Experiences from an Experiential Learning Course on Games Development

Stephan Krusche
TU München
Munich, Germany
krusche@in.tum.de

Barbara Reichart
TU München
Munich, Germany
reichart.b@in.tum.de

Paul Tolstoi
TU München
Munich, Germany
tolstoi@in.tum.de

Bernd Bruegge
TU München
Munich, Germany
bruegge@in.tum.de

ABSTRACT

Games development brings fun into the software engineering curriculum, but it is a practical activity that educators cannot teach in traditional lecture-based environments. Including pedagogical aspects of problem-based, cooperative, blended and experiential learning is necessary to achieve a high learning experience.

In this paper we describe and evaluate a block course in which we ask students with almost no mobile application development experience to create games in just two weeks. We offer the course in two different modes, for beginners who learn games development, and for tutors who help us in the organization of the course and in the teaching activities.

Apart from games development and game design, students learn modeling, design patterns and software configuration management. They practice soft skills in team work, present their games to their classmates using Pecha Kucha and publish their games into the App Store. Our evaluations show that students appreciate the great learning experience.

Keywords

Software Engineering, Problem-based Learning, Blended Learning, Cooperative Learning, Interactive Tutorials

1. INTRODUCTION

Software engineering is an interactive and collaborative activity [23] that requires the practical application of knowledge [8, 20]. Educators struggle when teaching it in traditional lecture-based environments where activities take place in the front of the classroom. Lectures are usually similar to broadcasting, where essential education interactions are initiated by the teacher with only limited students participation. Self-guided learning, personal responsibility, practical relevance and individualization are important elements of a great learning experience. Several pedagogic theories have been developed that include these elements.

Problem-based learning is a technique to learn about a subject through the experience of problem solving. Educators facilitate learning by supporting, guiding, and monitoring this process [4]. *Cooperative learning* is an educational approach which aims to organize classroom activities into social learning experiences: Students work in groups to complete tasks collectively towards a common goal [12].

Blended learning allows students to learn through delivery of content and instructions via computer-mediated activities, digital and online media [10]. *Experiential learning* is the process of learning from experience, a methodology in which educators engage with students in direct experience to increase knowledge, develop skills, and clarify values [14].

We developed a course that includes a mix of these approaches and teaches software engineering concepts trough games development. While the combination leads to a more complex experience for educators, it lowers their stress and leads to higher satisfaction [2]. We base our teaching approach on a Chinese proverb, first mentioned by Confucius and adapted by Benjamin Franklin. In recent publications, an extended version of the proverb is mentioned: *"Tell me and I will forget. Show me and I will remember. Involve me and I will understand. Step back and I will act"* [15].

Explaining a concept only theoretically does not allow students to apply it. Therefore we include the idea of cognitive apprenticeship: an apprentice observes the skills of a master who shows how a concept works in practice. Clarifying the thinking process behind the application makes it easier for the apprentice to imitate the behavior [7]. Involving students in the learning process using interactive tutorials follows the idea of experiential learning. Students apply a concept on their own, in a way that fits to their own techniques, to understand the concept and its application. Including self-guided learning, self-improvement and problem-based learning lets students take the responsibility to solve a problem on their own using the concepts they learned before.

We teach students the ability to develop mobile application in a two-weeks block course. Compared to traditional games development, the development of mobile games is less complex [18]. This enables students to create playable mobile games within two weeks. Games development is a popular topic among computer science students and brings fun into software engineering [6, 22]. It has personal meaning-

To appear in the 47th ACM Technical Symposium on Computing Science Education - SIGCSE, March 2016, Memphis, TN, USA.

fulness, hence it is easier for students to come up with their own ideas. Many students are interested in developing their own casual game and publish it into the App Store.

In this paper we show experiences from an experiential learning course on games development with up to 40 students. Section 2 describes the design of the course including learning objectives, organizational aspects, the schedule and the structure of an interactive tutorial. Section 3 shows experiences and observations during the conduction of several courses in the last years. We explain how to handle difficult situations and how tutors improve the learning experience. Section 4 shows findings of evaluations of the course, while Section 5 describes related work.

2. COURSE DESIGN

Creating and conducting a software engineering course on games development involves several aspects. The curriculum of the course needs to include topics like programming, the use of frameworks, game design, the application of design and architectural patterns, and user interface design. In this section we describe the course design starting with its learning objectives. We explain the organization including preparation, course and follow-up phase. We then describe details of the course phase, where students first attend interactive tutorials and then develop a mobile game in teams of two students. We finally explain structure and purpose of an interactive tutorial.

2.1 Learning Objectives

The learning objectives of the course support the seven outcomes described in the SE2004 recommendations for undergratuate software engineering curricula [19]: (1) show mastery of knowledge and skills necessary to begin practice, (2) work individually or in a team to develop quality software, (3) make appropriate trade-offs within cost and time limitations, (4) perform design in one or more domains, (5) demonstrate understanding and apply current theories, models, and techniques, (6) demonstrate soft skills such as negotiation, leadership, communication, (7) learn new models, techniques, and technologies as they emerge.

Depending on previous experience and interest, students participate in the course in two different modes: beginner and advanced. The beginner mode targets students who have not developed mobile applications or games before and have no experience with the programming language and the tools taught during the course. However, it is assumed that the beginners have basic knowledge of an object-oriented programming language and know how to model using UML. During the first week, the major focus is on learning the object-oriented programming language, the use of frameworks and the integrated development environment (IDE).

In addition to programming, the course includes concepts about design patterns and modeling to help the students structuring their source code and facilitating extensibility and reuse. Distributed version control allows the students to work collaboratively on source code. Another focus of the course are soft skills. The students practice communication with other developers, tutors, and teaching assistants (TA). Every team prepares and presents a presentation which is recorded. The students receive the videos and obtain feedback on content and delivery of their presentation.

The course does not focus on project management techniques like Scrum or planning poker, because the development teams consist of two students sitting next to each other. Including those topics would unnecessarily increase the complexity of the course. Such project management techniques only provide benefits in projects with an extended time period and a larger team as described in [17]. Instead, students work in pairs on a small prototype (using pair programming) for one week which encourages the cooperative learning process when working towards a common goal. This gives them enough confidence in their programming abilities, required to build larger applications, while at the same time not limiting them to mere implementation of small programming assignments. After the course, the students deepen their experience in capstone courses where they develop larger applications within a realistic context as described in [5]. Then the students also learn more about project management techniques.

In the advanced mode, students participate as tutors who already have acquired knowledge in the development environment, including programming language and tool chain. Tutors study one specialized and advanced topic such as sprite animation and prepare an interactive tutorial under the supervision of the TAs. They also learn about how to structure and teach the topic and how to present it as interactive tutorial. Tutors help beginners during development in the second week and deepen their knowledge about application and games development. All students learn about game design and the use of platform specific frameworks and have to deal with distributed version control to obtain the course material, to submit solutions to exercises and to work in teams during the development of the game. They practice their social and non technical skills such as working in a team, presenting, communicating and being proactive.

2.2 Organization

The course is organized by two TAs, one is responsible for the introduction to the programming language, the other one is familiar with games development. The TAs ask four to six students with experience in the programming language and tools to participate as tutors in order to help in the organization of the course. The course activities are shown in Figure 1. The TAs prepare the course starting about eight weeks in advance (colored orange in Figure 1). In the two weeks long course phase (green), all students attend fulltime, typically before the semester starts. Finally there is a follow-up phase (blue) that is about six weeks long.

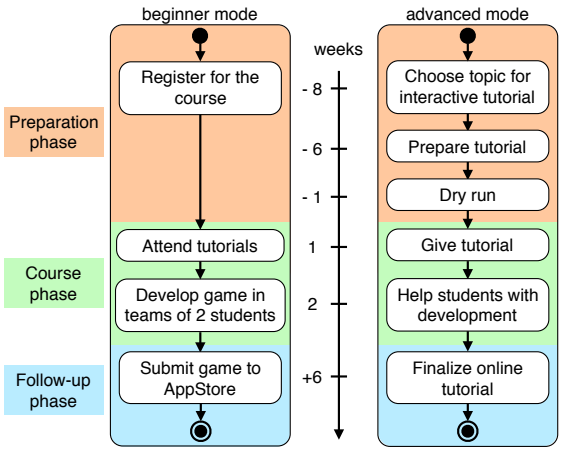

Figure 1: Course activities (beginner and advanced)

There are several reasons for conducting the course phase in two weeks. It is easier for TAs, tutors and students to keep focus and concentration high, when working full-time in the course. Developing a game in a short time is a huge motivating factor. TAs start to prepare the course at the end of the previous semester, when students apply for it. Undergraduate students (bachelor) have to be at least on the sophomore level to participate. Graduate students (master) also participate. The TAs can assume that participating students have learned enough about object-oriented programming and modeling. During registration the TAs tell the students that they can participate in the course in two different modes. In a short separate meeting, the TAs present the tutors possible advanced topics and distribute those, taking into account personal preferences.

Before the course starts, TAs supervise the tutors in creating one interactive tutorial using a master-apprenticeship approach. The tutors create an outline for the presentation and an idea for a sample project. Over several iterations they improve based on feedback and positive examples provided by the TAs. This process ensures that the tutor learns about the most important aspects in teaching. Furthermore it serves as quality control to ensure the content is correct and presented in an approachable manner.

One week before the course phase with all students, the tutors dry run their tutorial. They give the entire tutorial session as close to the final presentation as possible. TAs and other tutors participate in each dry run session, playing the role of beginners, so that the presenting tutor can observe them, discovering issues that could arise in the actual session. The tutor then receives detailed feedback on content and delivery. Details about the schedule of the course phase are described in the next section. In the follow-up phase, tutors write an online tutorial about their topic. Meanwhile, TAs help the beginners to submit their games into the App Store. While the submission is optional, usually all students are eager and finalize their game with respect to stability and feature completeness so that TAs can upload the game.

2.3 Schedule

The schedule of the course phase is shown in Figure 2. The goal of the first week is to teach basic as well as advanced concepts of the programming language and platform using interactive tutorials, which are described in more detail in the next section. TAs give 10-12, tutors give 4-6 tutorials. While TAs impart essential concepts required for games development in the first three days, tutors teach more specialized topics at the end of the week. The first day of the block course introduces students to distributed version control, so that they can obtain the course material, submit their solutions and collaborate in the second week. Then TAs start with basics about the programming language and the development environment, continue with intermediate language concepts like inheritance and closures and advance to graphic engine APIs. At the end of the first week, the programming assignment starts: the TAs ask the students to apply their newly gained knowledge by developing a game in one week until the end of the second week.

During the second week, the beginners develop a game in teams of two students. They choose their own game idea and receive feedback from TAs and tutors. The TAs ask the students to start with a simple idea they could actually develop in two days and that is extensible later on with features like leaderboards, achievements, particle effects or additional levels. A graphics designer from industry helps the students creating 2D and 3D graphics for their games. TAs also setup continuous integration and continuous delivery as explained in [16] and show the wallboard of the continuous integration server as shown in Figure 3. This ensures that the students always have a version in the repository that can be installed and tested on a real mobile device.

	Monday	Tuesday	Wednesday	Thursday	Friday
First week	Introduction to language & platform	Advanced language & platform concepts	Introduction to game development	Advanced game topics	Advanced game topics
					Programming assignment
Second week	Development of a game in teams of 2 students				
					Presentation and demo

Figure 2: Detailed schedule of the course phase

Figure 3: Wallboard with the status of the games

There is an intermediate milestone after two days, where the students present their ideas and current progress, and obtain feedback by the TAs. At the end of the second week, they present their game in a short Pecha Kucha [3] presentation. Presentations following the Pecha Kucha style have up to 20 slides each shown for exactly 20 seconds. The transition to the next slides is automatic and cannot be controlled manually. This presentation style requires the students to insert only simple graphics and only few text statements into one slide. It prevents the problem, that they talk too much on a boring slide, and have to rush on an interesting slide later in the talk. Not only the presenters, also the audience is more relaxed. It is easier for the TAs to control that students finish their presentation in time.

2.4 Interactive Tutorial

Each day of the first week includes three tutorial slots. An interactive tutorial is based on the idea of experiential and blended learning and consists of two parts. The first part is about 90 minutes long and includes theory intermixed with small exercises, each about 5-10 minutes long. The students learn concepts, then directly apply them and see step by step solutions afterwards. At the beginning of a tutorial, they download the slides to lookup concepts and solutions to these small exercises. After discussing the solution, the presenter also reflects on the concept and its application and tells the students when to use it. TAs and tutors directly help the students if they face problems during the tutorial.

The second part of the tutorial is an autonomous and problem-based learning exercise that is about 30 minutes long. Students have to solve this exercise on their own, challenging their understanding and participation in the first

part by repeating concepts of the tutorial. TAs and tutors only explain the task, but do not show the solution. If students have serious problems with the autonomous exercise, the tutors repeat the concepts of the tutorial with them.

3. CASE STUDY

We describe observations and experiences from several courses in the last years in this section. We explain typical problems and how we addressed them. We show insights about the relationship between TAs, tutors and students and how an open atmosphere improves the learning experience.

The courses uses Apple's iOS platform to teach object-oriented programming with Objective-C - recently Swift - and games development. Playgrounds, lightweight live programming environments, allow students to explore and evaluate code interactively without the need to compile and without the full complexity of the IDE. They show immediate feedback and are ideal to introduce new programming or language concept to beginners. Figure 4 shows an example of a playground. In the left editor area, students type in code that is constantly evaluated so they get immediate feedback. In the middle, they see these evaluation results and can use the quick look feature to display more information. The timeline view on the right inspects the code and visualizes results of loops or animations.

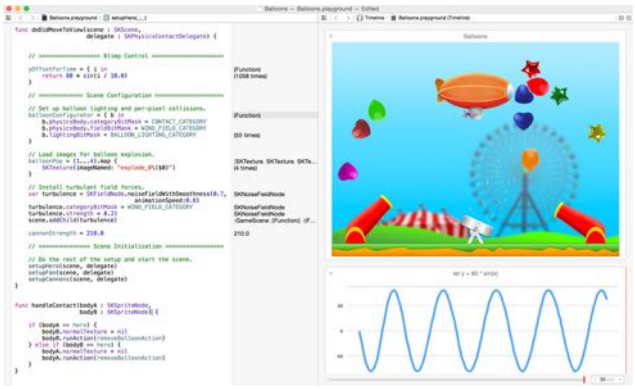

Figure 4: Xcode Playground: an interactive environment to experiment with Swift code [1]

Participating students have varying prior knowledge and experiences, forming a heterogeneous group. There are students with some experiences in mobile application development and students who never used the development environment before. To create a productive learning environment for all students, our interactive tutorials include easy exercises and optional advanced ones where even experienced students feel challenged. For such challenges it is necessary to combine knowledge from different sessions or to use the platform documentation to find the right solution. TAs introduce the delegation pattern and closures as an alternative, later the students need to be able to apply both techniques when they work with more advanced frameworks.

While experienced students solve more advanced exercises, TAs and tutors explain concepts again to less experienced students who need more time. TAs also ask students who finish exercises early to help their classmates to learn different ways of thinking and to consolidate the knowledge they just learned. If they are able to explain a concept to fellow students, they really understood it. Less experienced students need more time when interacting with the IDE, because they have to learn keyboard shortcuts and are initially unable to cope with the development tools. Due to the tight schedule some students cannot keep up with the pace of the course. In such cases, tutors help and motivate, especially if they experienced the same problems before.

Tutors have multiple responsibilities: They observe students and actively help them if they cannot follow the instructions. While helping students and experiencing their problems during the first sessions, the tutors get insights and can update their own tutorials to increase their usefulness. This includes adding small repetitions for important concepts of the course as well as adding hints to deepen the understanding. In the first week, the tutors also correct the submitted solutions of the students for the large exercises at the end of each tutorial and provide feedback. The course uses a distributed version control system for the submission of solutions which simplifies the correction of the tutors.

TAs ask students during an interactive tutorial to explain the rationale behind a solution because some students only copy and paste source code from the sample solution and do not try to understand the concept behind it. If tutors observe that students did not understand what they accomplished, they ask them to explain the solution. If students cannot explain the solution, because they did not understand it, the tutors explain the concept again to make sure the students can use it in later tutorials.

To improve the self-learning process of the students the tutors provide hints or references to documentation and slides. Another possibility is to refer to a prior solution and ask the students if they can adapt it to solve the current problem. Directly explaining the solution would reduce the sense of achievement for the students, but it is still better to help them instead of leaving them behind, especially because later sessions depend on the knowledge of previous ones. If students have difficulties, it helps to ask them to explain the problem in detail. During their own explanation of the problem, they suddenly understand it and a solution comes up to solve it. Hunt describes this process as Rubber Duck Debugging, an informal technique, where a developer explains his code, line-by-line, to a rubber duck [11].

Tutors assist the students in the game idea finding phase by asking questions about the main goal and core mechanics. We observed, that identifying core features of their game helps students to create a plan for the development. Asking questions encourages the students to think about their game idea and its potential flaws. This allows them to discard the game idea or to improve certain features, before they find out that the idea is not realizable during the actual development.

TAs ask students about feedback to improve the course. Tutors provide instant feedback during tutorials. Overall an open feedback culture is established where everyone should constructively criticize others and where feedback is not taken personal. While honest and detailed feedback is difficult for some of the students, they get used to it during the course and appreciate the idea of continuous improvement.

4. EVALUATION

We evaluated quality and usefulness of the tutorials and the course organization using surveys after three course instances with 24 beginners on average. This section describes the findings and limitations of these two evaluations.

4.1 Tutorial Evaluation

The first evaluation concerns the quality of interactive tutorials. We asked students in online surveys about the usefulness of each tutorial and about each speakers performance, for both, TAs and tutors. These surveys were completely anonymous and on average 75% of the students participated. The results are shown in Table 1.

Statement	Strong agree	Agree	Neutral	Disagree	Strong disagree
Objectives were clear	63%	33%	4%	0%	0%
Assignments supported objectives	58%	38%	4%	0%	0%
I learned new concepts	67%	25%	8%	0%	0%

Table 1: Results of tutorial evaluation

We asked the beginners about the amount of content, speed of delivery and difficulty of exercises of each tutorial. 79% of the beginners found the amount was right, only around 13% found it was too much content and 8% found it was too less. 84% found that the speed of the tutorial was right, while 8% found it too fast and 8% too slow. On average, 88% reported that the difficulty of the tutorial was right, only 8% found it was too hard and 4% found it was too simple. All tutorials had similar results in the survey. Therefore, we conclude that the quality of the tutorials was very high. We also asked open questions, e.g. about improvement recommendations. One student suggested to reduce the number of concepts in one tutorial and to have more repeating exercises instead.

4.2 Course Evaluation

The second evaluation concerns the general learning experience. We handed out the standardized paper based course evaluation of the university after the course and asked participants to fill it out anonymously. 76% answered 32 questions with a five point Likert answer scale. The results of different courses do not deviate significantly, so we combined them and present the most interesting findings.

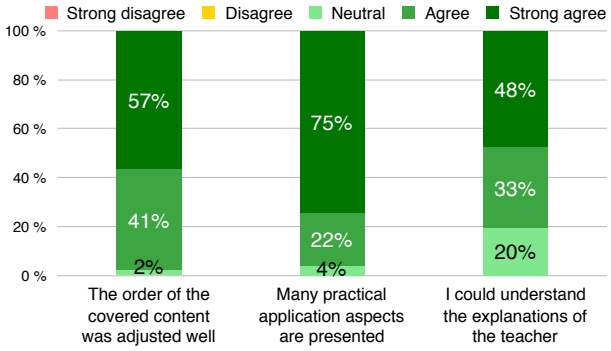

Figure 5: Evaluation of course difficulty

Figure 5 shows that almost all students agreed that the order of the covered content was adjusted well and that many practical application aspects are presented. About 80% of the students agreed that they could understand the explanations of the teacher, while only 20% had a neutral opinion. Regarding the heterogeneous group of participants we think these are great results. Figure 6 shows that students considered the general difficulty of the course as high. We think it is important to challenge students instead of demanding too little from them. Speed of the content delivery was also considered too high based on the fact that the course is only two weeks long and that beginners with almost no development experience learn to develop a game. Students reported that the required prior knowledge was not too high to successfully participate in the course. We experienced no case yet where a student dropped out during the course which confirms that all students are motivated to pass the course.

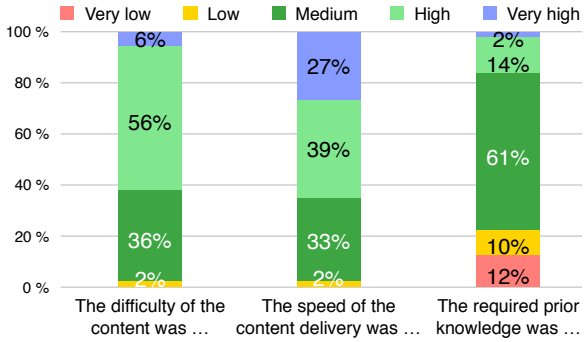

Figure 6: Evaluation of course content

68% evaluate the overall course as very good, 26% evaluate it as good and 6% evaluate it as neutral. No one evaluated it as bad or very bad. When we talked to students during and after the course we recognized that all of them really appreciated what they learned and that they can profit from the course in their future career. Most of the students participate in follow-up courses and are able to contribute greatly with the experiences they gained in this course.

4.3 Limitations

With this evaluation we aimed to explore whether students learned the concepts we taught, whether they found tutorials useful and whether speed and difficulty of the course was appropriate. We see threats to the validity of our evaluations and want to discuss them briefly. We might have the problem of selection bias because not all participants of the course took part in the questionnaires and the prior knowledge and experiences of the students were different.

We observed that experienced students found speed and difficulty of the course appropriate or sometimes too low, while inexperienced students usually found the tutorials too hard and too fast. We cannot guarantee that the distribution of experienced and inexperienced students were representative because not all students participated in the anonymous evaluation. However, in the discussions during and after the course the students agreed with our findings.

We were not able to measure the benefits of the learning techniques that we used. In general, the students agree that the mix of techniques such as experiential and blended learning is helpful, especially in contrast to traditional lectures. However, this is not an objective evaluation whether the particular technique is useful or not.

5. RELATED WORK

In this section, we describe related work. Eow et al. describe an approach where games development and appreciа-

tive learning enhance the creative perception of learners [9]. They studied 13-14 year old form one students and focused on creativity, while the course that we describe in this paper focuses on software engineering abilities. Jones describes a capstone course for undergraduates where students design and implement computer games [13]. He teaches a variety of computer science concepts but does not describe the application and benefits of pedagogic theories in his course.

Kurkovsky engages students through mobile game development early in the curriculum [18]. It enables students with limited programming experience to develop playable games within one course which is the same observation we made. While he describes different computer science areas that he covers in his course, he does not detail the educational approaches. Sweedyk and Keller use computer games as projects in their introductory software engineering course [22]. They focus on a detailed description of the developed games and describe how they meet the SE2004 learning outcomes [19], but leave out interesting details about which pedagogic theories they applied.

Smith et al. describe experiences with agile games development in a senior design course [21]. They found that emphasis on agile methods and fast releases is effective in aiding students to produce working games, however they do not explain pedagogic theories used in their course. Claypool and Claypool present how developing a game engages students to learn software engineering through games design [6]. They describe a linear teaching approach with 10 software engineering modules targeted to games design, but do not detail which pedagogic techniques they apply to teach these modules.

6. CONCLUSION

In this paper we described a practical two-weeks block course where we use different teaching aspects from problem-based, cooperative, blended and experiential learning. We teach students with almost no development experience to develop mobile games in two weeks. This brings fun into the software engineering curriculum and increases their skills in object-oriented programming. Each semester, more than 100 students apply for the course and want to learn software engineering with the help of games development. We continuously improve the learning experience of the students and get help by tutors in the organization of the course.

We further accompany the students after the final presentations to publish their games. Until now, they submitted 18 games into the iOS App Store. Our evaluations show that students appreciate a great learning experience with practical aspects that they can further use in their career. They improve their software engineering abilities, in particular object-oriented programming, and their soft skills with the help of games development and they have fun.

7. REFERENCES

[1] Apple. Introducing Swift, 2014. https://developer.apple.com/swift.

[2] R. Ben-Ari, R. Krole, and D. Har-Even. Differential effects of simple frontal versus complex teaching strategy on teachers' stress, burnout, and satisfaction. *International Journal of Stress Management*, 2003.

[3] A. Beyer. Improving student presentations pecha kucha and just plain powerpoint. *Teaching of Psychology*, 2011.

[4] D. Boud and G. Feletti. *The challenge of problem-based learning*. Psychology Press, 1998.

[5] B. Bruegge, S. Krusche, and L. Alperowitz. Software engineering project courses with industrial clients. *ACM Transactions on Computing Education*, 2015.

[6] K. Claypool and M. Claypool. Teaching software engineering through game design. In *SIGCSE Bulletin*, volume 37, pages 123–127. ACM, 2005.

[7] A. Collins, J. S. Brown, and A. Holum. Cognitive apprenticeship: Making thinking visible. *American educator*, 1991.

[8] T. M. Connolly, M. Stansfield, and T. Hainey. An application of games-based learning within software engineering. *British Journal of Educational Technology*, 38(3):416–428, 2007.

[9] Y. Eow, W. Ali, R. Mahmud, and R. Baki. Computer games development and appreciative learning approach in enhancing students creative perception. *Computers & Education*, 54(1):146–161, 2010.

[10] R. Garrison and H. Kanuka. Blended learning: Uncovering its transformative potential in higher education. *The internet and higher education*, 2004.

[11] A. Hunt and D. Thomas. *The pragmatic programmer: from journeyman to master*. Addison-Wesley, 2000.

[12] D. Johnson et al. *Cooperative Learning: Increasing College Faculty Instructional Productivity. ASHE-ERIC Higher Education Report*. ERIC, 1991.

[13] R. Jones. Design and implementation of computer games: A capstone course for undergraduate computer science education. In *Proceedings of the 31st SIGCSE Technical Symposium*, pages 260–264. ACM, 2000.

[14] D. Kolb et al. *Experiential learning: Experience as the source of learning and development*. Prentice-Hall, 1984.

[15] F. Korthagen et al. *Linking practice and theory: The pedagogy of realistic teacher education*. Routledge, 2001.

[16] S. Krusche and L. Alperowitz. Introduction of Continuous Delivery in Multi-Customer Project Courses. In *Proceedings of ICSE*. IEEE, 2014.

[17] S. Krusche et al. Rugby: An agile process model based on continuous delivery. In *Proceedings of the 1st Workshop on RCoSE*, pages 42–50. ACM, 2014.

[18] S. Kurkovsky. Engaging students through mobile game development. In *SIGCSE Bulletin*, pages 44–48. ACM, 2009.

[19] T. Lethbridge et al. Se2004: Recommendations for undergraduate software engineering curricula. *IEEE Software*, 23(6):19–25, 2006.

[20] D. Shaffer. Pedagogical praxis: The professions as models for postindustrial education. *Teachers College Record*, 106(7):1401–1421, 2004.

[21] T. Smith et al. Software engineering senior design course: experiences with agile game development in a capstone project. In *Proceedings of the 1st GAS Workshop*, pages 9–12. ACM, 2011.

[22] E. Sweedyk and R. M. Keller. Fun and games: a new software engineering course. In *SIGCSE Bulletin*, pages 138–142. ACM, 2005.

[23] J. Whitehead. Collaboration in software engineering: A roadmap. *FOSE*, 7(2007):214–225, 2007.