SCRUM PACKAGE FOR SOFTWARE ENGINEERING EDUCATION

C. Pang

MacEwan University (CANADA)

Abstract

Nowadays, software is embedded in almost all devices (e.g., car, refrigerator, kettle). Creating high-quality software is extremely important now and in the future. The process of creating high-quality software is called Software Engineering (SE). As far as I know, every undergraduate Computer Science (CS) curriculum includes SE education.

Agile is the most used SE methodology in the IT industry, and Scrum is the most popular Agile framework. Therefore, most SE courses in the CS undergraduate curriculum teach Scrum through software-developing course projects to prepare students for their careers. One challenge in teaching Scrum is its lack of a definitive definition. Organizations customize the Scrum framework according to their needs without restriction. Therefore, educators must have in-depth Scrum experience to teach students how to employ Scrum when dealing with various software development challenges. Through these challenges, educators guide students to understand the pros and cons of the Scrum framework and learn how to apply Scrum in different organizations under different situations.

Educators having in-depth Scrum experience are essential in undergraduate SE education. However, many educators, including professors, lecturers, and teaching assistants (TAs), lack industrial Scrum experience. As a technical architect consultant in the IT industry for over 15 years, who participated in over 20 projects in more than 10 organizations, mainly using Scrum, I created a Scrum teaching package for my single-term Introduction to Software Engineering (Intro-SE) course. The package includes Scrum process templates and marking schemas with students' feedback. With the package, educators can teach Scrum professionally without industrial experience. I used the teaching package in Intro-SE for six terms through three years and enhanced the package to improve students' learning experience. This paper describes the package's content, usage, and benefits. This paper also points out significant shortcomings and downfalls of the students so that the educators may be more aware of the learning opportunities, guide the students to learn from their mistakes and apply remediations.

Keywords: Software Engineering, Education, Agile, Scrum.

1 INTRODUCTION

IEEE is the world's largest technical professional organization dedicated to advancing technology for the benefit of humanity ([1]). According to IEEE, the definition of software engineering (SE) is (1) The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software. (2) The study of approaches as in (1) ([2]). Many students misunderstand SE as learning programming. However, SE is about learning the approach to developing software. A good development approach (or methodology) results in good software.

In the undergraduate Introduction to Software Engineering (Intro-SE) course, students learn how to apply software development methodology in a hypothetical software development project. Since Agile is the most popular SE methodology in the IT industry, and Scrum is the most popular Agile framework ([3]), many Intro-SE courses used Scrum. To prepare students for the IT industry, keeping the hypothetical project as close to the industrial setting as possible is essential to maximize students' experience with Scrum. Students should fulfill expectations and receive feedback on the industrial level.

One challenge in teaching Scrum is that Scrum lacks a definitive definition ([4]). Though there are many resources helping IT practitioners to learn Scrum ([5]), most IT practitioners have to advance their Scrum's knowledge through working in industry. Many Scrum resources are available to support IT practitioners. In controversial, most SE educators or teaching assistants (TAs) do not have industrial Scrum experience, which makes their teaching less convincing and sometimes confusing. This paper introduces a Scrum teaching package to fill the gap.

A Scrum project starts with defining a **product backlog**. Then, the project is split into **sprints** of a few weeks each. The teaching package consists of **sprint backlog template**, **sprint task board template**,

sprint burndown chart template, **sprint planning template**, **sprint tracking template**, **sprint retrospective template**, **sprint demo feedback template**, and **self & peer evaluations template**. The package also includes detailed marking schemas with feedback suggestions that enforce positive behaviours and recommend improvements. With the package, the educators may mark students' submissions according to the industrial practices with meaningful feedback.

2 BACKGROUND

Traditionally, SE used the Waterfall methodology to manage software development projects. However, Waterfall projects have a high failure rate. The Agile methodology was introduced to deal with the problems in Waterfall, resulting in a better success rate ([6]). Since then, Agile has dominated the software development industry. Multiple frameworks implemented Agile. Scrum ([7], [8]) and eXtreme Programming (XP) ([9]) are two popular Agile frameworks. Since Scrum has fewer restrictions than XP, it is more favorable in the IT industry and Intro-SE.

3 HYPOTHETICAL PROJECT USING SCRUM

The Scrum teaching package is for a single-term undergraduate course that lasts 13 weeks. In the first week, students form into teams of four. In the second week, the teams receive a high-level description of the hypothetical project. The educators should ensure the students know that clients can only provide high-level software requirements in real life. Before Intro-SE, most students are accustomed to receiving detailed requirement specifications from CS courses. Students only need to implement the specification word-by-word for pre-allocated marks. However, in Intro-SE, students need to understand the high-level requirements, discuss with the clients to pull their real needs and wants, get clarification for technical details, then draft detailed project requirements, and negotiate priority with the clients. The educators usually take the roles of the clients in the project.

In the second week, the teams discuss project requirements with the clients and create **product backlogs**, described in the subsection below. In the third week, the teams research and prepare technologies necessary for the software development. The teams should also establish their source code sharing repositories (e.g., GitHub ([10])) and agree upon their source code maintenance practices. There are many best practice recommendations in the industry ([11], [12]), but not the focus of this paper.

The rest of the course is split into three sprints: weeks 4-7, 8-10, and 11-13. A team will start with a **sprint planning** meeting to create a **sprint backlog** in each sprint. Throughout the sprint, the team will have **Daily Scrum** (also known as **stand-up**) meetings to keep track of the progress. The progress is documented in **the sprint tracking document**, **sprint task boards**, and **sprint burndown chart**. At the end of each sprint, the team will provide a **sprint demo** for the clients. After the demo, the team will wrap up the sprint with a **sprint retrospective** and **self & peer evaluations**. The teaching package is available on the site *https://github.com/uacspang/ScrumEduPackage*, which includes templates and marking schemas to be described in the subsections below.

Most importantly, the educators should emphasize that the course's learning objectives are how to apply SE methodology during software development under different situations, not to complete the software. Teams do not need to deliver a finished product after the three sprints. A significant percentage of the course evaluation is on the processes, not the products. However, if the products are not satisfactory, the processes must have problems. For example, if the clients expect a function to work in one way, but the product works in another, then the requirement-gathering process must have some problems.

3.1 Product Backlog

At the beginning of a Scrum project, each team creates a product backlog from the product backlog template in the teaching package. The team converts the project requirements into stories and adds to the product backlog. All Scrum stories should be in the format of "As a <user role>, I <want/need/etc.> <goal> so that <reason>." Many existing guides describe how to write Scrum stories and how to teach students to write stories ([13]). Therefore, story writing is not described in this paper. Only important lessons are included for other educators.

One significant difficulty for the students is differentiating functional requirements from non-functional requirements. For example, "As a user, I can reserve a hotel room" is a functional requirement. "As a user, I should get my search results within one second" is a non-functional requirement. In addition,

students have difficulty describing functional requirements accurately. For example, students may write a story, "As a user, I can reserve a hotel room conveniently." Convenience is not quantifiable, so it is not an accurate story. To avoid inaccurate stories, the team has to define test cases for each story. As the team tries to define test cases for "conveniently", it should become clear that "conveniently" is not testable. Then, the team should refine the story to clarify "conveniently". For example, "As a user, I can complete my hotel room reservation in one webpage."

The product backlog template, in Excel format, has a "Product Backlog" tab for functional requirement stories and a "Non-Functional Requirements" tab for non-functional requirements. The "Non-Functional Requirements" tab is self-explanatory. Each **Story** in the "Product Backlog" tab must have a unique ***ID*** (like a primary key in the database) that never changes. The *IDs* may be numbers or strings. For example, it may take the format of "<component>-#.#" to group stories by component. It is too cumbersome to refer to the story all the time. Therefore, the team should assign a meaningful **Name**, with three words or less, to each story so that other Scrum artifacts (e.g., task boards) can refer to a story by its name. After writing a story, the team should define **Test Cases** for the story. At the end of each sprint, the team should update the ***State*** of the story. A story may be completed, split, or discarded as the project progresses.

The product backlog contains stories that cover all the project's functional requirements. After the stories are defined, the clients have the right to decide the **Importance** of the stories. However, the team can determine the **Dependency** between stories. For example, if Story1 depends on Story3, then Story3 should be completed before Story1 regardless of their importance. If the team has different opinions on the significance of the stories, the team should negotiate the importance of the stories with the clients. The team should describe the meanings of the importance. For example, if the importance is between 1 and 100, is one most important or 100 most important? The team may also use words to describe importance, such as high, medium, and low.

After stories are defined and prioritized, the team should assign an estimated workload for each story. In the industry, the team estimates the number of man-days (or work days) necessary to implement a story to be its workload. The estimation is based on the assumption that a member of the team focuses on implementing the story without distraction. Since students take multiple courses, they cannot allocate a whole day to a single course project. Therefore, the package substituted man-days with man-hours. The team should estimate the hours needed to implement the story as **Estimated Man-Hours**. The estimation won't be accurate at the beginning of the project. Since Scrum is a progressive team-building framework, the team should update their estimation from sprint to sprint to become more accurate over time.

Many students misunderstand that product backlog is a static document. Therefore, they try to create a perfect product backlog before the first sprint. Actually, the product backlog is a dynamic document. The team should continuously update and improve the product backlog with the clients throughout the project. Therefore, the team should store the product backlog in a team-sharing version-controlled environment. Each team creates a Google Drive for the course to share all project-related documents.

In the industry, a project may have hundreds of stories in the product backlog. However, Intro-SE students are learning how to use stories to manage project requirements. It will be counterproductive to have many stories. Therefore, each team is allowed to have a maximum of 12 active stories in the product backlog. At the beginning of the project, the team defined 12 high-level stories that cover all functional requirements of the project. At the end of each sprint, the completed stories become inactive. The team splits the remaining high-level active stories into low-level stories to replace the completed stories before starting the next sprint.

In conclusion, the process of creating a product backlog is very delegated, involving the team and the clients. The teams should learn how to draw unwritten requirements from the client, ask proper questions for clarification, and negotiate priorities.

3.2 Sprint Artifacts

In the industry, after the product backlog is ready, the team will go into multiple splits of two to eight weeks to develop the software. As the Agile Principles ([14]) stated, "Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale." The sprints will continue until the software is delivered or the clients stop paying. Scrum is a framework of the Agile methodology; therefore, the splits are very fluid and dynamic. As the Agile Manifesto ([15]) stated, "Responding to change over following a plan." To keep the splits agile, the team and the clients decide

the goal and the tasks at the beginning of each sprint. The team and clients can revamp the project's direction from sprint to sprint.

A sprint starts with a **Sprint Planning** meeting that produces the sprint artifacts: **Sprint Backlog**, **Sprint Task Boards,** and **Sprint Burndown Chart**. Then, the team begins their software development and tracks their progress through **Daily Scrum** (or stand-up) meetings, sprint task board changes, and sprint burndown chart changes. At the end of each sprint, the team will perform a **Sprint Demo** to show their progress, solicit feedback, and obtain direction for the next sprint. Finally, the team will wrap up the sprint with a **Sprint Retrospective** meeting.

The teaching package has multiple sprint templates: Sprint Backlog Template in Excel format, Sprint Tracking Template in Word format, Sprint Demo Feedback Template in Word format, and Self & Peer Evaluations Template in Word format. The Sprint Backlog Template includes the following artifacts: **Sprint Backlog**, **Sprint Task Boards**, and **Sprint Burndown Chart**. The Sprint Tracking Template consists of the artifacts: **Sprint Planning**, **Daily Scrum**, and **Sprint Retrospective**. At the beginning of each sprint, each team creates a new sprint backlog document from the Sprint Backlog Template and a new sprint tracking document from the Sprint Tracking Template. The rest of the section describes how these artifacts and templates are used in each step of a sprint.

3.2.1 Sprint Planning

Sprint planning is part of the Sprint Tracking Template. Sprint planning is the most challenging step for the students. Most students are used to receiving assignments with explicit details and deadlines. Through the course project, students learn to decide the scope and the requirements for each sprint according to the available resources and timeline. As the Agile Principles ([15]) stated, "Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely." To keep a project sustainable, each team calculates their sprint man-hours. Then, the team will look for stories with total story man-hours approximated by the sprint man-hours according to the stories' importance.

To determine the available man-hours of the sprint, the students start by counting the sprint's available work days. A sprint starts on the day of the sprint planning meeting, and the team has up to the day before the demo for software development. Counting the days in between and subtracting the days the team breaks from the project (e.g., holidays, weekends) is the total work days of the sprint. However, the end date of the sprint should be after the demo, with time allowing the team to complete the retrospective.

Unlike full-time workers, students can only spend part of their days on some of the days on the course project. After calculating the total work days of the sprint, each member should declare the number of days they delegate for the project during the sprint and the number of hours on each of those days they will work on the project. Summing up all members' delegated hours is the total available man-hours for the sprint. For example, Sprint 2 starts on Feb 4, with the demo on Feb 25, and Sprint end on Feb 28. There are 21 days between Feb 4 and 24 inclusively. The team decides to take a break from Valentine's Day and work on all weekends. Then, the sprint has 20 total work days. In these 20 days, each member delegates to work on the project for 12 days and 3 hours on each of those days. With four members in each team, the team has (12 days * 3 hours * 4 members) = 144 **Total Available Man-Hours**. However, that is the total number of hours the team plans to spend on the project, but not all focus on software development. The team must also spend time on Scrum administration (e.g., planning, Daily Scrum meetings, discussion with clients, documentation, and retrospective). Hence, the team will use only a percentage of their total available man-hours on software development, which is called the **Focus Factor**. If the focus factor of a team is 70%, then the **Sprint Man-Hours** are (144 hours * 70%) = 100.8 man-hours. To make sure the team is not over- or under-worked, the team should implement stories with total estimated man-hours approximated to 100.8 man-hours. Most Scrum tutorials cover focus factor calculation; it is covered briefly in the retrospective section.

To find stories with approximately 100.8 man-hours, the team should look into the product backlog. The stories with the highest priority should be considered first. Ultimately, the team should negotiate the stories to be included in the sprint with the clients. After selecting the stories of the sprint, the team and the clients should set a **Goal** for the sprint. The goal should be a simple sentence describing a business purpose. The sprint goal should not include technical information or a list of stories.

In a Scrum project, members of the team must keep communicating. Therefore, during sprint planning, the team should establish times and locations for regular **Daily Scrum** meetings. There is more detail

in the sprint tracking section. The students learn requirement management, scoping, scheduling, and time management during sprint planning.

3.2.2 Sprint Backlog

Sprint backlog is part of the Sprint Backlog Template. Sprint backlog is similar to product backlog. The product backlog includes all the project stories, while the sprint backlog includes only the stories selected for the sprint. After the team and the clients agree on the stories of the sprint, the team should copy the selected stories from the product backlog to the sprint backlog.

With the selected sprint stories, the team starts the detailed design with the clients. The team should research and discuss each story with the clients to finalize all details. Since students are used to receiving finalized details from the educators, they have difficulty identifying the necessary details. For example, in a story to gather customer information, the team should extract the customer information the clients need (e.g., names, contacts). Which pieces of information are required or optional? Furthermore, there is information that the clients cannot provide; for example, what should the maximum length of the names be? As a technical team, the team is responsible for searching for relevant standards, for example, government standards on names (e.g., field lengths, allowed characters).

The team should also draft wireframes for the graphical user interface (GUI) to confirm the human-computer interaction (HCI) with the clients. The detailed design process advances the students' communication skills, which is an essential soft skill for SE that many students lack.

3.2.3 Sprint Task Boards

Sprint task boards are part of the Sprint Backlog Template. After the team has finished the detailed design, the team needs to create one task board per story. Then, the team should break each story into tasks on the task board. Many students have only done individual programming in pre-configured environments. Therefore, they lack the ability to select and set up a technical stack and break software down into pieces for multiple developers. For example, the story of gathering customer information can be broken down into setting up database tables, creating a user interface (UI), validating entered data, and inserting customer data into the database. After defining tasks for all stories, the team should look for the dependencies between tasks and create a schedule to complete all tasks by the end of the sprint. All members should sign up for tasks and update the task boards when a task is completed.

Through the task boards, the students learn how to break software development into tasks, assign tasks to teammates to make the best use of individual skills, schedule tasks according to dependencies, communicate needs between members, and support each other in challenges.

3.2.4 Sprint Burndown Chart

Sprint burndown chart is part of the Sprint Backlog Template. Since sprints are short, Scrum uses visual tools to monitor the health of the sprint. Many visual tools are available for Scrum. Only the burndown chart is included in the package to reduce the learning curve. Burndown chart reflects stories completed in the sprint thus far. The burndown chart is a line chart starting with the total story man-hours. When a story is complete, the line chart will go down by the estimated man-hours of the story. If all stories are completed by the end of the sprint, then the burndown chart will reach zero at the end of the sprint. Students improve their Excel, data analysis, and plotting skills by setting up and maintaining the burndown chart.

3.2.5 Sprint Tracking

Sprint tracking is part of the Sprint Tracking Template. The team should start developing software after completing the sprint planning, sprint backlog, task boards, and burndown chart. During the development, the clients who pay the bill need to know that there is progress. However, the Agile Manifesto ([15]) values "Working software over comprehensive documentation." Therefore, the Scrum framework eliminates most documentation. Instead, the team should summarize their Daily Scrum meetings to keep the clients informed. Since the students do not work on the course project daily, they do not meet daily. Instead, they meet in person or online according to the Daily Scrum schedule decided during the sprint planning. After each meeting, a member should create a Daily Scrum report in the sprint tracking document. To ensure all members learn how to report Daily Scrum, each member must report at least two meetings per sprint. Hence, each team should schedule at least eight Daily Scrum during sprint planning.

Daily Scrum (stand-up) meeting is the core of the Agile methodology and is covered by many tutorials. Therefore, it is not covered in detail here. In general, a stand-up meeting of a team of four should not last longer than 15 minutes. Each member should answer these three questions in the stand-up: What have you done since the last meeting to help the team finish the sprint? What will you do until the next meeting to help the team finish the sprint? What obstacles are getting in the team's way? The member reporting the stand-up should summarize the answers from all members. If any members have obstacles, the team should allocate help and set deadlines to resolve the obstacles. Most importantly, at the end of the stand-up, the team should ask themselves whether they are still **On Target** to finish all stories in the sprint backlog. If not, the team should discuss further how they will put the sprint back on target and document the remedy.

The member reporting the stand-up should also check the task boards. If all tasks of a task board are completed, the member should execute the story's test cases. If the test cases pass, the member should update the burndown chart correspondingly. If the test cases do not pass, the member should identify the problem and acknowledge the member responsible for fixing it. Finally, if any task boards or the burndown chart have been changed since the last stand-up report, include screen captures of all task boards and the burndown chart in the stand-up report. The students learn to monitor and manage the software development progress through stand-up meetings.

3.2.6 Sprint Demo

The Sprint Demo Feedback Template prepares the team for their demo. At the end of each sprint, the team needs to demo their completed stories. The template identifies what the team should cover in their demo.

In the industry, the team usually works with one delegated client day by day. However, multiple stakeholders, users, and sponsors will attend the sprint demo. Therefore, it is uttermost important for the team to demo working software and gather feedback from the larger audiences. As the Agile Principles ([14]) stated, "Our highest priority is to satisfy the customer through early and continuous delivery of valuable software." If the sponsors are not impressed, they may terminate the project.

During the course, each team has a maximum of fifteen minutes for their demo in front of other teams. The educators and other teams function as stakeholders, users, and sponsors to provide feedback. Many teams made the mistake of continuing development until before the demo and failed to show working software. The team should learn to freeze their source code one day before and rehearse their demo. Failing to do so, many teams ran into problems during demos or even crashed their software. The teams have three demo opportunities. Educators should focus on guiding the teams to improve from sprint to sprint.

Another problem was that many teams had demoed stories that had not yet been merged into the team's GitHub repositories. In that case, one member would show one story from one's computer, and another would show another story from another computer. Stories are not completed until the source codes are merged into GitHub and tested. An Agile principle ([14]) states, "Working software is the primary measure of progress." Software that has not been merged into the team repository is not working.

Furthermore, teams found it challenging to demo "completed stories only," as stated in the Sprint Demo Feedback Template. Many teams work on multiple stories in parallel, end up with all stories near completion but none completed, and have nothing to demo. Since working software is the primary measure, the teams should learn when to give or take. Concentrating their effort on completing some stories for the demo is better than completing no story. Through demos, students learn to select and focus on their targets and improve their presentation skills.

3.2.7 Sprint Retrospective

Sprint retrospective is part of the Sprint Tracking Template. Scrum is not only a project framework. It is also a team-building framework. As stated in the Agile Principles ([14]), "At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behaviours accordingly." Therefore, the team must have a retrospective meeting at the end of each sprint to learn from the past and improve the future. The retrospective meeting starts with members reflecting on the sprint. Based on their reflections, the members suggest team improvement actions for the next sprint. Then, the team uses dot-voting to select three actions to be worked on in the next sprint. Those should be team improvement actions, not software improvement suggestions.

Scrum is a framework for teams to improve from sprint to sprint. Therefore, based on the result of the current sprint, the team will calculate the focus factor for the next sprint. The next focus factor is the sum of the estimated man-hours of all completed stories divided by the total available man-hours calculated in the sprint planning. Hence, the focus factor will go higher if the team completes more stories than in the last sprint. The focus factor is usually low in the first few sprints but improves over time. In the first sprint, teams may have failed to complete any story and got the focus factor of (0/total available man-hours) 0%. In that case, suggest that the team use the default focus factor of 70%, as in the first sprint.

Every term, some students argued about how to calculate the completed story man-hours. For example, there was a story of 10 estimated man-hours with five tasks. The team finished four of the five tasks and argued that the team should count (4/5 * 10) = 8 man-hours in the burndown chart and the focus factor calculation. It is a misconception that all tasks are equal. More importantly, that is against the Agile Principles ([14]), which state, "Working software is the primary measure of progress." Until all story tasks are completed and the test cases are passed, the software will not work. Therefore, the team should target fewer completed stories over many almost completed stories.

Nonetheless, the team should update the sprint backlog with the final status of each story at the end of the sprint. Corresponding status should also be reflected in the product backlog. Then, the team should review the product backlog with the clients. With experience from the previous sprint, should any of the estimated man-hours be updated? Are the stories in the product backlog still valid? Should any stories be updated, added, split, or discarded? Should the importance of the stories be changed? The retrospective exercise helps students learn from mistakes and grow from sprint to sprint.

3.2.8 Peer & Self-Evaluation

At the end of each sprint, each team submits their product backlog as an Excel file, sprint backlog (with sprint backlog, task boards, and burndown chart) as an Excel file, and sprint tracking document (with sprint planning, Daily Scrum meetings, and sprint retrospective) as either Word or PDF file. In addition, each member should individually submit a Self & Peer Evaluation document in Word or PDF.

Each member should create a Self & Peer Evaluations document from the Self & Peer Evaluations Template to evaluate themselves and other team members for their performance during the sprint. Most of the students in Intro-SE have never worked as a team on a software development project. Occasionally, there are team issues. Through Self & Peer Evaluations, educators can identify potential team issues early and help students resolve them professionally. It is unlikely for IT practitioners to work alone in the industry. Therefore, students need to learn how to be responsible team members.

3.2.9 Submissions and Marking Schemas

In Intro-SE, 10% of the course is allocated per sprint. Six out of 10% are for the product backlog, sprint backlog, and sprint tracking documents submitted by the team. Three out of 10% are for the team demo. 1% is for the Self & Peer Evaluations. Since the Self & Peer Evaluations document is only information for the educators, the students receive 1% as long as they made a valid submission with evaluations of their teammates.

For the team submissions, marking schemas for Sprint 1, Sprint 2, and Sprint 3 are available in the package. There are minor differences between the three marking schemas. Overall, the learning objective is that the students correct their mistakes after receiving their Sprint 1 feedback and improve in Sprint 2. Then, learn from their Sprint 2 feedback and improve in Sprint 3. As in the industry, when a new employee joins a company, the new employee needs to learn the customized Scrum framework of the company. The new employee won't be perfect in their first sprint. They will receive feedback from their employers and improve according to the feedback.

Since the learning objective is to learn and improve, the marking schemas only include positive (+) and negative (-) comments. For each team, the educators review the list of comments, keeping those applicable to the team. Then, add additional feedback specific to the team. There is no particular marking ratio for the positive or negative comments. The educators can assign marks according to how much the team may improve in the next sprint. Similarly, there is a demo marking schema with positive (+) and negative (-) comments. The educators can remove demo comments not applicable to the team and add team-specific comments. The purpose of the feedback is for the team to improve from demo to demo.

Some students argue about using open-source Scrum applications they found online instead of the provided templates. Their suggestions missed a primary learning objective. Each company has multiple

Scrum teams, each working on various projects. Projects will become unmanageable if each team uses a different Scrum framework for each project. Therefore, a significant learning objective is that students should be able to adopt whatever local Scrum framework a company uses.

Many students have one noteworthy misunderstanding about the learning objectives of Intro-SE. Compared with other programming courses that they have taken, the students wrongfully presume that their marks are based on their source code. Therefore, educators must emphasize that this is a software engineering course, not an application development course. The learning objectives are learning software engineering processes. The team should pay attention to the processes.

3.2.10 Microsoft Office Skills

Last but not least, through using the templates, students improve their skills in using Word and Excel. From Word, students learn about fonts & styles, page layout, page break, header, footer, table of contents, content alignment, and more. From Excel, students learn additional skills in using equations, analyzing data, plotting charts, formatting charts, and more. Nonetheless, the students should ensure the documents have high readability and look professional.

At work, people assume CS students are proficient in Microsoft Office. Microsoft Office is very powerful and has many sophisticated features, but teaching Microsoft Office is not part of the CS curriculum. Furthermore, students are never asked to deliver professional-looking documents for courses but are expected to do so at work. Through using the templates, students learn to create professional-looking documents. Educators should point out bad formats in the submitted documents and teach students how to fix them.

4 CONCLUSIONS

Many Intro-SE courses teach SE through hypothetical course projects using Scrum. This paper introduces a Scrum teaching package to give students a Scrum experience that is as realistic to the industry as possible. The package comes with Scrum templates and marking schemas. Students can use the provided Scrum templates to create Scrum artifacts for the course project. Educators can use the provided marking schemas to give students professional feedback. This paper describes in detail how to use the templates in the Scrum processes and shared learning opportunities throughout.

The professional value of IT practitioners has long been underestimated. Many CS students also lack the recognition of getting into a profession. For example, when nurse students take their nursing courses, they realize that they are learning skills needed as nurses. However, when CS students take CS courses, many are only concerned about marks and grades. They are not trying to acquire the knowledge and skills necessary for their professional career. Students get their grades and forget the material immediately, forcing CS educators to keep reteaching material from the previous courses. A cause of the problem may be that CS educators do not emphasize the profession-ship enough. For this reason, I created the Scrum teaching package. I want the students to learn Scrum in an environment as similar to the industry as possible. I want the students to experience how real software development projects work. Using the package through six terms, some students recognized the learning objectives of the package, but some did not. Some students acknowledged that the package had prepared them for their jobs after working in the industry. In the future, I would like to study the differences between IT practitioners who turn CS educators versus academia as CS educators.

ACKNOWLEDGEMENTS

Thanks to Stephanie Husby for sharing her Scrum templates. Many features from Stephanie's templates have been adapted to my teaching package.

REFERENCES

[1] "IEEE," Institute of Electrical and Electronics Engineers, Incorporated, [Online]. Available: https://www.ieee.org/about/help/site-terms-conditions.html. [Accessed 15 04 2025].

[2] IEEE, "610.12-1990 - IEEE Standard Glossary of Software Engineering Terminology," [Online]. Available: https://ieeexplore.ieee.org/document/159342. [Accessed 15 04 2025].

[3] D. West, S. Porter and M. James, "Scrum The Most Popular Agile Methodology," DZone, Inc., 2017. [Online]. Available: https://dzone.com/refcardz/scrum. [Accessed 29 04 2025].

[4] I. Mitchell, "Agile Patterns," DZone, Inc., 2018. [Online]. Available: https://dzone.com/refcardz/agile-patterns. [Accessed 29 04 2025].

[5] H. Kniberg, Scrum and XP from the Trenches 2nd Edition, A. Ciobotaru, Ed., C4Media, publisher of InfoQ.com, 2015, p.169.

[6] A. Mersino, "Why Agile is Better than Waterfall (Based on Standish Group Chaos Report 2020)," Medium, 25 05 2020. [Online]. Available: https://medium.com/leadership-and-agility/agile-project-success-rates-are-2x-higher-than-traditional-projects-376a05e590d4. [Accessed 29 04 2025].

[7] Scrum.org, "The Home of Scrum," Advanced Development Methods, Inc., [Online]. Available: https://www.scrum.org/. [Accessed 29 04 2025].

[8] K. Schwaber and J. Sutherland, The Scrum Guide, 2020, p.15.

[9] K. Beck and M. Fowler, Planning Extreme Programming, Addison-Wesley Professional, 2000, p.160.

[10] GitHub, "GitHub - Build and ship software on a single, collaborative platform," GitHub, Inc., [Online]. Available: https://github.com. [Accessed 29 04 2025].

[11] M. Heller, "27 essential tips for Git and GitHub users," IDG Communications, Inc. (doing business as Foundry) , 09 12 2019. [Online]. Available: https://www.infoworld.com/article/2253995/27-essential-tips-for-git-and-github-users.html. [Accessed 29 04 2025].

[12] S. Patel, "15 Git tips to improve your workflow," GitLab Inc., 07 04 2020. [Online]. Available: https://about.gitlab.com/blog/2020/04/07/15-git-tips-improve-workflow/. [Accessed 29 04 2025].

[13] M. Cohn, "Introduction to User Stories," Mountain Goat Software, 06 06 2014. [Online]. Available: https://www.mountaingoatsoftware.com/presentations/introduction-to-user-stories. [Accessed 29 04 2025].

[14] "Principles behind the Agile Manifesto," agilemanifesto.org, 2001. [Online]. Available: https://agilemanifesto.org/principles.html. [Accessed 29 04 2025].

[15] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland and D. Thomas, "Manifesto for Agile Software Development," 2001. [Online]. Available: https://agilemanifesto.org/. [Accessed 19 04 2025].