International Journal of Engineering Education Vol. 36, No. 1(B), pp. 309–316, 2020
Printed in Great Britain

0949-149X/91 $3.00+0.00

Students' Selection of Teamwork Tools in Software Engineering Education: Lessons Learned*

RICARDO COLOMO-PALACIOS and TERJE SAMUELSEN
Østfold University College, B R A Veien 4, 1783 Halden, Norway. E-mail: ricardo.colomo-palacios@hiof.no, terje.samuelsen@hiof.no

CRISTINA CASADO-LUMBRERAS
Universidad Complutense de Madrid, Campus de Somosaguas. 28223 – Pozuelo de Alarcón Madrid, Spain. E-mail: crcasado@ucm.es

XABIER LARRUCEA
Tecnalia, Parque Tecnológico de Bizkaia. Calle Geldo, Edificio 700, E-48160 Derio, Bizkaia, Spain. E-mail: xabier.larrucea@tecnalia.com

Software work is normally developed in groups. As a result, there is a need to develop teamwork competence in related activities such as, software engineering education. In higher education educational settings, courses often propose several tools for student groups to either guide or support their work. In this paper, authors present main results and lessons learned from courses on software engineering. Specifically, the aim of this paper is the study of the selection and the adoption of software engineering tools by students working in teams, in the context of a software engineering course. The purpose of the study is analysing the students' decision making process and reasoning strategies to such selection. In this scenario, driven by a project based learning approach, a qualitative study on the use of specific tools to support group work was conducted. Results reveal that students demonstrate a rational decision making process based on logical efficient reasoning. As consequence six lessons have been learned: *everything in one place; the new over the known; freedom over imposition; performance and freedom; social influence and secondary role of project management*. In addition, these six lessons have been compared with previous literature in the topic and backing them up with main theories in the field. Finally, authors reflect on the implications of such lessons learned analyzing deeply aspects like freedom of choice, performance, tools features, imposition and social influence, to bring a set of grounded argumentations to the reader.

Keywords: software engineering education; teamwork; software engineering tools; decision-making; qualitative study, freedom of choice

1. Introduction

Software is pervasive in our life. As a result, software engineering has been highlighted as a key to support the digitalization of society. Software production is, in its nature, different from other production processes and its development is highly dependent on human factors [1]. Software is normally produced in teams [2] and more complex structures include teams of teams [3] or distributed teams [4], [5]. While agile approaches tried to simplify teams [6], it is still true that team management is a key aspect for software practitioners and researchers alike [7].

Teamwork in software engineering scenarios have evolved from a setup in which, team members shared the same room or building to a new approach in which, by means of global software development, development is distributed across cultures, time zones and continents. Nowadays, a big portion of the software is produced offshore or by a mix of inshore and offshore teams.

Software engineering education is heavily based on the use of tools [8]. However, the problem we actually face in software engineering education, is the wide panoply of tools available. Tools are providing ways to automate software process [9], but also to alleviate some of the tensions present in software development projects. Literature, reported the gap between the needs of industry and education in software engineering [10, 11]. This gap is also a fact in terms of tools used. Given the huge amount of tools available, literature proposed ways to measure the acceptance of software engineering tools in academic education [8]. In this work, authors apply the Unified Theory of Acceptance and Use of Technology (UTAUT) [12] model for the analysis of the adoption process. However, this is not the only model valid for these purposes and alternative models include Technology Acceptance Model (TAM) [13], TAM2 [14], TAM3 [15], Innovation Diffusion Theory [16] and Theory of Planned Behaviour [17], naming just a few of the most important models in the area.

The aim of this paper is the study of the selection and the adoption of software engineering tools, in the context of a software engineering course, by students. Results are aimed to illustrate the aspects regarding teamwork and performance, connected with aspects like freedom of choice and features. Finally, a set of recommendations for instructors is given to the reader.

The remainder of this paper is structured as follows. Section 2 presents main curricular efforts

* Accepted 19 November 2019.

but also overviews scientific literature, reporting main studies on teamwork in software engineering education. Section 3 introduces the experience and the case study considered to draw lessons learned from. Furthermore, six different lessons learned are presented by authors in Section 4, comparing also the main results with relevant literature. Section 5 depicts main limitations of this work. Finally, section 6 wraps-up the paper and proposes future work.

2. The Role of Teamwork in Software Engineering Education

Teamwork is considered as a fundamental transferable skill to new undergraduate programs adapted to the Bologna Declaration or ABET criteria. Teamwork is a traditional topic in computing education and more precisely in Software Engineering education. Not in vain, according to a recent study [18], one of the main active area of software engineering research is teamwork and collaboration, being one of the central actions for Software Engineering research [19] in several application areas and setups [20, 21].

The study of the topic started in the morning of the discipline in the late sixties and is present in early literature of the topic e.g., [22]. In recent times, the guide to the software engineering body of knowledge (SWEBOK) [23] stated that software practitioners must be able to interact cooperatively and constructively with colleagues to first determine, and then meet both needs and expectations.

Teamwork is present also in the software engineering body of skills (SWEBOS) [24]. SWEBOS states that ''Software engineers are capable of cooperating with others in a team.'' Teamwork is part of the competencies for the professional collaboration with others being then a core element of software engineering practice.

The current and more global initiative in computing curricula is the one leaded by the Joint Task Force on Computing Curricula (IEEE Computer Society & Association for Computing Machinery). Curricular efforts include publications in four different sub-disciplines under the umbrella of Computing, namely, Computer Science, Computer Engineering, Information Technology and Software Engineering. However, in computing curricula 2005 [25], the overview report of the whole joint task includes also the Association for Information Systems that includes also Information Systems as the fifth volume. Lately, an in the Post-Secondary degree, a new recommendation was issued for Cybersecurity. Focusing on the Software Engineering recommendation, the last version of the document was issued back in February 2015 [26]. In this work, editors report a previous study performed back in 2013, in which, software employers were asked to rate the importance of candidate qualities and results show that, after communication skills, ability to work in teams is the second most desired characteristic for employers. As a result of its importance, dynamics of working in teams is one of the Professional Practice factors, more specifically PRF.psy.1. But, apart from that, aspects on working with teams are present in process concepts for both software engineering process improvement (PRO.con.3) and individual and team software process (PRO.imp.3)

As stated before, teams in software engineering are key in the study of the discipline, and literature has reported a good set of studies on the topic. Focussing just on studies that report aspects on teamwork assessment in courses, efforts devoted to study teamwork, by means of learning analytics, have been reported [27, 28], artificial intelligence based techniques to assess and predict teamwork results in classes [29] or assign people to groups [30, 31], monitoring artefacts like reflexive weekly monitoring [32] or lean techniques [33] to improve learning experience and results, or the use of tools (GitHub, SonarQube) to support software engineering courses [34], naming just a few of the initiatives conducted and reported in recent years in the literature.

However, to the best of authors' knowledge, there is not a paper reporting the effects of freedom of choice of tools in the software education arena, focusing the results on the effects of these choices on teamwork. This paper is aimed to fill the gap. To do so, a case study will be presented and analysed in the following sections.

3. Case Study

3.1 The Course

In this section, authors present the setup of the case study. Experience took place in the spring semester (January-May) in the 2017–2018 academic year at the Faculty of Computer Sciences, Østfold University College, Norway. Østfold University College is a public higher education institution that offers four bachelor programmes in the broad field of computing, namely, Digital Media Production, Computer Engineering, Computer Science and, finally Information Systems. The course Software Engineering is available for the latter two bachelor studies and covers 10 ECTS. The requisite to enrol the subject is a previous course on Object-oriented Programming. It is programed for the second year of studies of the three that all bachelor programs have in the institution. The course present four hours of lec-

tures and two hours devoted to group efforts and workshops per week.

The content of the course includes aspects like life-cycle models, software specifications, object-oriented design using UML, development methods, software documentation and static and dynamic testing.

The final grade is calculated on the basis of two partial exams. Students must pass each partial exam in order to pass the whole course.

- Partial exam 1 is a group project worth 45%. Individual grades are awarded.
- Partial exam 2 is an individual written exam lasting 3 hours and worth 55%. No support materials are permitted.

As a part of the contents, there are some classes and practices on software project management adopting the agile approach. Aspects covered by students include:

- Team participation.
- Effort estimation by means of agile approaches.
- User stories.
- Epics.
- Story points.
- Sprints.
- Issues.
- Retrospectives.

Students working in groups were asked to define epics, sprints and break down work items into granular pieces and estimate by means of story points (abstract measure of effort required to implement a user story) as well as maintaining a product backlog of the project developed in group that is meant to gather all aspects in the course (specifications, design, development and testing). All these aspects were related to a project defined by teachers at the beginning of the course.

In order to support the set of tasks, teachers recommended the use of Taiga.io. Taiga is a free open source agile project management online tool developed as incubator by Kaleidos Open Source. It was awarded back in 2015 as the best agile tool in the Agile Awards and is also a recognized tool in the open source community. It is among the top 10 open source projects 2014 and among top 11 project management tools 2016 both by opensource.com. The tool covered in a perfect way all requirements expressed by teachers with regards of the course. Teachers asked students to grant them view access to their Taiga project in order to monitor the advancement of the developments and to assess the final work performed in groups, including teamwork aspects. Although the use of the tool was considered somehow easy, a part of a class was devoted to show the tool and explain the main aspects of it. Apart from this session, another mentoring session was scheduled to guide students in a more personal way working in timeslots with groups.

Two student assistants (master student in first and second year and former students) helped the students to compete their tasks constituting a first line of information to them.

3.2 The Process

The course started as planned and lectures were given to students in due time. However, after some days, initial suggestions started to appear. Complains about the mandatory use of Taiga appeared and around half of the class putted pressure on student assistant to inform teachers about their will to use any tool at their will to develop their work as intended. They were not complaining about the work to be done or the amount of it, but simply about the tool to do it. Their suggestion was using GitHub instead.

GitHub includes a fully functional Kanban board based on issues, but cannot be considered a full project management tool as Taiga is. For instance, epics are not covered and some of the agile concepts present different names, that is, sprints (milestones), tasks (issues) and boards (projects). This is the reason why tools built on the top of GitHub appear. For instance, ZenHub is a project management tool that integrates natively within GitHub's user interface. It is a free for all open source projects making the use of it closer to commercial tools like Trello or Jira and implementing key aspects of the course including epics, estimates and reports. ZenHub was offered also to students as a valid option for the project.

However, in spite of the suitability of the tool, no group adopted ZenHub as a midway option, taking advantage of GitHub as the tool they already knew but also of the genuine implementation of agile concepts on the top of GitHub.

From the twelve groups in the course, six of them adopted Taiga, the recommended tool, as the platform to guide their efforts in the course and the remaining six chose GitHub for this purposes. This paper is devoted to analyse the reasons behind these decisions and the results in terms of final results and teamwork.

3.3 The Method

The findings discussed in this paper are based on a case study conducted in the course. The research reported is a part of an exploratory study. Findings in the form of lessons learned are reported based on a set of semi-structured interviews conducted with student assistants and students alike. All interviews were conducted at Østfold University College. The

conversations were not recorded but documented during the interview.

The analysis of the interviews was carried out using NVIVO 11 software. This tool is used to organize, classify and analyse information, but also to explore and review trends in both studies. Besides, it permits to establish connections among content as well to extract conclusions from data. Aspects in section 4 are derived from the analysis performed. In spite of the exploratory nature of the study, widely accepted guidelines for conducting research in the area have been adopted and adapted by authors [35].

4. The Results: Lessons Learned

In this section, authors present main conclusions derived from the case study. Lessons learned papers are pervasive in software engineering discipline e.g., [36, 37] and also in software engineering education e.g., [38, 39]. As stated before, lessons learned are based on the analysis of the case study. Lessons learned are as follows:

4.1 All In One Place: Expected Performance and Effort

GitHub is a web-based, social software development environment and code management platform. Its popularity has been increasing in a dramatic way in recent years. GitHub is open-source, free to use, collaboration-oriented tool providing features like task management, access control, bug tracking and feature requests, focusing just on a handful of them. In recent years, literature has reported hundreds of initiatives to use GitHub as an educational tool. In the case reported here, it was not intended to use GitHub for the specific purposes of project management, but students reported several times ''It's better to have everything in one place'' and ''We do not need to use a different tool to do the work, no matter we need to perform some tasks to support the project management process, but this is not a problem''. Other aspect students underlined was ''GitHub is the standard''. Asked about the possibility to use ZenHub, students agreed that ''pure'' GitHub simply works, ''no need to make it difficult''. It is worth to underline that none of the teams adopted to mid-way ZenHub option. Students also underlined they do not want to use two kind of environments, ''all was in GitHub including documentation, coding. . .''. On the other hand, half of the teams chose Taiga. Some of them simply liked it, although others, in spite of the adoption, complain about the set of tools they need to learn in their studies.

The lesson learned by teachers was clear: Integrated tools present higher acceptation rates in software academic education. Agreeing with [8], behavioural intention to use the tool is depending directly on factors like performance expectancy, effort expectancy and diffusion. Here first two factors combined lead to the conclusion that higher performance and less effort is expected in integrated tools compared to isolated options.

4.2 The New over the Known: it is about Features

GitHub was not a new tool for the students. In fact, both ZebHub and Taiga were new to students. Classic acceptance models include experience as one of the main factors to consider in technology acceptation [40]. This aspect was also reported by students in an extensive way. Although it is not surprising, it is interesting to note that ZenHub is a tool that covers both worlds, on the one hand, agile project management and on the other hand, GitHub as ecosystem. The acceptance of the tool was zero. Authors connect this fact with the lack of importance that students give to project management (lesson learned 6). This can be the reason behind the low acceptance of ZenHub. In this case, the lesson learned is the necessity to justify the need to adopt new tools by means of their features. This will increase performance expectancy and decrease effort expectancy compared to generic tools making.

4.3 Freedom over Imposition

In spite educational programmes must be accredited and supervised by educational authorities, the tools that support students' learning are not normally coded in syllabus. Literature reported the trend among students to choose tools they normally use over imposed options [41]. Although no respondent reported any comment on this topic, during the course, several comments to teachers from student assistants were demanding more autonomy. This case has also been reported in the case of practitioners by literature [42]. This need for freedom can be linked to modern approaches of personalised learning environments that can be connected also to tool selection e.g., [43, 44].

Main lesson learned here is the need to embrace an open approach in teaching activities: focus on concepts and not on tools.

4.4 Performance and Freedom

Maybe the most important finding in this exploratory study is the fact that groups who chose GitHub performed better than the ones who used Taiga in the course as a whole. However, in the specific aspects of project management, that are not the only ones present in the course, there are not important differences among groups. There is a need to isolate factors that leaded to these differ-

ences in performance, studying, for instance, previous work together and preceding performance, but initially and, in our case, freedom leads to higher performance. As it was reported by students, there was not a need to learn internals for Taiga and aside tasks not included in GitHub (e.g., story points and epics) were easy going for them to implement in GitHub settings. This is the only reported factor on the connection of performance and tools by students.

Freedom avoids unnecessary friction in agile scenarios as reported widely in the traditional agile literature. This is backed up with several papers highlighting the aspiration to freedom by software developers in their work [45]. Team performance in software teams is depending on a set of factors [46] namely, team coordination, goal orientation, team cohesion, shared mental models and team learning. However, clan control in the group level and self-control in the individual are defined over freedom [42] and they influence previous listed factors in a direct way. Again, authors are unable to isolate factors, but it is reasonable to think that freedom is affecting group performance in a positive way. Other possibility to think about is the positive connection of overall performance with GitHub proficiency. One can think that mastery in GitHub can be associated with a good competence in other professional aspects present in the course, analysis and design, for instance. Finally, there are arguments that could suggest that the students who chose GitHub took more time in other tasks and less time in learning new tools, therefore increasing their overall efficiency.

Taking all these reasons into account, main lesson learned in this aspect is that, in choosing software tools, one size does not fit all. Adoption is depending in a set of circumstances and previous knowledge is the key. Tool selection can harm final performance comparing in some cases.

4.5 The Helping Hand: Social Influence

Student assistants were crucial in the course. They provide mentoring to groups and guide them in the use of tools. According to the information provided by them, students proposed GitHub almost the first day of the course as the project management tool. In spite of this, student assistants tried initially to convince students to use Taiga. After two weeks of pale results in their duties, they escalated the problem to teachers. Student assistants agreed that their influence was very limited in tool selection. One even reported that, he was using also GitHub as a project management tool in his previous industrial experience, exactly what students were doing in the course. Although there is no reported evidence in the literature, working habits in industry by students could also lead to this situation. There is no evidence on the influence of student assistants on the students, however, it seems reasonable to infer that such influence on this aspect could be limited. Finally, another aspect to take into account is the fact that one of the student assistants reported the fact that GitHub was his project management tool. Maybe his belief on the need to adopt a different perspective was not too high, and as a consequence of this, his credibility and influence as also scarce.

Authors want to link social influence with the existing gap between software engineering education and industrial needs underlined pervasively in the literature [47]. Students may detect this gap (in their studies overall and not just in the course) and the influence of the student assistant with relevant and recent experience in the industry is giving way to question tool suggestion. In this case, the relevant experience in industry presented by instructors is not compensating the influence of the student assistant.

Counting on with student assistants is of great help in courses, however, authors underline the need to ensure the proper knowledge on methods and concepts and their mapping on available tools.

4.6 Project Management is the Cinderella

Although in general, students value the importance and benefits of project management activities in software development initiatives, they see project management activities as secondary. Although project management community agreed on the need to invest in project management training and education [48], literature reported the vision of project management as secondary among students [34], in spite of its importance [49]. In our case, students were concentrated on core tasks of software development and project management was seen just as a support activity. Authors, as a lesson learned, believe academics must underline the importance of monitoring and control activities to support activities in software development projects. This must be done specially in first courses in undergraduate level. As the student progresses in the bachelor path, aspects like project management becomes crucial in capstone projects [50, 51] as well as in the professional practice.

5. Limitations

The approach adopted to study the phenomenon was qualitative. That is the reason why validity threats are analysed from this viewpoint. Validation in qualitative research is rather ambiguous and contentious compared to quantitative approaches [52]. Authors, following the works by [53], analysed different types of validity threats, namely, credibil-

ity, transferability and confirmability. Authors believe this set of limitations are enough to justify threats of validity in an exploratory study.

Concerning credibility, this involves establishing that the results are believable from the perspective of the participants in the research to convincingly rule out alternative explanations. Authors ponder that, this is a main limitation of the work performed. In order to improve this aspect, a holistic approach must be adopted including a wide variety of data sources and a mixture of data collection techniques (e.g., triangulating data) to build sound results. However, it is also true that the nature of this work is just exploratory.

Regarding transferability, which is related to the generalisability of research findings, two threats are anticipated. The first is the limited number of subjects. Although this threat exists, making difficult the generalization of results, it is also true that the case study is representative enough for an exploratory study. The second threat is entrenched in the circumstance that the sample was not taken in a random way. It is assumed that generalisation of results is not guaranteed, however replication is also possible in similar circumstances.

To conclude, confirmability can be defined as the degree to which the results could be corroborated by others. Authors adopted the guidelines provided by [54] to improve summaries and notes to bring transparency to the overall process. However, it is also true that this paper can be seen as an interpretive research, and because the presupposition of relative objectivity, this aspect is limited by nature. To improve this factor, reflexibility (relative transparency of the researcher) has been maximized by means of reflexion and bias disclosure.

6. Conclusions and Future Works

Teamwork is a main aspect for software practitioners. Software is normally developed in teams and students must be encouraged to work in teams to produce software. In this paper, an initiative to analyse the effects of choice of tools on performance and team work within software engineering education is presented. Adopting a qualitative approach, authors present a set of lessons learned, classified into six different groups, namely, integration of tools, imposition over freedom, knowledge of the tool, freedom and performance, the role of student assistants and, finally, the importance of software project management. This set of lessons learned could not only be of software engineering teacher's interest in the definition of syllabi and in the design of course tools, but also to provide guidelines to student assistants in their duties. Results show the need to adopt integrated tools in teaching and in professional environments, the necessity to justify tools adoption by means of their features, the requirement to focus in concepts and not in tools no matter how accurate they are, the need to consider previous knowledge on tool selection, select student assistants and other teachers ensuring they present previous knowledge on the set of tools, and, finally, the necessity to underline to students the need to monitor and control their activities.

Future works will be twofold. Firstly, a first branch of studies could be devoted to the study of the phenomenon by means of a cultural perspective to compare results in different countries and cultures. Secondly, it is aimed to perform a comparison between practitioners and students' views in the election of software engineering tools and its effects on performance and team work.

References

1. R. Colomo-Palacios, C. Casado-Lumbreras, P. Soto-Acosta, S. Misra and F. J. García-Peñalvo, Analyzing human resource management practices within the GSD context, *J. Glob. Inf. Technol. Manag.*, **15**(3), pp. 30–54, 2012.
2. S. Kudaravalli, S. Faraj and S. L. Johnson, A Configural Approach to Coordinating Expertise in Software Development Teams, *MIS Q*, **41**(1), pp. 43–64, Mar. 2017.
3. D. Šmite, N. B. Moe, A. Šāblis and C. Wohlin, Software teams and their knowledge networks in large-scale software development, *Inf. Softw. Technol.*, **86(**Supplement C), pp. 71–86, Jun. 2017.
4. J. Kroll, I. Richardson, R. Prikladnicki and J. L. N. Audy, Empirical evidence in follow the Sun software development: A systematic mapping study, *Inf. Softw. Technol.*, **93**, pp. 30–44, Jan. 2018.
5. N. Rashid and S. U. Khan, Agile practices for global software development vendors in the development of green and sustainable software, *J. Softw. Evol. Process*, **30**(10), p. e1964, Oct. 2018.
6. R. Hoda, N. Salleh and J. Grundy, The Rise and Evolution of Agile Software Development, *IEEE Softw.*, **35**(5), pp. 58–63, Sep. 2018.
7. A. Heredia, R. Colomo-Palacios, and P. Soto-Acosta, Tool-supported continuous business process innovation: a case study in globally distributed software teams, *Eur. J. Int. Manag.*, **11**(4), pp. 388–406, 2017.
8. S. Wrycza, B. Marcinkowski and D. Gajda, The Enriched UTAUT Model for the Acceptance of Software Engineering Tools in Academic Education, *Inf. Syst. Manag.*, **34**(1), pp. 38–49, Jan. 2017.
9. R. V. O'Connor, P. Elger and P. M. Clarke, Continuous software engineering – A microservices architecture perspective, *J. Softw. Evol. Process*, **29**(11), p. e1866, 2017.
10. G. Subrahmanyam, A Dynamic Framework for Software Engineering Education Curriculum to Reduce the Gap between the Software Organizations and Software Educational Institutions, in *2009 22nd Conference on Software Engineering Education and Training*, pp. 248–254, 2009.

11. D. M. C. Nascimento, R. A. Bittencourt and C. Chavez, Open source projects in software engineering education: a mapping study, *Comput. Sci. Educ.*, **25**(1), pp. 67–114, Jan. 2015.
12. V. Venkatesh, M. G. Morris, G. B. Davis and F. D. Davis, User Acceptance of Information Technology: Toward a Unified View, *MIS Q.*, **27**(3), pp. 425–478, 2003.
13. F. D. Davis, R. P. Bagozzi and P. R. Warshaw, User Acceptance of Computer Technology: A Comparison of Two Theoretical Models, *Manag. Sci.*, **35**(8), pp. 982–1003, Aug. 1989.
14. V. Venkatesh and F. D. Davis, A Theoretical Extension of the Technology Acceptance Model: Four Longitudinal Field Studies, *Manag. Sci.*, **46**(2), pp. 186–204, Feb. 2000.
15. V. Venkatesh and H. Bala, Technology Acceptance Model 3 and a Research Agenda on Interventions, *Decis. Sci.*, **39**(2), pp. 273–315, May 2008.
16. G. C. Moore and I. Benbasat, Development of an Instrument to Measure the Perceptions of Adopting an Information Technology Innovation, *Inf. Syst. Res.*, **2**(3), pp. 192–222, Sep. 1991.
17. I. Ajzen, The theory of planned behavior, *Organ. Behav. Hum. Decis. Process.*, **50**(2), pp. 179–211, Dec. 1991.
18. J. DeFranco and P. Laplante, A software engineering team research mapping study, *Team Perform. Manag. Int. J.*, **24**(3–4), pp. 203–248, Mar. 2018.
19. K.-J. Stol and B. Fitzgerald, The ABC of Software Engineering Research, *ACM Trans. Softw. Eng. Methodol. TOSEM*, **27**(3), p. 11, Aug. 2018.
20. E. Klotins, M. Unterkalmsteiner and T. Gorschek, Software Engineering Anti-patterns in start-ups, *IEEE Softw.*, **36**(2), pp. 118–126, 2019.
21. D. Lopez-Fernandez, L. Raya, F. Ortega and J. Jesus Garcia, Project Based Learning Meets Service Learning on Software Development Education, *Int. J. Eng. Educ.*, **35**(5), pp. 1436–1445, 2019.
22. S. Smith, M. Mannion and C. Hastie, Encouraging the development of transferable skills through effective group project work, *WIT Trans. Inf. Commun. Technol.*, **12**, 1970.
23. A. Abran and D. Fairley, *SWEBOK: Guide to the software engineering Body of Knowledge Version 3*. IEEE Computer Society, 2014.
24. Y. Sedelmaier and D. Landes, Software engineering body of skills (SWEBOS), in *2014 IEEE Global Engineering Education Conference (EDUCON)*, pp. 395–401, 2014.
25. R. Shackelford et al., Computing Curricula 2005: The Overview Report, in *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education*, New York, NY, USA, pp. 456–457, 2006.
26. M. Ardis, D. Budgen, G. W. Hislop, J. Offutt, M. Sebern and W. Visser, SE 2014: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering, *Computer*, **48**(11), pp. 106–109, Nov. 2015.
27. M. A. Conde, R. Colomo-Palacios, F. J. García-Peñalvo and X. Larrucea, Teamwork assessment in the educational web of data: A learning analytics approach towards ISO 10018, *Telemat. Inform.*, **35**(3), pp. 551–563, Jun. 2018.
28. Á. Hernández-García, E. Acquila-Natale, J. Chaparro-Peláez and M. Á. Conde, Predicting teamwork group assessment using log data-based learning analytics, *Comput. Hum. Behav.*, **89**, pp. 373–384, Dec. 2018.
29. D. Petkovic et al., Using the random forest classifier to assess and predict student learning of Software Engineering Teamwork, in *2016 IEEE Frontiers in Education Conference (FIE)*, pp. 1–7, 2016.
30. M. A. Paredes-Valverde, M. del P. Salas-Zárate, R. Colomo-Palacios, J. M. Gómez-Berbís and R. Valencia-García, An ontology-based approach with which to assign human resources to software projects, *Sci. Comput. Program.*, **156**, pp. 90–103, May 2018.
31. D. Dzvonyar, D. Henze, L. Alperowitz and B. Bruegge, Algorithmically supported team composition for software engineering project courses, in *2018 IEEE Global Engineering Education Conference (EDUCON)*, pp. 1753–1760, 2018.
32. M. Marques, S. F. Ochoa, M. C. Bastarrica and F. J. Gutierrez, Enhancing the Student Learning Experience in Software Engineering Project Courses, *IEEE Trans. Educ.*, **61**(1), pp. 63–73, Feb. 2018.
33. R. Chatley and T. Field, Lean Learning – Applying Lean Techniques to Improve Software Engineering Education, in *2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering Education and Training Track (ICSE-SEET)*, 2017, pp. 117–126.
34. C. Raibulet and F. Arcelli Fontana, Collaborative and teamwork software development in an undergraduate software engineering course, *J. Syst. Softw.*, **144**, pp. 409–422, Oct. 2018.
35. P. Runeson and M. Höst, Guidelines for conducting and reporting case study research in software engineering, *Empir. Softw. Eng.*, **14**(2), pp. 131–164, Dec. 2008.
36. A. Nowak and H. J. Schünemann, Toward Evidence-Based Software Engineering: Lessons Learned in Healthcare Application Development, *IEEE Softw.*, **34**(5), pp. 67–71, 2017.
37. J. G. Guzmán, A. F. del Carpio, R. Colomo-Palacios and M. V. de Diego, Living Labs for User-Driven Innovation: A Process Reference Model, *Res.-Technol. Manag.*, **56**(3), pp. 29–39, May 2013.
38. M. V. Palacin-Silva, A. Seffah and J. Porras, Infusing sustainability into software engineering education: Lessons learned from capstone projects, *J. Clean. Prod.*, **172**, pp. 4338–4347, Jan. 2018.
39. M. Niazi, Teaching global software engineering: experiences and lessons learned, *IET Softw.*, **9**(4), pp. 95–102, 2015.
40. V. Venkatesh, J. Y. L. Thong and X. Xu, Consumer Acceptance and Use of Information Technology: Extending the Unified Theory of Acceptance and Use of Technology, *MIS Q.*, **36**(1), pp. 157–178, 2012.
41. V. Stantchev, R. Colomo-Palacios, P. Soto-Acosta and S. Misra, Learning management systems and cloud file hosting services: A study on students' acceptance, *Comput. Hum. Behav.*, **31**, pp. 612–619, Feb. 2014.
42. T. Dreesen and T. Schmid, Do As You Want Or Do As You Are Told? Control vs. Autonomy in Agile Software Development Teams, *Hawaii Int. Conf. Syst. Sci. 2018 HICSS-51*, Jan. 2018.
43. K. Kuusinen and S. Albertsen, Industry-academy Collaboration in Teaching DevOps and Continuous Delivery to Software Engineering Students: Towards Improved Industrial Relevance in Higher Education, in *Proceedings of the 41st International Conference on Software Engineering: Software Engineering Education and Training*, Piscataway, NJ, USA, 2019, pp. 23–27.
44. P. Melzer, A conceptual framework for task and tool personalisation in IS education, in *A Conceptual Framework for Personalised Learning*, Springer, 2019, pp. 47–76.

45. B. Schatz and I. Abdelshafi, Primavera gets agile: a successful transition to agile development, *IEEE Softw.*, **22**(3), pp. 36–42, May 2005.
46. T. Dingsøyr, T. E. Fægri, T. Dybå, B. Haugset and Y. Lindsjørn, Team Performance in Software Development: Research Results versus Agile Principles, *IEEE Softw.*, **33**(4), pp. 106–110, Jul. 2016.
47. V. Garousi, G. Giray, E. Tuzun, C. Catal and M. Felderer, Closing the Gap Between Software Engineering Education and Industrial Needs, *IEEE Softw.*, pp. 1–1, 2019.
48. J. Ramazani and G. Jergeas, Project managers and the journey from good to great: The benefits of investment in project management training and education, *Int. J. Proj. Manag.*, **33**(1), pp. 41–52, Jan. 2015.
49. J. Varajão, R. Colomo-Palacios and H. Silva, ISO 21500:2012 and PMBoK 5 processes in information systems project management, *Comput. Stand. Interfaces*, **50**, pp. 216–222, Feb. 2017.
50. P. Laplante, J. F. Defranco and E. Guimaraes, Evolution of a Graduate Software Engineering Capstone Course – A Course Review, *Int. J. Eng. Educ.*, **35**(4), pp. 999–1007, 2019.
51. M. Pozenel and T. Hovelja, A comparison of the planning poker and team estimation game: a case study in software development capstone project course, *Int. J. Eng. Educ.*, **35**(1), pp. 195–208, 2019.
52. C. S. Ridenour and Newman, *Mixed Methods Research: Exploring the Interactive Continuum*, SIU Press, 2008.
53. Y. S. Lincoln, S. A. Lynham and E. G. Guba, Paradigmatic controversies, contradictions, and emerging confluences, revisited, *Sage Handb. Qual. Res.*, **4**, pp. 97–128, 2011.
54. K. L. Wester, Publishing Ethical Research: A Step-by-Step Overview, *J. Couns. Dev.*, **89**(3), pp. 301–307, Jul. 2011.

Ricardo Colomo-Palacios, Full Professor at the Computer Science Department of the Østfold University College, Norway. Formerly he worked at Universidad Carlos III de Madrid, Spain. His research interests include applied research in information systems, software project management, people in software projects, business software, software and services process improvement and web science. He received his PhD in Computer Science from the Universidad Politécnica of Madrid (2005). He also holds a MBA from the Instituto de Empresa (2002). He worked as Software Engineer, Project Manager and Software Engineering Consultant in several companies including Spanish IT leader INDRA.

Cristina Casado-Lumbreras is working at Educational Psychology Department, Universidad Complutense de Madrid, Spain. She received her PhD in Psychology from the Universidad Autónoma of Madrid (2003). She has been working in educational and psychological consulting, and in several research institutions and Universities since 1990s. Her research interests include the study of emotions from different approaches and contexts, such as the study of emotions and values from a cross cultural perspective, or the analysis of emotions in the workplace. Likewise, her research production has also focused on the study of mentoring and coaching processes in educational and organizational contexts, or on the study of the influence of technology on education, learning and performance.

Terje Samuelsen, Assistant Professor at the Computer Science Department of the Østfold University College, Norway. His research interests include information systems, didactic in IT education, guidance as a tool for learning and people studies within projects. Apart from his teaching and research duties, he worked as Software Engineer for Kongsberg Gruppen, Norway.

Xabier Larrucea is currently project leader at Tecnalia and has mainly been involved in European projects (more than 25 research projects). Since January 2017 he is leading the SHIELD project related with cybersecurity in national health systems. His research interests cover all aspects of the software engineering discipline including cybersecurity. He is also part time lecturer at the University of Basque Country (EHU-UPV) on Software Engineering subjects. He is serving in the board of IEEE Software and IET Software. He is an IEEE Senior member and PMP certified.