Scrum in software engineering courses: an outline of the literature

Viljan Mahnič

University of Ljubljana
Ljubljana, Slovenia

ABSTRACT: This article provides an outline of the literature dealing with teaching Scrum in software engineering courses. The search of studies in the Scopus database revealed 23 papers that were identified as primary studies relevant to this research. These studies are classified by their main topic and their main results are presented. All studies stress that the teaching of Scrum should not be limited to traditional lectures, but some practical experience should be provided in order to strengthen comprehension and achieve deep learning. For this reason, the use of Scrum in capstone (or similar) projects requiring students to work in teams is the most widely adopted strategy, described in seven studies. Four studies describe the use of simulation games as an alternative to practical project work (and even traditional lectures). Teaching Scrum through a capstone course provides a suitable environment for in-depth analyses of students' perceptions of typical Scrum practices (discussed in four studies), but also opens some pedagogical issues, such as the assessment of individual performance and consideration of students' learning styles (three studies). The remaining studies describe general experience and recommendations, the development of teaching aids and the combination of Scrum with other process models.

Keywords: Scrum, software engineering education, agile methods, capstone project, problem-based learning

INTRODUCTION

Agile methods are increasingly being adopted by software organisations all over the world [1]. In order to fulfil industry needs, teaching these methods has become an important issue when defining a software engineering curriculum. The Software Engineering 2004 Curriculum Model [2] did not pay enough attention to this issue, and until a few years ago courses on agile methods were rather rare [3]. However, it is expected that the revision of this model, which is now in draft version pending approval [4], will embrace agile as a valid part of the curriculum from the start [5].

The most widespread agile method is Scrum [6][7]. According to the latest State of Agile Survey, Scrum and its variants (Scrum/XP Hybrid and Scrumban) are used by 72% of respondents [8]. Therefore, there is an increasing need to acquaint students with this topic and many educators are faced with the issue of how to incorporate Scrum into their software engineering courses. In order to help them in search of an appropriate solution, this article provides an outline of the literature describing hitherto experience in teaching Scrum. The review aims to present: 1) what has been reported about the use of Scrum in software engineering courses; and 2) what are the typical approaches most frequently used in teaching practice.

In order to answer these questions, a review protocol was developed following the guidelines suggested by Kitchenham and Charters that enabled the selection of most important primary studies [9]. These studies were then classified by their topics and further analysed with regard to their approach to teaching Scrum. The next two sections describe the selection of primary studies and their classification into five groups. Then, each group of studies is described in a separate section. Finally, the most important conclusions are summarised with emphasis on assignment of Scrum roles in student projects.

IDENTIFICATION AND SELECTION OF STUDIES

Relevant studies were searched in the Scopus electronic database (www.scopus.com) in June 2015 using the search string (TITLE-ABS-KEY(Scrum) AND TITLE-ABS-KEY(software engineering) AND TITLE-ABS-KEY(course)), which means that all publications containing the words *Scrum*, *software engineering* and *course* in their title, abstract or keyword list were expected to be retrieved. The Scopus database was chosen because it provides a single source of a wide range of high quality publications that should be otherwise searched in different digital libraries (e.g. ACM Digital Library, IEEE Explore, ISI Web of Science, etc).

The search resulted in 32 matches, out of which five were excluded from the review after reading the abstract (for being either conference reviews or not dealing with Scrum in software engineering education), while another seven were eliminated after reading the full text of the article (for not being focused on Scrum as their main topic). The remaining 20 studies (seven journal articles and 13 conference papers) were chosen for review and are listed at the end of this article as references [10-29]. During the review this list was augmented with references [30-32] that the author also found relevant, but did not appear in the search result in spite of being a part of the Scopus database.

CLASSIFICATION OF STUDIES

For the purpose of the review, the studies were classified by their main topic into five groups as shown in Table 2. The table clearly indicates that the most widespread approach to teaching Scrum is through practical work on student projects. This approach takes into account the statement of Scrum creators that Scrum is simple to understand yet difficult to master [33]. Therefore, in order to learn Scrum the students do not need extensive lectures, but should try it in practice. Courses of this type mainly exploit the benefits of capstone projects that simulate professional working environment as much as possible. Apart from seven studies in group 1 that concentrate on the content of such courses (i.e. studies by Damian et al [11], Kropp and Meier [15], Mahnič [17], Paasivaara et al [22], Reichlmayr [25], Scharf and Koch [27] and Zorzo et al [29]), student projects also served as a basis for most of studies dealing with other topics, e.g. students' perceptions of Scrum practices and teamwork (studies by Mahnič [18], Mahnič and Hovelja [19], Mahnič and Rožanc [20] and Poženel [24]) and pedagogical issues (studies by Gamble and Hale [13], Igaki et al [14] and Scott et al [28]).

Table 2: Classification of studies by topic.

Group	Topic	Studies dealing with this topic
1	Teaching Scrum through practical work on student projects	11, 15, 17, 22, 25, 27, 29
2	Teaching Scrum through educational games	21, 30, 31, 32
3	Students' perceptions of Scrum practices and teamwork	18, 19, 20, 24
4	Pedagogical issues	13, 14, 28
5	General experience and recommendations	12
	Teaching aids	23, 26
	Scrum and other process models	10, 16

Another widespread approach to teaching Scrum uses educational games as an alternative to practical project work. This approach is particularly suitable when the course does not allow enough time for the development of a complex project. The games usually require students to follow Scrum rules and practices when developing a simple product in several iterations. The review revealed four studies of this kind.

The third group consists of studies that analyse students' perceptions of Scrum practices and teamwork with the aim of improving Scrum teaching and finding those practices that most significantly affect the success of a Scrum project. The studies in the fourth group deal with pedagogical issues in a narrower sense, such as the assessment of students' performance and the relationship between project outcomes and students' learning styles. The remaining studies consider different topics, from outlining experience and recommendations on the one hand, to the use of Scrum in combination with other process models on the other.

TEACHING SCRUM THROUGH PRACTICAL WORK ON STUDENT PROJECTS

Studies from this group describe the design of courses that require students to work in teams in order to develop a non-trivial software project, mostly within the scope of a software engineering capstone course.

Studies by Mahnič [17] and Scharf and Koch [27] use similar design consisting of a preparatory Sprint (also called Sprint 0) and a sequence of regular Scrum Sprints. During Sprint 0, the students attend formal lectures on Scrum, get acquainted with the project they are going to develop and prepare the development environment. Regular Sprints are executed strictly following Scrum rules, each Sprint starting with the Sprint planning meeting and ending with the Sprint review and Sprint retrospective meetings. During the Sprint, the team members must meet regularly at Daily Scrum meetings to inform each other on their current activities and possible impediments. Each Sprint must provide an increment of the required functionality that must be demonstrated at the Sprint review meeting.

The designs differ in the Sprint length, the number of releases, and the assignment of Scrum roles. Mahnič assumes that the Product Owner role is played by a domain expert (either a member of the teaching staff or a representative of a company), while the ScrumMaster role is played by instructors [17]. On the other hand, Scharf and Koch advocate the assignment of the Product Owner and the ScrumMaster roles to students [27].

Kropp and Meier start from the premise that the competence required for agile software development can be divided into three major categories: engineering practices, management practices and agile values [15]. Engineering practices are best covered by Extreme Programming (XP) [34], management practices by Scrum and agile values by propagating

them as working values throughout the course. Consequently, the course is divided into two parts. Scrum is introduced in Part Two after students have mastered engineering practices advocated by XP. The students work in teams of six to eight in order to develop a 2D computer game in six one-week Sprints. One student is voted ScrumMaster, while the lecturer plays the role of the Product Owner. Every week each team does the Sprint planning, Sprint review and Sprint retrospective coached by the lecturer. After six Sprints a demonstration of games takes place.

Reichlmayr describes an undergraduate elective course in agile software development, which used Android mobile phones as the development environment for student teams to learn and practice Scrum [25]. A substantial part of the study is devoted to a description of Scrum, its rules and practices. However, it also provides an interesting description of how the students are engaged in writing user stories, estimating the effort using planning poker, identifying acceptance tests, and decomposing the stories into constituent tasks.

The study by Zorzo et al represents an attempt to combine the theoretical and practical parts of a software engineering course in order to execute both of them in parallel [29]. The theoretical part starts with an overview of Scrum, basic programming, and requirements and analysis, while more advanced topics are introduced later. The practical part of the course runs in parallel requiring students to work in teams of six in order to develop a mid-size project. The project work consists of a preparatory Sprint and eight regular Sprints. Scrum is used to plan the deliverables of each Sprint and to ensure that the deliverables are completed in due time. From this point of view, the authors claim that the use of Scrum justified their expectations since all teams were able to deliver the complete project in the end of the course. However, they also point out the problem of incompatibility between the gradual evolution of the learning process through the theoretical part of the course and the Scrum requirement that each Sprint must provide a complete increment of shippable product functionality. For this reason, many pieces of software were not completely implemented in initial Sprints, but had to be enhanced or adjusted afterwards, thus violating the Scrum concept of *done*.

Damian et al [11] and Paasivaara et al [22] use Scrum for teaching global software engineering skills within the scope of a capstone course. Both studies describe how three mixed globally distributed Scrum teams composed of students in Finland and Canada worked on a real software project in direct interaction with its Product Owner (a member of Finnish teaching staff). Damian et al describe how the Scrum method was implemented and adapted to work in a distributed environment, and present the infrastructure used to support collaboration [11]. Paasivaara et al further analyse the impact of Scrum on learning of global software engineering competences [29]. It is shown that Scrum adequately supports the learning of these competences, such as distributed communication and teamwork, building and maintaining trust, using appropriate collaboration tools and inter-cultural collaboration.

TEACHING SCRUM THROUGH EDUCATIONAL GAMES

In order to practice the application of Scrum before using it on a real project, Paasivaara et al propose a LEGO-based Scrum simulation game as an alternative to traditional lecture-based teaching [21]. The game was initially developed as an internal training tool in a Finnish software company and then used at Aalto University. The Product Owner and ScrumMaster roles are played by experienced professionals, while students work in teams in order to build a new product from LEGO blocks.

The simulation of Scrum starts with release planning, during which the teams (with the help of the Product Owner) create their Product Backlogs. Then, three to four iterations follow, each of them consisting of Sprint planning and two work periods with a Daily Scrum meeting between them. At the end of each iteration, a Demo and Retrospective take place. Each iteration lasts 26 to 30 minutes. The game aims to accomplish the learning goals concerning the Scrum process and roles, requirements management and customer collaboration, estimation, working in a team, and visualising work and progress.

Von Wangenheim et al present a low-cost paper and pencil game to reinforce and teach the application of Scrum in complement to theoretical lectures [30]. The game can be applied during a typical university class taking about 60 minutes. Students play the game in groups of six with the aim of planning and executing a hypothetical Sprint. Each group consists of a ScrumMaster, a Product Owner, an Auditor (optional), and three project team members. The Product Backlog consists of user stories representing fictive customer orders that require the production of different items (paper boats, hats and planes). The production should maximise overall gains in terms of profit and business value. The execution of the game consists of five steps: estimation of user stories, Sprint planning, Sprint execution, Sprint review and release. Sprint execution is further divided into substeps comprising a kick-off meeting and a sequence of three working periods, each of them ending with a Daily Scrum meeting. After the game, the students and the teacher conduct a debriefing session to reflect and share their learning of Scrum.

The game proposed by Fernandes and Sousa is a competitive game in which each student plays the role of a ScrumMaster [32]. The game uses several types of cards. The Product Backlog Cards determine the characteristics of the project, the Problem Cards represent the problems that may occur during the project, the Concept Cards serve as a remedy to problems and the Developer Cards correspond to the development team members of a player. At the beginning of the game, a Product Backlog card is chosen that defines the number of Sprints and tasks that should be completed, the cost of artefacts and the budget available to each player. Then, the players try to complete their tasks by

playing the Developer Cards, to sabotage the other players by playing the Problem Cards and to mitigate problems by playing the Concept Cards. The winner is the player who first completes all tasks defined for the project or has the highest percentage of fully completed tasks after the end of the last iteration.

Ramingwong and Ramingwong [31] describe a variation of the game developed by Krivitsky [35] that is often used in professional training courses. Instead of using LEGO bricks, which the authors claim to be too expensive to use in large classes, the study suggests the use of plasticine as the main development component.

STUDENTS' PERCEPTIONS OF SCRUM PRACTICES AND TEAMWORK

A capstone project provides a suitable environment not only for teaching Scrum, but also for studying students' perceptions of Scrum practices and teamwork. Since the capstone course takes place at the end of their studies, it can be assumed that the students behave similarly to young professionals. Therefore, the results of these studies can be generalised (although with caution) to industrial environment.

The study by Mahnič analyses students' opinions regarding particular Scrum concepts after teaching a Scrum-based capstone course for the first time with the aim of improving the course content and identifying those Scrum practices that significantly affect satisfaction with the work on a Scrum project [18]. It is shown that overall satisfaction depends on the clarity of requirements specified in the Product Backlog, clear rules for the maintenance of Sprint Backlog, appropriate administrative workload, and good co-operation with the ScrumMaster and the Product Owner. In the case studied, the students' satisfaction increased from Sprint to Sprint and to a considerable extent, their opinions matched the anecdotal evidence about Scrum benefits reported in the literature.

Mahnič and Hovelja provide an in-depth analysis of the use of user stories for requirements specification on the basis of several surveys that were conducted between 2010 and 2013 [19]. Although students often seem to be suspicious about the suitability of user stories, the study indicates that their opinions significantly improve after they gain more experience. Students successfully grasp the main concepts and understand the advantages and limitations of user stories. However, better students are more confident about potential benefits and keener to use user stories in practice. The authors attribute students' satisfaction to proper instruction of the course, which stimulates learning through problem solving and requires close co-operation between students, the Product Owner and the ScrumMaster.

Mahnič and Rožanc analyse students' opinions on how much the proper execution of particular Scrum practices contributes to the success of a Scrum project and compare their opinions to those of professional developers [20]. Both groups of respondents agree that teamwork, communication among team members and good communication with the Product Owner are most important. The students also stress the importance of strict adherence to the notion of *done*, clarity of the Product Backlog and proper execution of Sprint review meetings.

Poženel [24] analyses different aspects of teamwork within the capstone course described by Mahnič [17]. The analysis is based on an instrument recommended by Moe et al [36] and Stettina and Heijstek [37], and presents the results along five dimensions: shared leadership, team orientation, redundancy, learning and autonomy. Comparison with findings from industry reported by Stettina and Heijstek revealed no significant difference between students and professional developers [37].

PEDAGOGICAL ISSUES

Two studies in this group address the assessment of students' performance in Scrum projects, while the third one investigates the relationship between learning styles and learning outcomes when students use Scrum for the first time.

Gamble and Hale argue that evaluating individual student performance is often difficult because the team's productivity and outcomes can mask individual contributions [13]. Therefore, the study defines four performance metrics that characterise the depth and quality of individual performance (i.e. contribution, influence, impact and impression) and outlines a methodology to assign values to each metric. Collection of metrics is supported by SEREBRO 3.0 courseware, an open source collaborative environment that embeds a social network, project management modules and event capture system. It is shown that the proposed metrics provide broad characteristics of the level of engagement, activity and product related results of an individual on a team.

Igaki et al start from the premise that there are two challenges that make project assessment difficult: the concept of self-organisation of Scrum teams and the inequality of task assignment among students [14]. In order to solve this problem, a ticket-driven development method is proposed that visualises the students' process and enables a quantitative assessment of their projects. The study defines three assessment criteria, i.e., the quality of the project, the equity of task assignment, and the delivery management. For each assessment criterion a set of metrics is presented and the experience with their use in practice is described.

Scott et al [28] investigate the hypothesis that there is a relationship between the way the students perform Scrum practices and the students' learning style according to the Felder-Silverman model [38]. To address this issue, the authors mined association rules from the interaction of 33 Software Engineering students with Virtual Scrum [39],

a teaching aid that supports development of the software engineering capstone project at the UNICEN University, Tandil, Argentina. The analysis corroborated the existence of the aforementioned relationship; thus, encouraging further investigation on how to improve students' learning experience by adapting Scrum teaching to their learning profiles.

OTHER STUDIES

Devedžić and Milenković summarise their experience of teaching agile software development to different groups of students at different universities and different levels [12]. The study stresses the importance of practical project work and provides lessons learned and recommendations of how to overcome potential problems.

Potineni et al describe an interactive Web-based tutorial that leads the student through three phases: observation, data collection and development [23]. In the first phase, the student observes activities of a Scrum team during a week-long Sprint. The student can proceed from day to day only after answering a number of quiz-type questions correctly. In the second phase, the student is taken through a new Sprint, which requires that he/she acts as a data collector contributing towards data collection part of the implementation of the project. In the third phase, it is envisioned that the student will act as a developer on the team. When the study was published only the first two phases had been implemented.

Rodríguez et al [26] describe Virtual Scrum [39], a virtual reality environment that assists students with the running of a software project following the Scrum framework. Virtual Scrum supports artefacts needed for carrying out Scrum meetings and media-based tools to achieve permanent communication among team members, e.g. virtual Product Backlog, virtual Sprint Backlog, virtual planning poker, virtual Daily Scrum, etc.

The studies by Bruegge et al [10] and Krutchen [16] do not deal directly with teaching Scrum, but can be interesting for Scrum teachers since they treat Scrum in connection with other process models. Krutchen [16] describes the experience with teaching software project management on the basis of a conceptual model that accommodates a wide range of process models including Scrum, while Bruegge et al [10] introduce a new process model that combines the Unified Process [40] with Scrum elements.

CONCLUSIONS

This review of the literature has clearly shown that providing students with practical experience is of vital importance when teaching Scrum in software engineering courses. Scrum concepts are simple to explain, but can only be fully grasped after trying them in practice. Therefore, most courses require students to work in teams in order to develop a non-trivial software project. Using simulation games is another possible solution.

When designing practical project work or a simulation game, the instructors are often faced with a problem of how to define the Scrum roles. The existing literature does not provide a uniform answer to this issue. While it is clear that the students must play the role of the development team, the studies differ in their treatment of the Product Owner and ScrumMaster roles. As shown in Table 2, the prevailing approach is to assign the Product Owner role to an instructor or a domain expert from industry and the ScrumMaster role to one of the students. However, there are further differences concerning the student ScrumMasters. While most studies of this type assume that each student team has its own ScrumMaster, the studies by Damian et al [11] and Paasivaara et al [22] used a single ScrumMaster (an experienced student) for all development teams. Some studies (especially Scharf and Koch [27]) also claim that the students who play the Product Owner and ScrumMaster roles should change after each Sprint or release.

Table 2: Assignment of the Product Owner and ScrumMaster roles.

Role	Performed by instructors or domain experts from industry	Performed by students
Product Owner	11, 15, 17, 21, 22, 26, 29	25, 27, 30
ScrumMaster	17, 21, 25	11, 15, 22, 26, 27, 29, 30

The author's opinion is that having a domain expert as the Product Owner creates a more realistic simulation of collaboration with a real customer and increases the students' awareness of *...the complexity of many a piece of software, which is not so much caused by the intrinsic complexity of the problem, but rather by the vast number of details that must be dealt with,* as stated by Van Vliet [41]. Consequently, this approach makes it possible to impart the concept of *done* strictly and uniformly. Knowing all details behind each user story that must be implemented, an experienced Product Owner is competent to judge whether a story is fully implemented or not.

On the other hand, while it is beneficial for students to also practice the writing of user stories, using students as Product Owners introduces some risks. A student Product Owner usually has insufficient knowledge about the project domain, which may jeopardise the quality of user stories. Given the fact that a user story is just a rough description of the required functionality, it is difficult for instructors (as well as the other team members) to determine which details the student Product Owner actually had in his/her mind when he/she wrote the user story. Consequently, the imposing of the concept of *done* becomes difficult or even impossible. Therefore, such an approach is only appropriate if the project

requires development of a well-defined problem without idiosyncrasies that are usually inherent to real business applications and cause misunderstandings between the customer and developers.

The idea of assigning each team a student ScrumMaster seems useful since it increases responsibility of each team to follow Scrum rules and practices. Nevertheless, one of the instructors should still play the role of an *overall* ScrumMaster supervising the whole process.

Of the remaining studies, special attention deserve studies dealing with teaching Scrum in combination with global software development, assessing individual performance of students and combining Scrum with other process models.

Using Scrum in mixed globally distributed teams represents not only a promising way of teaching Scrum and global software engineering competences, but also contributes to overcoming of cultural differences and establishing better cooperation among universities from all over the world. Therefore, the teaching approach presented by Damian et al. [11] and Paasivaara et al [22] should be further elaborated and more widely used in practice.

Since Scrum stresses teamwork and collective responsibility for the final outcome of a project, individual contributions are often masked. It may happen that due to free riding or social loafing some members contribute little or nothing to the work of a team. Therefore, studies on metrics for assessing individual student performance are beneficial for the development of an appropriate grading scheme.

Finally, being aware that the pure agile and the pure disciplined approach to software development both have their own strengths and weaknesses, teaching how to combine Scrum with disciplined approaches in order to make the most of them seems to be an interesting topic, which also deserves more attention in the future.

REFERENCES

1. Williams, L., Agile software development methodologies and practices. *Advances in Computers*, **80**, 1-44 (2010).
2. IEEE Computer Society and ACM Joint Task Force on Computing Curricula, Software engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering (2004), 22 June 2015, http://sites.computer.org/ccse/SE2004Volume.pdf
3. Rico, D.F. and Sayani, H., Use of agile methods in software engineering education. *Proc. Agile 2009 Conf.*, Chicago, USA, 174-179 (2009).
4. IEEE Computer Society and ACM Joint Task Force on Computing Curricula, Software engineering 2015: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering (2015), 22 June 2015, https://www.acm.org/education/SE2014-20150223_draft.pdf
5. Fox, A. and Patterson, D., Is the new software engineering curriculum agile?. *IEEE Softw*are, 30, **5**, 85-88 (2013).
6. Schwaber, K. and Beedle, M., *Agile Software Development with Scrum*. Upper Saddle River, NJ: Prentice Hall (2002).
7. Schwaber, K., *Agile Project Management with Scrum*. Redmond, WA: Microsoft Press (2004).
8. VersionOne, 9th Annual State of Agile Development Survey (2015), 22 June 2015, http://info.versionone.com/state-of-agile-development-survey-ninth.html
9. Kitchenham, B. and Charters, S., Guidelines for Performing Systematic Literature Reviews in Software Engineering (2007), 22 June 2015, http://www.elsevier.com/__data/promis_misc/525444systematicreviewsguide.pdf
10. Bruegge, B., Krusche, S. and Wagner, M., Teaching tornado: from communication models to releases. *Proc. 8th Educators' Symp., EduSymp 2012*, Innsbruck, Austria, 5-12 (2012).
11. Damian, D., Lassenius, C., Paasivaara, M., Borici, A. and Schröter, A., Teaching a globally distributed project course using Scrum practices. *Proc. 2nd Inter. Workshop on Collaborative Teaching of Globally Distributed Software Develop. CTGDSD 2012*, Zurich, Switzerland, 30-34 (2012).
12. Devedžić, V. and Milenković, S.R., Teaching agile software development: a case study. *IEEE Trans. on Educ.*, 54, **2**, 273-278 (2011).
13. Gamble, R.F. and Hale, M.L., Assessing individual performance in agile undergraduate software engineering teams. *Proc. 2013 Frontiers in Educ. Conf.*, Oklahoma City, USA, 1678-1684 (2013).
14. Igaki, H., Fukuyasu, N., Saiki, S., Matsumoto, S. and Kusumoto, S., Quantitative assessment with using ticket driven development for teaching Scrum framework. *Companion Proc. 36th Inter. Conf. on Software Engng,*, Hyderabad, India, 372-381 (2014).
15. Kropp, M. and Meier, A., Teaching agile software development at university level: values, management, and craftsmanship. *Proc.26th Conf. on Software Engng. Educ. and Training*, San Francisco, USA, 179-188 (2013).
16. Kruchten, P., Experience teaching software project management in both industrial and academic settings. *Proc. 24th Conf. on Software Engng. Educ. and Training*, Waikiki, Honolulu, USA, 199-208, (2011).
17. Mahnič, V., A capstone course on agile software development using Scrum. *IEEE Trans. on Educ.*, 55, **1**, 99-106 (2012).
18. Mahnič, V., Teaching Scrum through team-project work: students' perceptions and teacher's observations. *Inter. J. of Engng. Educ.*, 26, **1**, 96-110 (2010).
19. Mahnič, V. and Hovelja, T., Teaching user stories within the scope of a software engineering capstone course: analysis of students' opinions. *Inter. J. of Engng. Educ.*, 30, **4**, 901-915 (2014).

20. Mahnič, V. and Rožanc, I., Students' perceptions of Scrum practices. *MIPRO 2012 - Proc. 35th Inter. Conv. on Infor. and Communication Technol., Electronics and Microelectronics*, Opatija, Croatia, 1178-1183 (2012).
21. Paasivaara, M., Heikkilä, V., Lassenius, C., and Toivola, T., Teaching students Scrum using LEGO blocks. *Companion Proc. 36th Inter. Conf. on Software Engng.*, Hyderabad, India, 382-391 (2014).
22. Paasivaara, M., Lassenius, C., Damian, D., Raty, P. and Schroter, A., Teaching students global software engineering skills using distributed Scrum. *Proc. 36th Inter. Conf. on Software Engng.*, San Francisco, USA, 1128-1137 (2013).
23. Potineni, S., Bansal, S.K. and Amresh, A., ScrumTutor: a web-based interactive tutorial for Scrum software development. *Proc. 2013 Inter. Conf. on Advances in Computing, Communications and Informatics ICACCI 2013*, Mysore, India, 1884-1890 (2013).
24. Poženel, M., Assessing teamwork in a software engineering capstone course. *World Trans. on Engng. and Technol. Educ.*, 11, **1**, 6-12 (2013).
25. Reichlmayr, T., Working towards the student Scrum - developing agile Android applications. *Proc. 118th ASEE Annual Conf. and Exposition*, Vancouver, Canada, 12 (2011).
26. Rodríguez, G., Soria, A. and Campo, M., Teaching Scrum to software engineering students with virtual reality support. *Lecture Notes in Computer Science 7547*, 140-159 (2012).
27. Scharf, A. and Koch, A., Scrum in a software engineering course: an in-depth praxis report. *Proc. 26th Conf. on Software Engng. Educ. and Training*, San Francisco, USA, 159-168 (2013).
28. Scott, E., Rodríguez, G., Soria, Á. and Campo, M., Are learning styles useful indicators to discover how students use Scrum for the first time?. *Computers in Human Behavior*, 36, 56-64 (2014).
29. Zorzo, S.D., De Ponte, L. and Lucrédio, D., Using scrum to teach software engineering: a case study. *Proc. 2013 Frontiers in Educ. Conf.*, Oklahoma City, USA, 455-461 (2013).
30. Von Wangenheim, C.G., Savi, R. and Borgatto, A.F., SCRUMIA - an educational game for teaching SCRUM in computing courses. *J. of Systems and Software*, 86, **10**, 2675-2687 (2013).
31. Ramingwong, S. and Ramingwong, L., Plasticine Scrum: an alternative solution for simulating Scrum software development. *Lecture Notes in Electrical Engng. 339*, 851-858 (2015).
32. Fernandes, J.M. and Sousa S.M., PlayScrum - a card game to learn the Scrum agile method. *Proc. Second Inter. Conf. on Games and Virtual Worlds for Serious Applications (VS-GAMES)*, Braga, Portugal, 52-59 (2010).
33. Schwaber, K. and Sutherland, J., The Scrum Guide (2013), 22 June 2015, http://www.scrumguides.org/
34. Beck, K., *Extreme Programming Explained: Embrace Change*. (2nd Edn), with C. Andres. Boston, MA: Addison-Wesley (2005).
35. Krivitsky, A., Scrum Simulation with LEGO Bricks (2009), 26 June 2015, https://www.scrumalliance.org/system/resource_files/0000/3689/Scrum-Simulation-with-LEGO-Bricks-v2.0.pdf
36. Moe, N.B., Dingsøyr, T. and Røyrvik, E.A., Putting agile teamwork to the test - a preliminary instrument for empirically assessing and improving agile software development. *Lecture Notes in Business Infor. Processing 31*, 114-123 (2009).
37. Stettina, C.J. and Heijstek, W., Five agile factors: helping self-management to self-reflect. *Proc. 18th European System and Software Process Improvement and Innovation Conf. (EUROSPI 2011) (Communications in Computer and Infor. Science, 172, Springer)*, Roskilde, Denmark, 84-96 (2011).
38. Felder, R. M. and Silverman, L. K., Learning and teaching styles in engineering education. *Engng. Educ.*, 78, **7**, 674-681 (1988).
39. Rodriguez, G., Soria, A. and Campo, M., Virtual Scrum: a teaching aid to introduce undergraduate software engineering students to Scrum. *Computer Applications in Engng. Educ.*, 23, **1**, 147-156 (2015).
40. Krutchen, P., *The Rational Unified Process: an Introduction*. Boston, MA: Addison-Wesley (2004).
41. Van Vliet, H., *Software Engineering.*(3rd Edn), Hoboken, NJ: John Wiley & Sons, Ltd (2008).

BIOGRAPHY

Viljan Mahnič is an Associate Professor and Head of the Software Engineering Laboratory in the Faculty of Computer and Information Science at the University of Ljubljana, Slovenia. His teaching and research interests include agile software development methods, software process improvement, empirical software engineering and software measurement. He received his PhD in Computer Science from the University of Ljubljana in 1990.