned consists of going through the seven "Alphas" - the key elements and areas of interest common to all software projects, as identified by the SEMAT Kernel - and for each of them identify which state the current product resides in. The seven Alphas are Requirements, Software System, Team, Work, Way of Working, Opportunity and Stakeholders, and their states are complemented by checklists, informal ways to move in-between states. This provides the developers with two great advantages: a clear idea of where the project sits (and which Alphas still need to be worked on), and a clear path ahead, due to the checklists. We completed this activity at the end of all sprints, since we found it extremely useful for the two uses outlined here.

Instead, in the Retrospective, we adopted a second serious game called "Practice Patience". A detailed description is available here[2], and discussion about it can be found later in this paper in Sect. 4.4.

4.2 Final Tool Configuration

Throughout the first two sprints, the tool selection varied. What we will now describe is our final usage, what we used since the end of the second sprint. We find that this selection allowed us to respect the Instructor's wishes without adding so much configuration work to go beyond a full sprint's number of hours.

We hosted all self-hosted tools on a Google Cloud machine. We picked this because of both ease of use and how generous the "first use" credit is: thanks to the initial credit and how long it lasts, we were able to use it for free. All management and system administration tasks were completed by the development team.

GitLab. We already discussed GitLab in the previous section. Git Lab was the corner stone of our tool architecture and contained the complete source code and branches. Initially, we mainly worked together on one branch, but later we moved to a feature branch approach, with new features being developed in experimental branches and merged in master once they were completed and tested. Tests were man operated.

Taiga. As we mentioned in the last section, Taiga was our management software of choice. We installed a self-hosted version of it in the same cloud as the other services, but initially used the web version while the setup was being completed. Our management of Taiga is discussed in the first Sprint Reports, but we will outline the main characteristics here. In order to include both User Stories and Development Needs, we used Taiga cards as "Backlog Items". Each Item would then go through a pipeline of stages from New to Archived. The stages were New, Wait Approval (by the POs), Wait Verify (confirm the request is within technical constraints of Twitter API and architecture), Ready, In Progress, Ready for Test, Done, Archived and Rejected. If either approval or verification failed, or if we deemed them too minor to include them at all, they were Rejected. User Stories could be moved to Done once they respected the Definition of Done; they were then Archived after the end of the Sprint. User Stories not explicitly required by the PO were marked as "optional". To distinguish between User Stories and Dev Needs, we used tags.

We used the Wiki to archive Sprint Documents, the Definition of Done and a Useful Links section. We included the Scrum Master role, but the point attribution quickly got out of hand, so we only used it partially. Each User Story had an estimated time attribute. User Stories were divided into tasks, both at the

[2] https://essence.ivarjacobson.com/publications/blog/better-scrum-through-essence-part-2.

start and during a Sprint. Each task has a field for recording how long it took to complete it, and the Tasks follow a similar pipeline to User Stories (but don't need to be approved or verified). Not all tasks were assigned, because we often worked in pairs or groups and Taiga does not allow multiple assignees for Tasks.

SonarQube. We used SonarQube since the second sprint. We did not include it into an automatic pipeline but ran oneoff scans. In each Sprint Report a section is dedicated to SonarQube metrics and performance. SonarQube was installed as a selfhosted service in the same Google Cloud account as the previous two.

Telegram. Because of our familiarity with Telegram, as soon as we picked our group members, we made a Telegram group and started chatting there. This made switching to a different service complicated. Telegram is partly open-source. We used basic messaging features, occasional polls, file upload and pinned messages the most.

WakaTime. WakaTime was our time tracking software of choice. Although all its plugins are open-source, the server code and front end are not. We understand this was a compromise on the Instructors' position, but it was driven by urgency and ease of use. In a more organized setting or a future installment, we suggest switching it with either Kimai, fully opensource but requiring more customization for our chosen use, or looking into other alternatives, such as Super Productivity or GitLab time tracker. We only used Wakatime to track IDE usage, but a Word plugin and a chrome extension are available.

Etherpad. This is a service we used for short-lived text, shared and collaborative documents. We decided against hosting it on Google Cloud and used an alternative provider (riseup) instead.

Discord. This is one of the two completely non-open-source tools we used. We used Discord because of our familiarity with it and the vastness of its features. We tried replacing it a few times with open-source tools such as Jitsi Meet, but its reliability made us use it more often than not. Still, we do not deem it unavoidable: there are many tools which provide similar functionality and, with some time and effort, we're sure they could be used instead.

MS Live Share. For the sake of completeness, we mention that we frequently used the Microsoft extension Live Share while pair-programming. It is not open-source.

4.3 Discussion

In this section, we will discuss our experience with two pervasive themes of software engineering: Being Agile compared to Doing Agile, and using Open Source Software for university projects. Then, we will review our use of Essence and what we achieved with it.

The implementation of the Agile process in our academic environment differs substantially from what is commonly done in a standard working environment: instead of following a specific set of Agile practices and a specific framework such as Scrum, we spent a lot of time thinking about what we really found useful and what we wished to change, both at the beginning and during the actual development.

This "Being Agile" way of thinking, i.e. continuously questioning our way of working while trying to improve it, as opposed to sticking to any pre-defined practice or tool, helped us a lot while having to work through unexpected problems without increasing product risk or unbalancing the team stability. These problems were either unexpected and external, such as the COVID-19 pandemic that forced us to shift to a remote setting, or internal and expected, such as the additional requirements proposed by the PO on every sprint or the privacy concerns about the logger that worried the team.

This is not to say that a completely disorganized course would be more beneficial than a structured course: as we mentioned in Sect. 3, this is only the first iteration of the new structure of the course, which meant a short amount of time to put it together. Add to this the unfortunate coincidence of the pandemic, and it is easy to see how hard it would have been to prepare accordingly. Moreover, the differences between teams made it complicated for the Instructors to grade such vastly different projects objectively. At the same time, we do wish to make an argument in favour of putting the students through some tough choices; after all, this is not a Scrum course, or a programming course: as a software engineering course, the skills of being able to select, learn, and adapt to new methods, tools, and best practices are fundamental and well connected to the modern practice of software development.

At the start of the project, we were instructed to prefer OSS (Open Source Software) tools to organize our work. As we mentioned in the Sect. 4.2, we opted against some of the suggested ones and proposed alternatives, such as WakaTime instead of the CAS Logger, that served similar purposes, although it is not a completely open-source solution. Additionally, the decision of self hosting the majority of our tools (except for Wakatime, which does not allow users to self-host) gave us a lot of freedom since we were not affected by some delays and issues which afflicted the departmental server, both at the beginning and during the project.

4.4 Essence and Framework Independence

We here discuss the role Essence had in the development project, both from process and product perspective. In our team, Essence received overwhelmingly positive feedback, so here we will outline the three most important advantages we found.

- **Product State.** The ability to clearly define the Product and Process allowed us to clearly communicate our self-evaluation to the Instructors; its checklists provided immediate actionable feedback, and going through the serious game did not need an excessive amount of time or specific tools.
- **Scrum Cards.** We used the Essence cards for Scrum. They were used in "Practice Patience" game, the Scrum Master printed them for quick reference

during the first Sprint. We found that Practice Patience revealed our true level of understanding and ability of applying Scrum, and how much we evolved compared to the previous instance.

– **Relative Novelty.** Although we understand this may not be true for all teams, and definitely should not be considered a strength of Essence *per se*, we found that its relative novelty allowed us to think for ourselves and "figure it out as we went along", rather than rely on predefined structures and practices.

Lastly, we mention an additional usage of Essence cards that we initially considered but later scrapped: using Essence to formalize the process of getting feedback and acting on it by the Product Owners (and possibly even the Professors); due to the universality the Essence kernel aims at, it is possible (and suggested) to include additional practices by using Essence to formalize practices from different development frameworks. We believe this would be an enlightening guided activity in following iterations, as it truly shows the power of Essence as an neutral, agnostic standard (as the authors call it) for formalizing and streamlining practices from different frameworks. At the same time, it would be unreasonable to expect student to have reached a sufficiently deep understanding of the standard to complete it on their own, which is why we need to be guided by instructors.

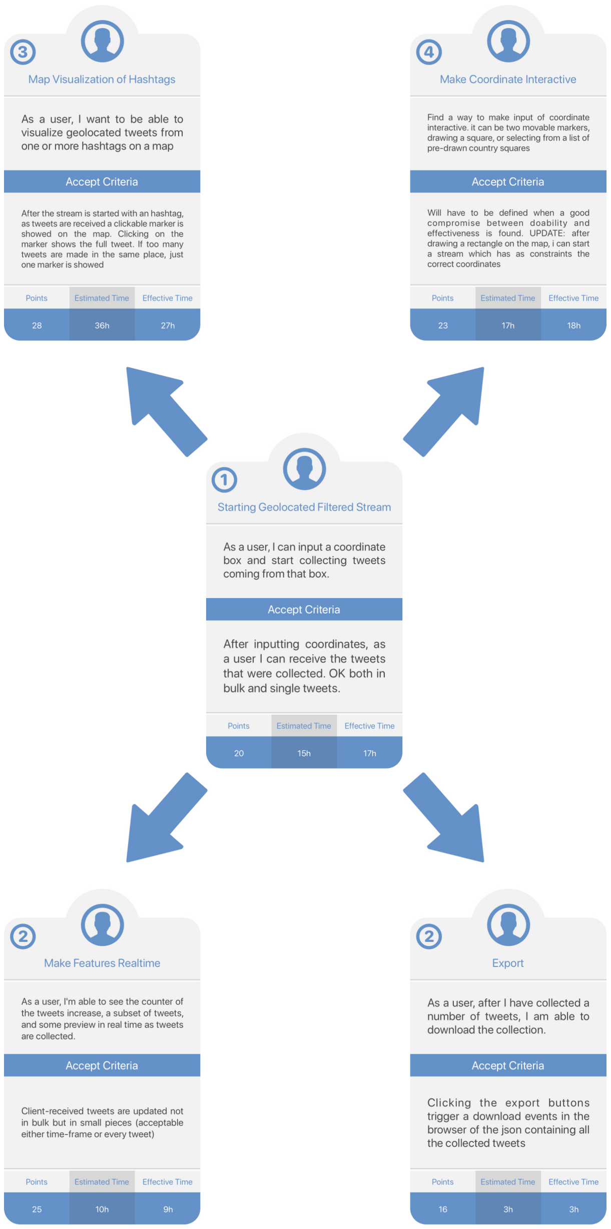
5 Artifacts

At the end of each sprint the students wrote a Final Sprint Report documenting some process data and the product status. Because of the data we collected this way, we were able to track progress throughout sprints; here, we report three relevant analyses.

5.1 User Story Evolution

First, we report a cross-section view of the way our Product Backlog evolved. In this case, we picked a main feature of the product - an epic - and a few of its notable derived user stories. Figure 1 shows some user stories in form of cards. In each card, the Sprint in which the US was completed is shown in the top left. We also included its description, part of the acceptance criteria, the points we assigned it, how long we estimated it would take to complete it and how long it actually took.

As we can see, our first story was in line with the minimum viable product; on the technical side, it required handling of the twitter API and a minimal user interface. In the second sprint both real-time and bulk versions were developed; the user story related to the map, completed in sprint 3, was included in the second sprint as well, but only completed later. The last user story, technically complex and initially optional, was only tackled in the fourth sprint.



3 Map Visualization of Hashtags

As a user, I want to be able to visualize geolocated tweets from one or more hashtags on a map

Accept Criteria

After the stream is started with an hashtag, as tweets are received a clickable marker is showed on the map. Clicking on the marker shows the full tweet. If too many tweets are made in the same place, just one marker is showed

Points	Estimated Time	Effective Time
28	36h	27h

4 Make Coordinate Interactive

Find a way to make input of coordinate interactive. it can be two movable markers, drawing a square, or selecting from a list of pre-drawn country squares

Accept Criteria

Will have to be defined when a good compromise between doability and effectiveness is found. UPDATE: after drawing a rectangle on the map, i can start a stream which has as constraints the correct coordinates

Points	Estimated Time	Effective Time
23	17h	18h

1 Starting Geolocated Filtered Stream

As a user, I can input a coordinate box and start collecting tweets coming from that box.

Accept Criteria

After inputting coordinates, as a user I can receive the tweets that were collected. OK both in bulk and single tweets.

Points	Estimated Time	Effective Time
20	15h	17h

2 Make Features Realtime

As a user, I'm able to see the counter of the tweets increase, a subset of tweets, and some preview in real time as tweets are collected.

Accept Criteria

Client-received tweets are updated not in bulk but in small pieces (acceptable either time-frame or every tweet)

Points	Estimated Time	Effective Time
25	10h	9h

2 Export

As a user, after I have collected a number of tweets, I am able to download the collection.

Accept Criteria

Clicking the export buttons trigger a download events in the browser of the json containing all the collected tweets

Points	Estimated Time	Effective Time
16	3h	3h

Fig. 1. User story evolution across sprints. The central card is an epic. In each card the completion sprint is shown top left in a circle

5.2 Sprint Backlog Sizes

Reporting the final state of our Backlog brought another restriction to our attention: for both sprints 2 and 3, a single User Story was delayed to the next Sprint only to be completed in the first few days of the next Sprint. In fact, a developer mentioned this in the third Sprint Review:

> "I really wish we could have had a few more days to complete our User Story. Even just one day would have meant not carrying it over to the next sprint..."

Although slight modifications to the timetable were allowed, because we had already overestimated our speed for sprint 2 we collectively decided that our mishandling should be accepted in order to avoid making the same mistake again, and moved it to the next sprint. That said, we believe both choices make sense, but giving a clear guideline at the start may clarify the process.

5.3 Review and Retrospective Evolution

As mentioned, our practices shifted considerably during the sprints: although our experience is of a single team, we believe further investigating the degree of strictness of rituals as teams mature would lead to interesting results. For what concerns our evolution, both Sprint Review and Retrospectives shifted considerably:

1. In our first Sprint Review, we first dedicated some time to recording a video showing the product, and then identified our current state using the "Alpha State" activity. We considered this part of the Sprint Review, but we acknowledge its purpose falls within the Retrospective as well. For our Retrospective, we held our first Practice Patience, and focused on giving actionable feedback to follow through on our observations.
2. The second Sprint's final activities were the same as the first, but further discussion was held based on our irregular progress on the Sprint Burndown Chart. Figure 2 shows the result of the Patience Practice game with Essence. We believe this retrospective was important to the team's feeling of growth and progress.
3. The third Sprint included a less structured activity, which we just called "Team Feedback". Although not all members were equally as vocal, a lot of useful feedback was collected: we agreed that the session was successful, and repeated it for both rituals.
4. The fourth Sprint included the Product State and Practice Patience serious games as well, but they only took on a *communicative* role to record our state and inform the PO of where we stood. The Team Feedback section became the main focus of both rituals.

As was outlined in the progression, our reliance on structured games and activities decreased steadily, as did our perceived gain from them. About this, we report our Scrum Master's thoughts as reported in the Fourth Sprint Report:

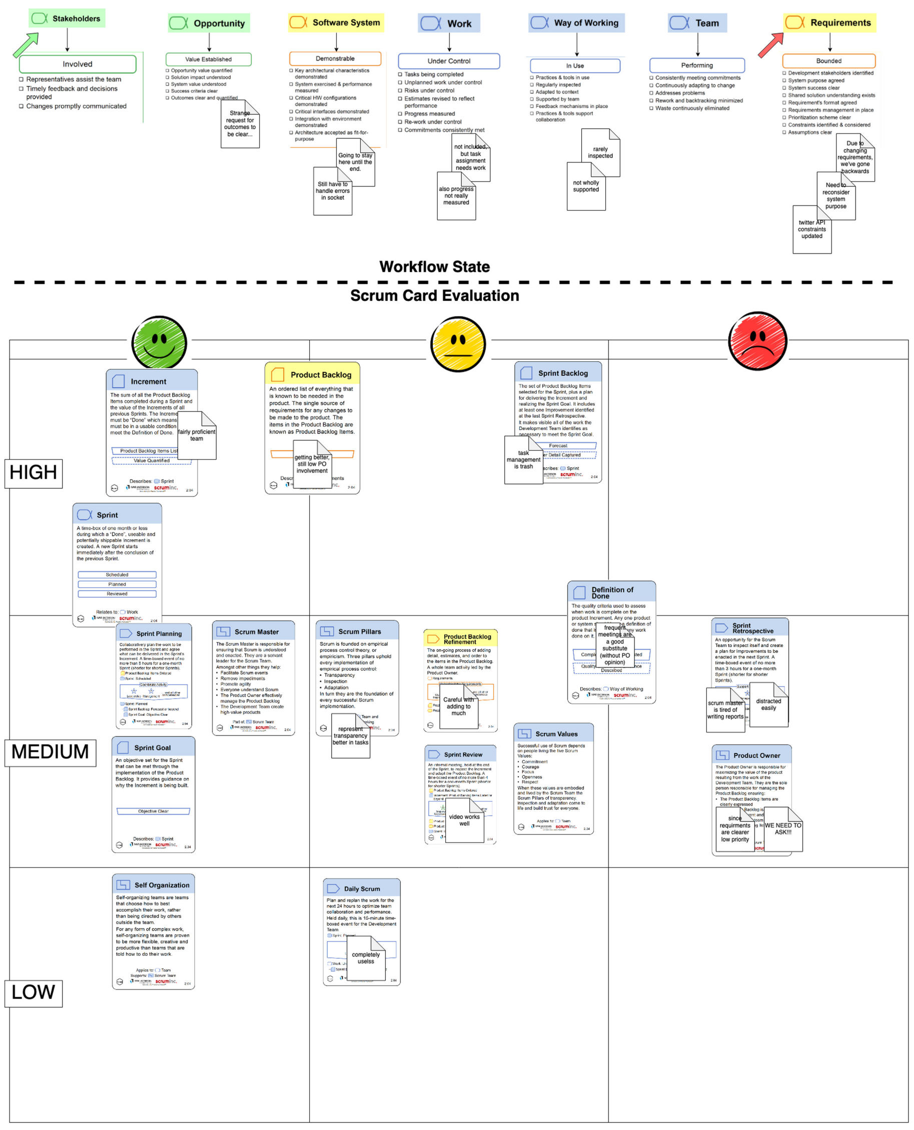

Stakeholders
Involved
Opportunity
Value Established
Software System
Demonstrable
Work
Under Control
Way of Working
In Use
Team
Performing
Requirements
Bounded
Workflow State
Scrum Card Evaluation
HIGH
MEDIUM
LOW
Increment
Product Backlog
Sprint Backlog
Sprint
Definition of Done
Sprint Planning
Scrum Master
Scrum Pillars
Product Backlog Refinement
Sprint Retrospective
Sprint Goal
Sprint Review
Scrum Values
Product Owner
Self Organization
Daily Scrum
Strange request for outcomes to be clear...
Going to stay here until the end.
Still have to handle errors in socket
not included but task assignment needs work
also progress not really measured
rarely inspected
not wholly supported
Due to changing requirements, we've gone backwards
Need to reconsider system purpose
twitter API constraints updated
fairly proficient team
getting better, still low PO involvement
task management is trash
frequent meetings are a good substitute (without PO opinion)
represent transparency better in tasks
Careful with adding to much
video works well
scrum master is tired of writing reports
distracted easily
since requirements are clearer low priority
WE NEED TO ASK!!!
completely usels

Fig. 2. The result of the retrospective of the second sprint. The top row of cards includes the seven alpha states of Essence, which represent the auto-evaluation by the team of the state of the project. The bottom part titled "Scrum card evaluation" represents the judgement on the sprint using the technique "Mad, Sad, Glad" [7]

"During this review, we all agreed that sharing the team's feedback was in fact a better way to voice our opinions than most serious games or guided activities. In my opinion, though, this was not always the case: although the developing team was very open about their doubts and trophies in this fourth review, this was not the case in some of the earlier ones."

6 Evaluation

The teams produced the following artifacts: a demo video of the final product release, its source code in gitlab, a SonarQube report of the final release, the team diary, the Essence cards arrangement produced during each sprint retrospective, some UML diagrams, personal questionnaire about team interactions. The evaluation of teams and their teamwork was conducted discussing the product in a final review and using two different quality models analyzing the main artifacts produced.

We used a teamwork quality model and an Agile maturity model. The teamwork quality model is inspired from [11], 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.

The evaluation constructs are the following:

- interaction analysis;
- 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.

Since the data collection involved different evaluation metrics (1 to 5 Likert scale for students' opinions from a 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 [26]. It includes five maturity levels and explores seven inner categories of analyses.

The radar graphs in Fig. 3 show the percentages obtained in each category of quality and maturity model, respectively.

The evaluations have be discussed in a companion paper [5]. We here summarize the results as follows.

The teams who performed best showed good balance of personal contribution and strong mutual support. The teams which performed worst were characterized by low quality of internal communication, scarce 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.

The results of the evaluation allowed the instructor to rank the teams. However, we remark that the ranking was not used to give a grade to the students, who were evaluated in a traditional way after an oral discussion concerning the final report and a demo of the last release.

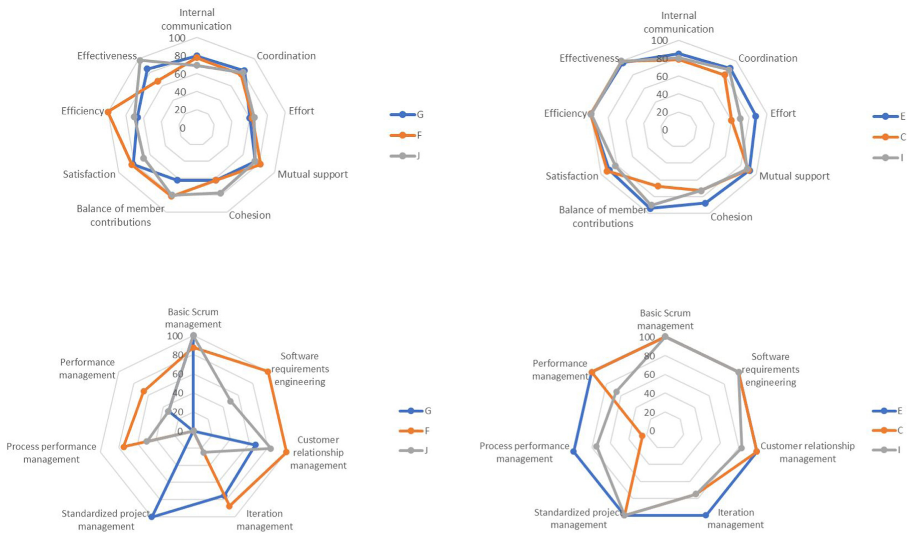


Fig. 3. Radar graphs of worst teams (left) and best teams (right) according to respectively the teamwork (top) and agile (bottom) models. From [5]

7 Conclusion

We presented the experience of one of the 21 agile teams that made up the 2020 Software Engineering Course class in our University. We aimed at reviewing the development process, in order to produce useful feedback for future work in similar contexts.

Concerning the Research Questions, we can give the following answers:

Answer To RQ1. We have asked our students to combine in a project of "Extreme Development" an Agile discipline and some Open Source tools; we believe that the combination is quite challenging and demanding for 3rd year Computer Science students of Software Engineering. All teams completed their projects, with a variety of grades.

Answer To RQ2. We have developed two quality models, one for teamwork and one for agile maturity, which have been quite effective and useful for assessing the results.

In this final section we summarize the main issues the students met throughout the sprints and what could be done in the future editions of the course.

7.1 Tools and Methods

Throughout this article, we mentioned how, in order to reach a suitable tool configuration, it was necessary to dedicate a sizable amount of time to exploratory testing and preliminary meetings. The experience was "extreme" especially because there was a strong push on using open source tools made available on a departmental server. We believe this helped push our team towards a better understanding of Agile. At the same time, we wish to reiterate that this had an effect on our productivity in our first two sprints: this is why we believe that such an activity, if deemed helpful, should be moved to the weeks preceding the start of the sprints. As part of the research behind this article, we looked into established tools selections or recent proposals, but we only found very few mentions. Some only took into account git hosting, while others expected a large infrastructure - e.g. Jira - to be supplied by Instructors. This is clearly a topic for further research.

Moreover, if the instructors wish to grade the teams on the quality of the process applied, we believe regular meetings should be held. The proposed solution of producing sprint reports was adequate for the instructors, but we believe a periodic meeting would prove more useful to both instructors and students. We understand the time instructors can dedicate to such activities is limited, so we propose that the meeting is held after Review and Retrospective have been completed, and aims at briefing the instructor on the contents of the meeting rather than participating in the rituals themselves. Alternatively, in order to include the figure of Product Owner, the instructors may only be part of the meeting for a limited amount of time.

7.2 Overlap of Learning and Applying

As outlined in the previous section, we believe less overlap between the learning and applying periods - of scrum, in our case - would have been useful to us. The Scrum Masters are also expected to have a general idea of their future duties; to these aims, we propose the following road map; although its steps may seem obvious, we found that many groups fail to complete the steps separately, and end up having difficulties during the actual coding.

1. Course and project introduction: both the course and the project should probably be introduced in the first few lessons. The introduction should make it clear that it is not possible to start early, as work will be tracked throughout the sprints, but also that listening attentively to agile practices and techniques will help make both the coding and reviewing events much easier.
2. Team building: we believe forming groups earlier would be beneficial to their ability of starting well and avoiding wasting time on simple tasks. The group will then face learning and setup as a "unit", which will help with forming interpersonal relationship that will facilitate their project development.
3. Tool setup: this is especially important if the students are expected to install their own versions of self-hosted software, as it is a time-consuming and difficult task. If some students need time to learn how to use such software, this is the correct time to do so.

7.3 Using Scrum for a Student Project

Lastly, we want to mention that in the future it should be made clear that Scrum is only a reference framework that could be modified and tailored to a specific workflow. This is because some of the requirements and practices of Scrum simply cannot be adapted to our context. As an example, we bring daily scrums: in our context of online remote education, when we have to attend several other courses beyond Software Engineering, they simply had no meaning whatsoever. They are annoying to organize, and do not allow for any further scheduling, as everyone's time constraints make it impossible to synchronize. Moreover, Scrum expects a level of involvement from the Product Owner which is simply unsustainable for an instructor alone with multiple teams.

Some teams suggested merging some of the rituals together; while we understand how useful it can be, we would advise taking into account developer exhaustion. This is because during our first two sprints, we completed review and retrospective meetings back to back; partly due to our ignorance, they both took longer than we expected, and we ended up frustrated by the amount of time they took. We also want to mention that the Retrospective - based on Essence guidance - was most definitely the most important ritual for our growth: this is supported by the final results by the teams.

As a final note, we recommend making sure to distinguish final activities and sprint planning, as they quickly collapse into one giant less-than-useful activity where it is very hard to accomplish all that's needed without growing annoyed and losing all enthusiasm.

References

1. Ashraf, S., Aftab, S.: Latest transformations in scrum: a state of the art review. Int. J. Modern Educ. Comput. Sci. **9**(7), 12–22 (2017)
2. Beedle, M., et al.: Manifesto for Agile Software Development (2001). https://agilemanifesto.org/
3. Ciancarini, P., Missiroli, M.: Teaching the essence of software development. In: Proceedings of 32nd Conference on Software Engineering Education and Training CSEE&T, pp. 1–2. IEEE (2020)
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., Zani, S.: Empirical evaluation of agile teamwork. In: Paiva, A.C.R., Cavalli, A.R., Ventura Martins, P., Pérez-Castillo, R. (eds.) QUATIC 2021. CCIS, vol. 1439, pp. 141–155. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-85347-1_11
6. Comella-Dorda, S., Garg, L., Thareja, S., Vasquez-McCall, B.: Revisiting agile teams after an abrupt shift to remote (2020)
7. Derby, E., Larsen, D., Schwaber, K.: Agile Retrospectives: Making Good Teams Great. Pragmatic Bookshelf, Raleigh (2006)
8. DigitalAI. State of agile (2021). https://stateofagile.com
9. Hanks, B.: Becoming agile using service learning in the software engineering course. In: Proceedings of Agile Development Conference, pp. 121–127 (2007)
10. Hoda, R., Salleh, N., Grundy, J.: The rise and evolution of agile software development. IEEE Softw. **35**(5), 58–63 (2018)

11. 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)
12. Jacobson, I., et al.: The Essentials of Modern Software Engineering. Association for Computing Machinery (2019)
13. Marek, K., Wińska, E., Dąbrowski, W.: The state of agile software development teams during the Covid-19 pandemic. In: Przybyłek, A., Miler, J., Poth, A., Riel, A. (eds.) LASD 2021. LNBIP, vol. 408, pp. 24–39. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-67084-9_2
14. Matthies, C., Kowark, T., Uflacker, M., Plattner, H.: Agile metrics for a university software engineering course. In: Proceedings of IEEE Frontiers in Education Conference (FIE), Erie, PA, USA, pp. 1–5. IEEE, October 2016
15. Missiroli, M., Russo, D., Ciancarini, P.: Learning agile software development in high school: an investigation. In: Proceedings of 38th International Conference on Software Engineering Companion, pp. 293–302 (2016)
16. OMG. Essence Specification. https://www.omg.org/spec/Essence/1.2/PDF
17. Paasivaara, M., Durasiewicz, S., Lassenius, C.: Using scrum in distributed agile development: a multiple case study. In: Proceedings of 4th International Conference on Global Software Engineering, Limerick, Ireland, pp. 195–204. IEEE (2009)
18. PMI: Pulse of the profession 2017 - success rates rise: transforming the high cost of low performance, p. 2017. Technical report, PMI (2017)
19. Pries, K.H., Quigley, J.M.: Scrum Project Management. CRC Press, Boca Raton (2010)
20. Sahin, Y.G.: A team building model for software engineering courses term projects. Comput. Educ. **56**(3), 916–922 (2011)
21. Schneider, J.-G., Vasa, R.: Agile practices in software development - experiences from student projects. In: Proceedings of Australian Software Engineering Conference (ASWEC), pp. 10-pp. IEEE (2006)
22. Schwaber, K., Sutherland, J.: The scrum guide: the rules of the game (2020). https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf
23. Sepulveda, C.: Agile development and remote teams: learning to love the phone. In: Proceedings of Agile Development Conference, Salt Lake City, UT, USA, pp. 140–145. IEEE (2003)
24. Waite, W.M., Jackson, M.H., Diwan, A., Leonardi, P.M.: Student culture vs group work in computer science. ACM SIGCSE Bull. **36**(1), 12–16 (2004)
25. Wedemann, G.: Scrum as a method of teaching software architecture. In: Proceedings of 3rd European Conference of Software Engineering Education, pp. 108–112. ACM (2018)
26. 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)