Software Engineering Education: Sharing an Approach, Experiences, Survey and Lessons Learned

José Carlos Metrôlho
R&D Unit in Digital Services, Applications and Content
Polytechnic Institute of Castelo Branco
Castelo Branco, Portugal
metrolho@ipcb.pt

Fernando Reinaldo Ribeiro
R&D Unit in Digital Services, Applications and Content
Polytechnic Institute of Castelo Branco
Castelo Branco, Portugal
fribeiro@ipcb.pt

***Abstract*—To provide the best training in software engineering, several approaches and strategies are carried out. Some of them are more theoretical, learned through books and manuals, while others have a practical focus and often done in collaboration with companies. In this paper, we share an approach based on a balanced mix to foster the assimilation of knowledge, the approximation with what is done in software companies and student motivation. A survey was also carried out involving students who had successfully completed the subject in past academic years; some had already graduated, and others are still students. We analyse the results of the survey and share some of the experiences and lessons learned.**

Keywords- agile methodologies; education; software engineering; teaching; teamwork.

I. Introduction

One of the biggest challenges in teaching software engineering is empowering students with the knowledge and skills they need to be well prepared to face the labour market. This includes providing students with technical skills but also providing them with the non-technical skills associated to the software engineering process. It is also known that the teaching of software engineering cannot be limited to the presentation of concepts and methodologies as a set of abstract concepts. Wherever possible, it should be adequately complemented with the practice of software engineering projects so that the students can assimilate and understand them successfully [1]–[3] Additionally, it is important to consider the growing importance of human factors in the software development process [4] and consequently the role that some of them play in the software engineering process, namely: communication, coordination, collaboration, trust, expert recommendation, program comprehension, knowledge management and culture.

Several approaches and strategies have been proposed and used to improve the teaching and learning of software engineering. They all hold the importance of giving students hands-on experience. However, the way they propose to do so differs greatly.

This paper describes an experience in teaching Software Engineering, of a Computer Engineering program, using a project-based approach. This project-based approach is enriched with the collaboration of two software houses giving the students a real-word experience of software engineering projects development. We also try to understand how the main concepts of the course are assimilated by the students and if they are applied in the professional life of our past students. Finally, we present some lessons learned through our experience and challenges faced.

The remainder of this paper will be as follows: Section 2 presents a brief review of related work; Section 3 we present an overview of our project-based approach for software engineering; Section 4 provides a brief description of the survey that was conducted to achieve feedback from former students; In Section 5 we present the survey results and analysis; Section 6 presents some lessons learned and challenges faced and finally, in Section 7, we present some conclusions and we outline some of the future work.

II. Related Work

To provide the best training in software engineering, several approaches and strategies have been proposed. Some of them are more theoretical, more focused on the study of theory through books and manuals, while others have a more practical focus and often done in collaboration with companies. Nowadays, it seems to be a well-accepted fact that the software engineering training should not be strictly focused on the theoretical study of concepts and methodologies. It is important to provide students with hands-on experience in a software engineering project and provide them with the non-technical skills in a software project. It is important to promote hands-on ability training and the rapprochement between teaching and practice. Additionally, the recent diffusion of agile methodologies in software development brings many difficulties and challenges to software engineering teaching. In this context, several authors refer that current approaches to teaching software engineering are outdated and lack authenticity [5], [6]. However, as referred in [5], it is not clear which should be the best approach and there are different perspectives with different proposed approaches. Some authors (e.g., Clear and Damian [5][7]) suggest that the best approach is to emulate the workplace through distributed software development projects, through cross-university or cross-course courses, others (e.g., [8]–[10] suggest involving students in a project where they have the possibility to experience team working and understanding in the practice of the theoretical concepts dealt with in the course and others (e.g., [11]–[13]) argue for using simulations and games to provide students with a variety of experiences that would not be possible within the constraints of an academic environment. Next, a brief analysis of some works that have been proposed for each one of the perspectives identified before is presented.

The emulation of the workplace through distributed projects or cross-university courses was approached and experienced by some authors. The DOSE [7], a Distributed and Outsourced Software Engineering course, followed an approach to teaching distributed software engineering centred in a distributed software development project. They experienced teaching software engineering using a geographically distributed software project involving various countries with different cultures, native languages and time zones. This approach gives the students the opportunity of facing the challenges of distributed software development and helps them understand typical software engineering issues, such as the importance of software requirements specifications, or the relevance of adequate system design. However, they also identify some time scheduling inconveniences, and difficulties in keeping teams committed to their peers. The Undergraduate Capstone Open Source Projects (UCOSP) program [14] ran for ten terms over six years providing for over 400 Canadian students from more than 30 schools. After this period, the authors identified some lessons they had learned: Students work on real distributed open-source projects as full members of software development teams; They use the same software development processes as regular team members and are provided with explicit mentorship from volunteer mentors from each project; Students integrate and apply the skills they have learned in their courses in a real development setting; Students develop and improve their technical communication skills in a real development setting.

A project-oriented course is followed in several software engineering training programmes. Its purpose is to teach students the theoretical and the practical aspects of developing software systems in a team environment giving students a chance to experience a work scenario that is closer to a real-world experience. A Project-Based learning in software engineering Lab, teaching through an e-Portfolio approach is described in [9]. In this approach, the e-Portfolio allows students to carry out a software project, addressing each phase collaboratively with other students and obtaining appropriate feedback from instructors. The e-Portfolio includes a single problem statement for the development of a complete software project comprising of a set of deliverables. To support the implementation, they chose the Moodle Platform. To assess the students' e-portfolios, various rubrics were implemented by scoring and weighting the sections and categories for every deliverable to be evaluated. Another project-based learning approach for teaching software engineering concepts is described in [10]. Their goal is to teach software engineering concepts using the Scrum framework in real life projects. Usually, projects have a capacity of about 1000 person-hours. To make the projects more relevant real customers were incorporated. They bring in requirements from industry and present their topics during a kick-off meeting. During the project, students work together as self- organized teams (5-7 elements). They chose an appropriate project management and team coordination process and they are only asked to use some core tools that are needed to monitor the projects.

A game-based learning methodology of teaching software engineering is presented in [12]. They suggest a methodology of two-fold use of learning games for teaching software engineers. Students, experienced in programming, develop learning games, and then they use the games that are developed for teaching the next generation of students. Students developing games learn the software development life cycle phases including testing, deployment and maintenance, they contact with customers (teachers of corresponding subjects act as customers) and users (students, learning these subjects). In their approach, they find both advantages and disadvantages. As advantages, they identify the increasing students' motivation and revealing their creativity. The main problems observed include difficulty of organization of team work especially for students of early years and lack of time for coordinating them. Schäfer [13] describes some lessons learned after two teaching periods in using scrum with gamification to learn and train the agile principles. They found that their approach has both advantages and disadvantages. Gamification is motivating and helps to bring participants with different backgrounds together in project teams. The game helps in focusing on the project management part and learning the Scrum methodology. As drawbacks, they refer to the importance of having a real external stakeholder or customer defining a project goal externally in a Scrum learning project.

There are different approaches and strategies that may be followed to provide students with the best training in software engineering. All of them agree that the theoretical study of concepts and methodologies should be complemented with hands-on experiences in a software engineering project. This would allow to provide students with a better understanding of the theoretical concepts and to provide them with the non-technical skills in software projects. However, the way different approaches propose to provide the students with the practical experience is very different. Some of them propose to emulate the workplace through distributed projects, which may involve several entities and thus provide interesting experiences in software engineering. Others suggest a project-oriented course where students can practice requirements analysis, project management, development methodologies and teamwork. Another recommendation is using simulations and games to simulate distinct scenarios in software engineering teaching and training. However, regardless of the approach or strategy, it is necessary to understand whether students have acquired the knowledge and skills they need for the performance of their duties, and whether they apply them in their professional activity in software engineering.

III. Overview of our Project-Based Approach for Software Engineering

In this case a project-based approach was adopted for teaching Software Engineering. It is part of a second year of a computer science course (undergraduate course). This is a discipline that has 5 ECTS and whose semester load is 30 hours for theoretical classes and 45 hours for laboratory classes. The focus of the adopted approach was to combine theory and practice. One teacher is responsible for the course management and theoretical lectures. In these classes, the teacher presents the concepts and methodologies and promotes discussion about them. Students are also provided with an introduction to some software development methodologies namely waterfall, Extreme Programming, SCRUM, Spiral, etc. In the assessment, this theoretical part has a weight of 40% for the final grade; the remaining 60% is from the practical component. Another teacher is responsible for the practical classes. In these classes, students acquire some practice of software engineering through the specification, design, implementation and validation of a software application, as a project for teams of 4-6 students. Scrum is the adopted agile software development methodology. The teacher acts as a product owner. Each team member has a specific function (e.g., Scrum Master, Designer, etc.). Each team develops a different project. However, all the projects are focused on the development of a game from a software engineering perspective. This is important to maintain the students motivated and engaged with the project. The first deliverable is revised to accommodate feedback from the product owner. Trello is used for project management and to track progress on tasks.

A. *Additional Realism*

One class of the course has been taught by professionals from software house companies. In this class, software development processes like Feature Driven Development (FDD) and Behaviour Driven Development (BDD) were approached and some of their practical aspects are discussed.

Another important initiative to enable students to get in touch with practice in software engineering is a one-day visit to the premises of another software house company. This company (Outsystems) is well-known for the software development platform they hold and that is used by many software companies worldwide. Their platform is a low-code platform for rapid application development. It is especially designed for developing applications in the context of agile projects. During this journey, students were able to have closer contact with some Scrum activities (namely Daily Scrum, Sprint, Sprint Execution) and contact with some SCRUM Roles (Scrum Master, Development Team). Professionals explain to the students what they are doing, and which technologies and tools are used to support their activities. Students also had a brief session about software cost estimation.

These events are very important since they provide students with the contact and interaction with real software engineering projects with real stakeholders. They help to improve the understanding and the assimilation of the concepts learned in the course.

B. *Student evaluation*

The student evaluation comprises both theoretical and practical evaluation. The theoretical evaluation is a written exam over the course material. The exam consists of 10 questions chosen from the list of 30 questions that were made available to the students at the beginning of the course. This is different from the usual practice on other courses. Most questions are reflexive questions about software engineering subjects. With this approach, the intent is to avoid students wanting to memorise the matters learned along the course period (15 weeks). Also, it is desirable that students learn and acquire knowledge for a long-life period, mainly to be used after graduation on their job integration experience. In section V some gathering data that wants to evaluate results about the achievement to this goal of our approach will be presented.

For the practical evaluation, along the semester, during the 15 working weeks, students´ working teams develop the product on 6 sprints (sprints here are defined as having 2 weeks each). The teacher (i.e. product owner) meets with each team at the end of the sprint to evaluate the work in progress, the achievements and the goals for the next sprint. The team works in class (3h/week) and out of class. Half way through the semester, after sprint 4, and at the end of the semester, after sprint 7, each team has an assessment session were both teachers are present to evaluate different parameters. Some of the parameters are: clear goals, state of the art, requirements (functional and non-functional), software development process (roles, artefacts, timings, hits and misses), team member´s description (roles, skills) task scheduling (monitoring using Trello tool), modelling (user stories), implementation (code), budget (estimated based on the lesson learned during the visit to the company referred to on the previous section of this paper), conclusions (pros and cons) and future work, used literature and citation on the final report, and final presentation and discussion.

One of the achievements that sometimes students realize is learning from mistakes. For instance, if they do not communicate within the team the achieved results are poor, when compared with other more cohesive teams. On the other hand, in collaboration with the “Scrum Master” of the team, a deeper evaluation to eventually gave different grades within the members of the team.

IV. Knowledge Assimilation and Practice

In order to gauge the post-retention cognitive load, a survey of former students was conducted in order to obtain feedback on the importance of the subject to the current professional activity (of those who finished the course and work in the area), and also to know if the knowledge transmitted in the theoretical classes remains. For this last component, the survey included questions that had been already used in the theoretical evaluation of the course. The answers were evaluated with the same evaluation criteria, graded in a scale of 0-20. The questions were selected from the same set of 30 questions referred to in Section III-B. Respondents were informed that the results were for one study and would not be disclosed to third parties. They were asked to respond without recourse to extra help, because what was at issue was whether the concepts and knowledge remained present. The survey was also used to gather insights about the usefulness of the course for the practical life of each graduate. Thus, questions about aspects that may be used in the day to day of their professional activities in the companies where they currently work, were included in the survey.

A. *Survey Description*

The survey was designed to be direct to our objectives and be filled quickly and simply. Some questions were answered in free text (case of questions of theoretical knowledge) and others are multiple choice questions (e.g., used software methodologies). The survey was organized in three parts: Questions about the current professional activity of the respondents; theoretical questions about software engineering; and space for feedback on the importance of topics in their current professional life (for those who had already finished the course).

As examples of questions, we asked if the graduated students are working. If yes, we asked about that actual tasks in their companies (planning, requirements, analysis, design, Code, Quality Control, Tester, Project Management, other), the used methodologies (waterfall, SCRUM, XP, Prototyping, Spiral, FDD, Lean, RUP, other, none). About the theoretical questions we asked about the fundamentals of Software Engineering, Software Quality, Verifications vs Validation, traditional vs Agile, team dimensions and roles, among other questions and feedback.

V. SURVEY RESULTS AND ANALYSIS

This section presents the results gathered in the survey and highlights some of the main findings.

A. *Data Collection/Methodology*

As a universe of respondents, surveys were sent to 97 students. Of these, 56 were undergraduate students (although they had passed in this subject) and 41 graduated.

The survey was done online, using the *LimeSurvey* webtool.

The response rate was of 24.4% of the graduated students and of 21,4% of the undergraduate students.

It is important to note also that some respondents did not answered to all questions.

B. *Results and Analysis*

Figure 1 shows the activities the respondents are involved in in their work. 84% of the respondents are involved in more than one activity. 50% of them are involved in planning, analysis and testing but they are not involved in implementation.

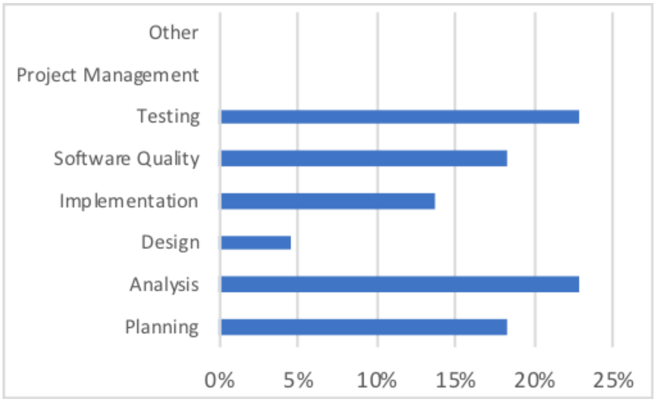

Figure 1. Activities carried out.

Students were also asked to identify the software development methodologies they use in their activities. They were able to identify the methodologies they use considering a list of given methodologies. Results are presented in Figure 2.

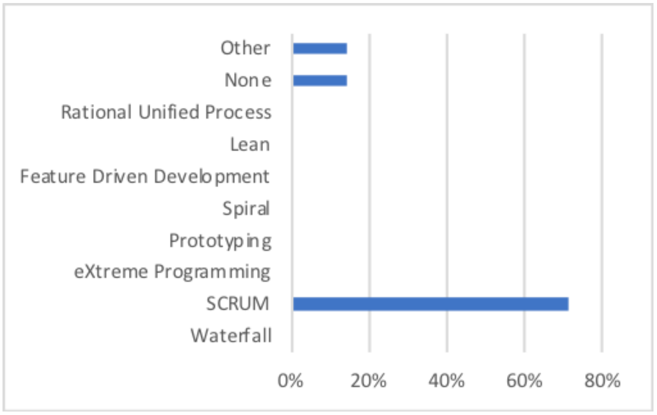

Figure 2. Software development methodologies.

More than 70% of the respondents refer that they use the Scrum methodology. This appears to be in line with the results presented in the “12th annual State of Agile report” [15] that refer that 52% of respondents stated that more than half of the teams in their organizations are using agile practices. And it is also in accordance with the results presented in another survey of more than 2,000 active Scrum and Agile practitioners [16]. This study refers that 94% of agile users use the Scrum approach in their agile practice (78% use Scrum with other approaches).

With respect to the importance of the subjects learned in the course, 87.5 percent, of the 8 graduated students that respond to this question, said that the content learned in the course has been considerably useful for their actual professional activity (see Figure 3).

The second part of the survey was related to theoretical questions about software engineering. This part was evaluated in a 0-20 scale and we compare these results with the results achieved by the same individual during the course. We consider the individual "maintained" if *(grade achieved in the course -1.5 ≤ grade achieved in the survey ≤ (grade achieved in the course +1.5).*

After evaluating the answers to the questions, we conclude that there is a majority (58%) that has maintained or increased the result (41% maintained, 17% increased) (see Figure 4).

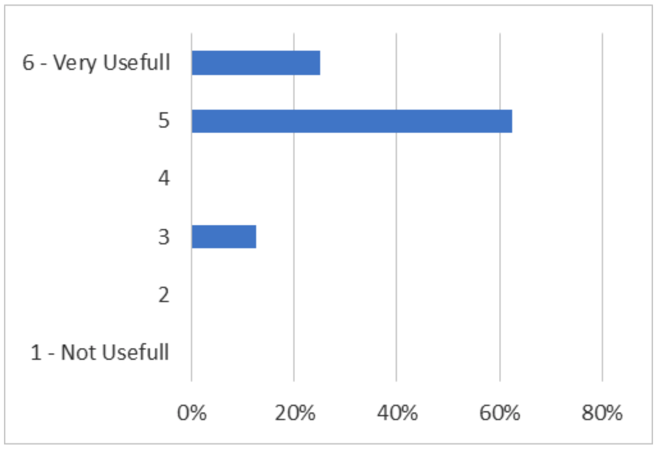

Figure 3. Course content vs profissional activity.

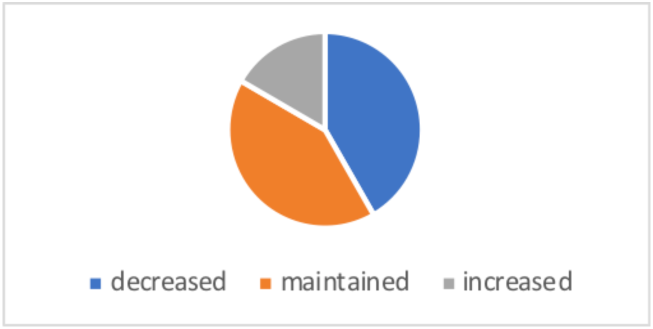

Figure 4. Grades evolution (not graduated respondents).

In the case of students already graduated, the results, presented in Figure 5, are better (less cases (37,5%) of lowering grades). Despite the long period of time after they attend the course, this is probably a consequence of the practical experience they get in the field of software development.

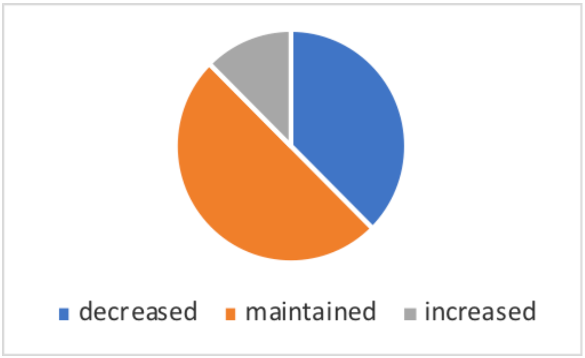

Figure 5. Grades evolution (graduated respondents).

VI. Lessons Learned and Challenges Faced

The contributions of this paper are in the form of the lessons learnt, which may be seen as guidance for others looking to approximate the know-how of students to the methods and techniques used by software companies. In summary, these are:

- Students should learn by doing and, wherever possible, software engineering principles should be assessed in the context of practical work, rather than by regurgitating material taught or extracted from text books.
- Students must have well defined and known goals. The assessment of the theoretical subjects does not need to be a surprise in the exam.
- Opening classes to external stakeholders (by promoting talks or visiting companies) during the last part of the semester helps students to reinforce knowledge (some of which are not in books) and motivate them to the subjects.
- It is very important to get feedback from past students and evaluate if the transmitted concepts and knowledge are still there, and if it was improved by the work experience in the labour market.
- It is important to choose projects that are of interest to the students and that can motivate them and involve them in their development. Projects that are related to games development can be very interesting.

However, during our experience, we faced challenges like:

- Difficulty to maintain all team members equally motivated and engaged in the same way throughout the entire project development period;
- Keeping all students involved in the project. Some students may drop out, leaving the team during the semester, and affecting the workflow and scheduling of the remaining members of the team;
- Allowing students to experience various roles within the team. It is necessary to find a way to rotate the roles of each one within the team, to avoid each student being too focused on just one role. It is important that everyone experiences a diversity, as broad as possible, of different roles;
- Allowing students to experience different methodologies in real environments. More field trips and contact with companies that use different methodologies, must be promoted to foster more diversity of experiences.

VII. Conclusions and Future Work

Our survey was the starting point of a reflexion about the impact of the approach followed in previous years in the course of Software Engineering. Based on the results, we think that allowing students to know the pool of questions in advance, fosters the students on important knowledge in the field and to understand these items, that we want students to maintain over a long period of time.

In future, the pool of questions will be increased to improve the effect of randomisation for the next exams. As for the practical component, based on the results, Scrum is still used as a case study since it is one of the most used processes by companies where our graduated students work. We will work to increase the number of respondents on the survey. Also, in future we will also extend and analyse data from a survey done to the employees of our graduated students and reach more feedback to improve and actualize the contents of this course.

REFERENCES

[1] R. Chatley and T. Field, "Lean Learning: Applying Lean Techniques to Improve Software Engineering Education," in *Proceedings of the 39th International Conference on Software Engineering: Software Engineering and Education Track*, 2017, pp. 117–126.

[2] S. D. Zorzo, L. de Ponte, and D. Lucrédio, "Using scrum to teach software engineering: A case study," in *2013 IEEE Frontiers in Education Conference (FIE)*, 2013, pp. 455–461.

[3] M. Kuhrmann and J. Münch, "Enhancing Software Engineering Education Through Experimentation: An Experience Report." 2018.

[4] C. Amrit, M. Daneva, and D. Damian, "Human factors in software development: On its underlying theories and the value of learning from related disciplines; A Guest Editorial Introduction to the Special Issue," *Inf. Softw. Technol.*, vol. 56, no. 12, pp. 1537–1542, 2014.

[5] S. Beecham, T. Clear, D. Damian, J. Barr, J. Noll, and W. Scacchi, "How Best to Teach Global Software Engineering? Educators Are Divided," *IEEE Softw.*, vol. 34, no. 1, pp. 16–19, 2017.

[6] F. Matthes, C. Neubert, C. Schulz, C. Lescher, J. Contreras, R. Laurini, B. Rumpler, D. Sol, and K. Warendorf, "Teaching Global Software Engineering and International Project Management - Experiences and Lessons Learned from Four Academic Projects," *3rd Int. Conf. Comput. Support. Educ. CSEDU 2011*, p. 12, 2011.

[7] M. Nordio, C. Ghezzi, B. Meyer, E. Di Nitto, G. Tamburrelli, J. Tschannen, N. Aguirre, and V. Kulkarni, "Teaching Software Engineering Using Globally Distributed Projects: The DOSE Course," in *Proceedings of the 2011 Community Building Workshop on Collaborative Teaching of Globally Distributed Software Development*, 2011, pp. 36–40.

[8] D. Dahiya, "Teaching Software Engineering: A Practical Approach," *SIGSOFT Softw. Eng. Notes*, vol. 35, no. 2, pp. 1–5, 2010.

[9] J. A. Macias, "Enhancing Project-Based Learning in Software Engineering Lab Teaching Through an E-Portfolio Approach," *IEEE Trans. Educ.*, vol. 55, no. 4, pp. 502–507, 2012.

[10] A. Heberle, R. Neumann, I. Stengel, and S. Regier, "Teaching agile principles and software engineering concepts through real-life projects," in *2018 IEEE Global Engineering Education Conference (EDUCON)*, 2018, pp. 1723–1728.

[11] M. Yampolsky and W. Scacchi, "Learning Game Design and Software Engineering Through a Game Prototyping Experience: A Case Study," in *Proceedings of the 5th International Workshop on Games and Software Engineering*, 2016, pp. 15–21.

[12] O. Shabalina, N. Sadovnikova, and A. Kravets, "Methodology of teaching software engineering: Game-based learning cycle," *Proc. - 2013 IEEE 3rd East. Eur. Reg. Conf. Eng. Comput. Based Syst. ECBS-EERC 2013*, pp. 113–119, 2013.

[13] U. Schäfer, "Training scrum with gamification: Lessons learned after two teaching periods," in *2017 IEEE Global Engineering Education Conference (EDUCON)*, 2017, pp. 754–761.

[14] R. Holmes, M. Craig, K. Reid, and E. Stroulia, "Lessons Learned Managing Distributed Software Engineering Courses," in *Companion Proceedings of the 36th International Conference on Software Engineering*, 2014, pp. 321–324.

[15] VersionOne Inc, "12th annual State of Agile report," 2018.

[16] Scrum Alliance, "STATE OF SCRUM 2017-2018. Scaling and agile transformation.," 2017.