Experience in Learning Test-driven Development: Space Invaders Project-driven

Isabelle Blasquez, Hervé Leblanc

Experience in Learning Test-Driven Development: Space Invaders Project-Driven

Isabelle Blasquez
Limoges University
Limoges, France
isabelle.blasquez@unilim.fr

Hervé Leblanc
IRIT
Toulouse, France
leblanc@irit.fr

ABSTRACT

From Kent Beck's point of view, Test-Driven Development (TDD) really encourages simple design and inspires confidence. This agile software engineering practice suggests a different programming way that requires writing tests before writing the code in short cycles to reduce feedbacks loops. To help novice programmers discover and appreciate it, this paper describes an experience based on a laboratory course for learning TDD by developping a Space Invaders game through a step-by-step project-driven approach. To improve student engagement, a game development is chosen. All concepts are taught within the context of this project and students learn by doing. Applied in the context of a basis of object-oriented design course, this experience should raise student awareness of design's and programming's good practices as testing, refactoring, simple design and short releases. The evaluation shows that this experience helps students to engage in the learning process, to reflect on the importance of testing in a software development, to make aware of code quality and to understand the benefits of TDD.

KEYWORDS

Test-Driven Development, Agile Game Development, Sofware Engineering Learning

1 INTRODUCTION

Test Driven Development (TDD) is an agile software engineering practice promoted by eXtreme Programming (XP) [2] which suggests a different programming way that requires writing tests before writing the code in short cycles to reduce feedbacks loops.

https://doi.org/10.1145/3197091.3197132

The mantra "Red-Green-Refactor" introduced by [3] outlines the three steps of such a cycle of development. The first step is to write a new test and check that it fails (Red) because the functionality it is testing for doesn't exist. The second step is to write quickly a code to pass all the tests (Green). The third step is to make code better by refactoring [10] or clean code [16].

As agile methodologies become mainstream in the software industry, many experience reports and surveys have been proposed about the teaching of agile software development. These studies are mainly focused on Scrum [14] and on agile collaboration and values skills [17]. Fewer learning approaches are proposed in litterature to teach TDD. Moreover TDD seems mostly to be taught to final-year undergraduates [6] although the benefits on student testing and programmer performance have already shown for early programmers [11].

As we think that students should be rise awareness as soon as possible of design's and programming's good practices promoted by eXtreme Programming, we present in this paper a centered-student learning approach that we apply to introduce TDD for first-year undegraduates. Unlike a classical Project-Based-Learning approach, the students are strongly guided at first by detailed instructions and tiny steps so they can adapt their learning progress at their own pace. In the course of time, the students become more and more responsible actors of theirs own choices. To engage students in the learning process, we also provide a concrete and engaging context with a 2D game development about a Space Invaders.

This paper is structured as follow : section 2 discusses related work. Section 3 describes the learning experience. Section 4 evaluates the quality of this learning resource and its impacts on students' perception towards TDD, good design, and programming practices. Section 5 summarizes the experience and gives final conclusions.

2 RELATED WORK

The actual approaches to learn TDD can be classified into three groups: educational games, coding dojo sessions, and practical works on projects.

Learning TDD through educational games is proposed in [12] with a A LEGO-based approach developped by agile practitioners who conducted numerous training workshops for professional software developers. This approach introduces TDD concepts more interactively and visually than a classical lecture.

Practising TDD through Coding Dojo sessions is a collaborative approach proposed by [5] for professional software developpers as a session where a group of programmers would gather to solve a code kata (a problem with further improvements) together. It is a collaborative and non-competitive environment where people can

Table 1: Space Invaders Product Backlog

Minimal Viable Space Invaders	Classical Space Invaders	Optional Features
1. Move the ship in the board	8. Fire multiple shoots from the ship	13. Add a pause
2. Size the ship	9. Add a line of invaders	14. Suggest other levels
3. Choose the speed of the ship	10. Add a score	15. Vary speed
4. Fire a single shoot from the ship	11. Fire a shot from an invader randomly	...
5. Add an invader	12. Add a wave of invaders	
6. Detect a collision between 2 sprites		
7. Finish the game		

be continuously learning and sharing agile practices [19]. Professional software developers usually organize coding dojo sessions to learn from others and to improve their design and programming skills by applying technical practices from XP. In academia, the coding dojo is commonly experimented to convey Agile software engineering best practices, to improve testing and programming skills and to improve motivation in students to learn TDD and pair programming [19]. In a coding dojo session, students have to code, explain their code and they review code of other students [9]. The limitation is that each session focuses on a specific and isolated exercise.

Learning TDD through Project-Based Learning is a strategy to explore large, open-ended problems [18]. Project Based Learning (PBL) is perceived as a student-centered approach [7, 20]. The students must to produce a solution to solve a problem and an outcome in the form of a report. PBL is based on five principles: students work together in groups; a real world problem that affects the life of the students is presented for investigation; students discuss findings and consult the teacher for guidance, input, and feedback; the maturity level of students skills determines the degree of guidance provided by the teacher; resulting products can be shared with the community. In addition to the variety of application domains, PBL promotes sudents commitment and autonomy in a collaborative learning way. Unfortunately, PBL for TDD seem to be not appropiate for first-year undergraduates which are novices and require guidance.

To engage novice students in learning TDD, we propose a project-driven approach as a PBL light. The *driven* side of this approach focus on providing a material offering a detailled guidance for the first steps of the project. According to Bloom's taxonomy [4], this *driven* side addresses the *knowledge* and *comprehension* of TDD as educational learning objectives. Then, the students will be gradually left autonomous in the development of their project and the *application* of TDD is also adressed as educational learning objective. The project-driven approach follows the recommendations of the study of Novice Programmers conducted by [13], namely learning by doing, programming by themselves, using example programs, avoiding students suffer from a lack of personal instruction, practicing sessions in computer rooms and in small groups.

Game development has been widely used in computer science education to increase students motivation, engagement and learning [22]. Moreover, as a game involves a large number of interacting objects, it is well-adapted to introduce object-oriented design [8]. As this experience is lead as a part of an object-oriented design course for novices programmers and designers, the domain application of game development improves the required engagement of students.

3 LEARNING EXPERIENCE DESIGN

In this section, we describe how to set up the experience by a project-driven approach based on a Space Invaders game development[1]. The goal of this shooting game is to destroy a wave of aliens invaders to earn as many points as possible. The invaders move back and forth across the screen, slowly advancing toward earth. They also fire randomly in a single direction. The player fires shoots from the ship restricted to a single axis of motion.

The learning objectives are the use of a part of eXtreme Programming best practices [2]: testing, refactoring, simple design, pair programming and short releases.

3.1 Kick-off

To simulate an agile development (iterative and incremental approach), the development is divided into three parts with different objectives. The first part is to deliver a minimal viable Space Invaders (only one ship, one alien invader and a single shot from ship on the screen). The second part is to deliver a classical Space Invaders (collection of aliens who fire randomly, multiple shots from ship and score). The last part is to deliver the Space Invaders with optional features as pause, levels, speed variation for example. The creativity of the students is encouraged at the last part. The two first guided parts are divided into features. The features have been prioritized and are listed in the product backlog in the table 1.

To deliver the first part, two technical tasks are necessary:

– a *sprint zero* (according in agile terminology) for the preparation of the technical environment and a quick analysis of the problem. The technical stack is composed by: Java for programming language, Eclipse for IDE, Infinitest[2] for Continuous Testing plug-in (each time a change is made, all the tests are run), git as version control system and a Github as the hosting service. The quick analysis session consist in studying the features of the backlog, according to an ubiqutous language. For example, an enemy can also be named alien or invader. We choose to use *invader* and we define it in a glossary and identify the first classes of the application as `Invader`, `Ship` and `Shoot`.

[1] http://www.classicgaming.cc/classics/space-invaders/
[2] https://infinitest.github.io

– the set up of a graphics engine and its integration in the game. We choose a light homemade graphics engine, but real graphics engine as libGDX [3] could also be convenient.

The previous tasks and the first four features of the table 1 are very detailed. The students are guided step by step to be immersed in the TDD approach and to gradually discover the refactoring tools of the IDE to increase their productivity. For the rest of the features, students are more and more autonomous to develop the game.

3.2 Focus on features' intructions

We briefly explain how the first feature *Move the ship in the board* is detailed.

First, the feature is decomposed in small *stories* which are: create a board, place the ship in the board at a certain position, move the ship to the right direction of the board, and move the ship to the left direction of the board.

Stories are implemented one by one. For each story, the same protocol is repeated: first, define the acceptance criteria and then develop using the TDD approach.

For the first story, the acceptance criteria is to obtain a 2D empty board of desired size. Then, the development begins as a succession of short cycles based on the three steps of the TDD mantra.

Step 1: Add a failing test (Make it fail) The test is given using carefully naming. It must add a new behavior. For example, the first test is:

```
@Test
public void test_anEmptyBoard_atTheBeginningOfGame() {
  SpaceInvaders spaceinvaders = new SpaceInvaders(5, 15);
  assertEquals(
  ".............\n" +
  ".............\n" +
  ".............\n" +
  ".............\n" +
  ".............\n" , spaceinvaders.toString());
  }
```

The test must be run and must fail.

Step 2: Make it work The implementation of `SpaceInvaders` class, constructor, and method `toString` is also detailed.

Step 3: Make it better (Refactor) A refactoring is a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior [10]. The question *Can my code be refactored (both the test and the production code)?* is always asked to help students reflect about their code and decide if a refactoring is necessary. Each little refactoring is preceded by questions about local design and mostly references one of the fourth eXtreme Programming simplicity rules [2]:

- pass all tests;
- clear, explicit and consistent;
- duplicates no behavior or configuration;
- minimal methods, classes, modules.

We focus on responsibility of classes and methods, naming of variables and methods, duplication of code, and the code smells long method and magic numbers.

Here only one cycle is necessary to implement this first story: *create a board*. The next iteration is concerned by the implementation of the second story: *place the ship in the board at a certain position*. First, the acceptance criteria must be defined and the implementation can start by coding the following test:

```
@Test
public void test_ANewShipIsCorrectlyPlacedAtTheBoard() {
  SpaceInvaders spaceinvaders = new SpaceInvaders(5, 15);
        spaceinvaders.placeANewShip(5,7);
        assertEquals(
        ".............\n" +
        ".............\n" +
        ".............\n" +
        ".............\n" +
        ".......S.....\n" , spaceinvaders.toString());
        }
```

A new method `placeANewShip` must be implemented in the class `SpaceInvaders`. The step of *Make it work* and *Make it better* will complete this iteration. The developpement of this second story will require several iterations because it is necessary to test that the ship can not be placed outside the board (too far right, left, up and down). Then the two last stories (move the ship to the right and to the left direction) can be implemented.

Concepts and good practices are presented, detailed and frequently repeated in all steps of iterations to make explicit the teacher's design and so transmit his knowledge and expertise. It is also indicated to commit at the end of an iteration.

Over the features, students acquire expertise. The instructions become less and less accurate until providing only the title of the feature. This point is reached from the feature 5 in our experience. At this point, the project-driven switches to a classical project-based learning. The teacher takes the role of an observator and if necessary helps student team to continously improve their work and remove impediments.

3.3 Focus on sessions' organisation

3.3.1 Material. The material includes presentation of the project, the features'instructions and a template for the outcome. The material is available from the first session. To easily access, Github is used to share on-line public material.

3.3.2 Planning. Because the project-driven approach aims to offer a self-organized learning space, no planning and no specific goal are defined for any session. By analogy with agile development, the project-driven approach can be seen as an implementation of a Kanban pull system [1]. Each team takes a new feature (new concepts to understand and implement) when the previous one is finished. Each team manages his own workflow, the only goal being to deliver the minimal skillable learning objective at the end of the last session. As the material is avalaible on line, each student can reworks some parts whenever he wants. To visualize the workflow, the commits history can be consult and an outcome is required to summarize the progression.

3.3.3 Outcome. Two outcomes are required per session. At the end of a session, students must push their last commit on their Github repository. For the next session, a summary has been required per team. This summary was based on a template with the following headings : the list of the implemented features and their acceptance criteria, the class diagram of their project (in reverse-engineering using Object Aid UML[4] for example), a word cloud of

[3] https://libgdx.badlogicgames.com

[4] http://www.objectaid.com

their production code (using Source Code Word Cloud Generator[5]), the difficulties they have encountered during the session (optionnal) and, any comments they find useful. The class diagram helps students to be aware of the emergence of the design over the iterations, the word cloud helps students to improve the naming and code readibility because only domain terminology should appear to reflect the code intent.

3.4 Assessment

Two types of assessment are used : report and review. During the last session, the teacher made a code review of the work done by the team after a demonstration of the game. A discussion is engaged to help students to demonstrate their knowledge and skills and to identify any problems. At the end of the experience, a report including a presentation of the project and all the summaries has been delivered by each team.

4 LEARNING EXPERIENCE EVALUATION

This experience has been designed and delivered as a part of the basis of an object-oriented design course. This course is a part of French National Pedagogical Program (PPN), a common program to all technical colleges specialized in Computer Technology. The first part of this course focus on classical design with lectures and exercices about object modelling for analysis and design (class diagrams, sequence diagrams). The laboratory is the second part of this course. A total of 40 one-year french undergraduates attended 7 sessions with two 2-hours sessions per week. Students are novice programmers: they have started learning Java for only a few weeks. They are novice designers too.

On one hand, the evaluation concerns the efficiency of this learning resource by the student engagement and the quality of the project-driven approach. On the other hand, it concerns the impacts on student's perception towards TDD, good design, and programming practices. This evaluation is a combination of questionnaires and open-ended questions. The survey is avalaible online and is completed anonymously per each student at the end of the last session.

4.1 Student's engagement

For the first question of the survey, students must indicate the last shipped feature. The answers are:

- Fire a single shoot from the ship (feature 4): 2 students
- Detect a collision between 2 sprites (feature 6): 2 students
- End the game (feature 7): 10 students
- Fire multiple shoots from the ship (feature 8): 8 students
- Add a line of invaders in the game (feature 9): 8 students
- Fire randomly a shot from an invader (feature 11): 6 students
- Add a wave of invaders (feature 12): 4 students

These results show that 90% of students achieve the minimal skillable learning objective and even 65% of students exceed this objective by shipping even beyond feature 7. The choice of developing a game seems to have motivated the students to achieve the objective. As the students were in pair-programming, these results also show that 2 groups of students were in difficulty, mainly because of their difficulties to code in Java and their lack of comprehension and practice of algorithmic techniques. The development was originally only in-class, but the desire to play motivated some students to continue their work in their own time between sessions. Moreover some students have planned to continue this game development as a side project.

[5] https://sourcecodecloud.codeplex.com/

4.2 Student's perception towards the quality of the learning resource

The evaluation of the project-driven approach has been adapted from a model initially presented in [21]. This model aims to assess the quality of a learning ressource (initially an educational game) through the students' perceptions about levels of motivation, user experience and learning promoted by this ressource. Results of our questionnaire are presented in table. 2. It consists in 17 items on a Likert scale with response alternatives ranging from strongly disagree (-2) to strongly agree (2).

4.2.1 Motivation. Overall, students perceived a positive contribution of the experience to motivate them to study. The students also indicated that the laboratory captures their attention: especially, the relevancy of laboratory content and its connection with other knowledge. Regarding the confidence dimension, the laboratory helped student to confident that they were learning, yet taking account the negative ratings of the item on the ease of understanding. Beyond the TDD technique, the lack of oriented-object programming practice can also explain difficulties for some novice programmers. The majority of the students also confirmed that the learning content is relevant to their professional work.

4.2.2 User Experience. The user experience has been rated very positively by the students. This demonstrates that they experienced the laboratory as a positive and engaging learning approach. Students would like practise this kind of lab again (with a game development) which obtain 80% of approbation, the highest rating of this questionnaire. Overall, results are positive in terms of fun, challenge, and social interactions. Regarding the competence dimension, 70% of the students believe that the game has been an efficient way to learn. They also confirmed that they had fun with the activities of this lab. In terms of challenge, the students found the laboratory moving at an adequate pace. The majority of the students agreed that the course promotes moments of cooperation.

4.2.3 Learning. The course seems to be relevant to the needs of the students. Students expressed that they believe that the lab contributed positively to their learning and 90% of them indicate that it helped them to learn TDD in an efficient way. As Students are only first undergraduates, it could be difficult to estimate if this lab will improve their professionnal performance in practice even if 50% of students had approved and perceived the impact of TDD.

4.3 Student's perception towards the TDD benefits and disadvantages

From Kent Beck's point of view, Test-Driven Development (TDD) really encourages simple designs and inspires confidence [3]. The students were asked to give their own opinion on this point as novice programmers and designers. The results confirm Kent Beck's

Table 2: Online survey results for evaluation of the quality of project-driven approach

Questions	-2 s. disagree	-1 disagree	0	1 agree	2 s.agree
Motivation					
There was something interesting in this course that captured my attention	0%	2.4%	19.5%	46.3%	31.7%
The way the lab works suits my way of learning relevance	2.4%	17.1%	24.4%	39%	17.1%
The lab content is connected to other knowledge I already had	0%	7.3%	22%	56.1%	14.6%
Domain experience are relevant to my interests and my needs	0%	7.3%	36.6%	41.5%	14.6%
As I worked on this lab, I felt confident that I was learning	4.9%	7.3%	34.1%	34.1%	19.5%
It was easy to understand TDD and start using it as study material	2.4%	9.8%	19.5%	34.1%	34.1%
I am satisfied because I know I will have opportunities to use in practice things I learned during this lab	0%	7.3%	29.3%	53.7%	9.8%
User Experience					
The lab was an efficient way to learn	0%	2.4%	26.8%	51.2%	19.5%
I had fun with the activities of this lab	4.9%	4.9%	19.5%	43.6%	24.4%
I would like to practice this kind of lab again (with a game development)	7.3%	4.9%	9.8%	39%	39%
The lab is properly challenging for me, the tasks are not too easy nor too difficult	0%	9.8%	34.1%	39%	17.1%
The lab progresses at an adequate pace and does not become monotonous - offers new obstacles, situations or variations in its tasks.	0%	7.3%	24.4%	46.3%	22%
I was able to interact with others during this lab	2.4%	12.2%	22%	36.6%	26.8%
The lab promotes moments of cooperation and/ or competition between the students	7.3%	12.2%	17.1%	51.2%	12.2%
Learning					
The lab help me to learn TDD	2.4%	2.4%	4.9%	65.9%	24.4%
This experience with this lab will improve my professional performance in practice	0%	12.2%	39%	29.3%	19.5%
The lab has been effective for my learning, comparing it with other class activities of a traditional teaching approach	2.4%	7.3%	17.1%	48.8%	24.4%

vision: 75% feel more confident about their code and 85% feel to improve the quality of their code.

To help student reflect on the benefits and disadvantages of TDD, two open-ended questions have been added: Why will you use TDD in your future developments ? What are the disadvantages of using TDD in a software development ?

In terms of benefits, students have identified TDD as “a method easy to follow” which “avoids errors and allows to obtain a cleaner and more understandable code”. They think that “testing and refactoring help to improve the readabilty of the code and thus make it possible to easilier change it.” Students appreciate “naming of the tests”, “code readibility”, that “each feature works in astonishment”. One student says “Usually, I code for a long time without testing. This kind of development can help me fix this mistake” and an other adds “this saves us from errors due to our precipitation to code”.

In terms of disadvantages, students find the TDD approach difficult to apply and “time-consuming” : “I have a lot of difficulty in writting unit tests”, “Time spent writing tests”and they add that “TDD requires a minimum of computer expertise to be effective”. Effectively, test code has the same value than production code: it requires tought, design, and care [16] and the emergence of a simple design is more efficient by the knowledge of advanced concepts as SOLID principles and design patterns.

4.4 Student’s awareness of good design and programming practices

We also use open-ended questions to help students reflect on testing and refactoring. For each of these techniques the questions are: Why will you use *this* in your future developments ? What are the disadvantages of using *this* in a software development ?

4.4.1 Testing. The first benefit highlighted by most of the students is to use testing as a verification tool which ensures the non-regression. They are now aware of the importance of the automation of tests in any software development. Some are more accurate :“they mainly allow me to avoid regressions and serve as documentation of the code”, “Tests help to better understand a problem and write a more efficient code”, “Make sure to write a code that meets the needs of our test”, “Make it possible to predict errors and/or to easily understand them when one is encountered” for example. Overall, the answers reflect the following thought from [15] *the act of writing a unit test is more an act of design than of verification. It is also more an act of documentation than of verification.* As mentioned earlier, the main disadvantage raised by students is writing tests is time-consuming and some students precise “to want test long and complicated applications but not simple applications”.

4.4.2 Refactoring. Students have a positive feeling about refactoring: "The refactoring steps helped me a lot". The readibility and code quality is mentioned by almost all students. Some add: "Refactoring allows a better understanding of the code, not only by others, but also by ourselves", "allows to reflect about the most effective implementation", "gives us the impression of improving our code". Students don't really notice any disadvantage for refactoring but rather difficulties to set up it: "refactoring is the hardest part of TDD for me, I find it really useful when you need to reuse code. But I still have trouble putting it in place", "Identify where to perform refactoring is difficult".

4.5 Global student's perception towards the experience

Two open-ended questions ended the survey to obtain feedback and improvement suggestions. For this experience, students most appreciate:

- **the "fun"** dimension of the experience and the choice of "developing a game from A to Z";
- **the structured and detailed dimension** that offers the opportunity to "progress at its own pace": "structures steps allowing a better reflection (a guideline)", "easy to understand", "progressive learning", "time to understand", "All the first steps are extremely detailed which allows to return if we had errors".

To improve the experience, two points have been underlined. Students would like that the "lack of instructions should be more progressive". Students would like "more sessions" to develop advanced features and play with a more complex Space Invaders.

4.6 Teacher's point of view

At the end of the course, the teachers organized a *retrospective* during which they discussed the experience and considered possible future improvements. They appreciated the challenge side of the TDD (green bar-red bar) which prompted students to get involved more than in a traditional lab. They have a positive feeling that the small steps of this project has prompted students to regularly commit and push and it was also a good introduction to practice git. They found that students consistently and seriously provided their outcomes and were pleasantly surprised by the students reflect on TDD techniques.

Moreover, they observed that the students had difficulty in writing tests for detecting a collision, which shows that the act of design is not easy to practice and requires more training. Teachers also plan to set up code review sessions to reduce the gap between the last shipped features.

5 CONCLUSION AND FUTHER WORK

In this work, we present an experience in learning Test Driven Development. Because teaching a new programming way is challenging, we propose a game development project-driven.

The project-driven approach offers a self-organized learning space. The *driven* side of this centered-student approach offers step-by-step detailed instructions for students to learn by doing. The *project* side gives students autonomy to organize their learning process at their own pace. The context of game development engage students in the learning process. By public sharing online all the instructional material, the students will be able to self-organize by building their own pedagogical progression, by imposing their own challenges. Because each student has his own abilities, a minimal skillable learning objective is set to give everyone the opportunity to deliver a minimum viable product.

At the end of the experience 90% of the students achieve the minimal skillable learning objective and 90% of the students are agree that this learning approach help them to learn TDD. Moreover, more than two out of three students feel that TDD inspires confidence and improves code quality. We will continue this experience for the advanced object design and programming course where SOLID principles and design patterns will be studied.

REFERENCES

[1] David J. Anderson. 2010. *Kanban: Successful Evolutionary Change for Your Technology Business.* Blue Hole Press.

[2] K. Beck. 2000. *Extreme Programming Explained: Embrace Change.* Addison-Wesley.

[3] Ken Beck. 2002. *Test Driven Development: By Example.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[4] B. S. Bloom, M. B. Engelhart, E. J. Furst, W. H. Hill, and D. R. Krathwohl. 1956. *Taxonomy of educational objectives. The classification of educational goals.* Longmans Green.

[5] Laurent Bossavit and Emmanuel Gaillot. 2005. The Coder's Dojo – a Different Way to Teach and Learn Programming. In *Proceedings of the 6th International Conference on Extreme Programming and Agile Processes in Software Engineering (XP'05).* Springer-Verlag, 290–291.

[6] Jon Bowyer and Janet Hughes. 2006. Assessing Undergraduate Experience of Continuous Integration and Test-driven Development. In *Proceedings of the 28th International Conference on Software Engineering (ICSE '06).* ACM, 691–694.

[7] S Chandrasekaran, A Stojcevski, G Littlefair, and M Joordens. 2012. Learning through projects in engineering education. In *Proceedings of SEFI Conference.*

[8] W. K. Chen and Y. C. Cheng. 2007. Teaching Object-Oriented Programming Laboratory With Computer Game Programming. *IEEE Transactions on Education* 50, 3 (2007), 197–203.

[9] R. B. d. Luz, A. G. S. S. Neto, and R. V. Noronha. 2013. Teaching TDD, the Coding Dojo Style. In *2013 IEEE 13th International Conference on Advanced Learning Technologies.* 371–375.

[10] Martin Fowler. 1999. *Refactoring: Improving the Design of Existing Code.* Addison-Wesley.

[11] David Janzen and Hossein Saiedian. 2008. Test-driven Learning in Early Programming Courses. In *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education (SIGCSE '08).* ACM, 532–536.

[12] Stan Kurkovsky. 2016. A LEGO-based Approach to Introducing Test-Driven Development. In *ACM Conference on ITiCSE.* New York, NY, USA, 246–247.

[13] Essi Lahtinen, Kirsti Ala-Mutka, and Hannu-Matti Järvinen. 2005. A Study of the Difficulties of Novice Programmers. In *Proceedings of the 10th Annual SIGCSE Conference on ITiCSE (ITiCSE '05).* ACM, 14–18.

[14] Viljan Mahnič. 2015. Scrum in software engineering courses: an outline of the literature. *Global Journal of Engineering Education* 17, 2 (2015).

[15] Robert C. Martin. 2002. *Agile Software Development: Principles, Patterns, and Practices.* Prentice Hall PTR.

[16] Robert C. Martin. 2008. *Clean Code: A Handbook of Agile Software Craftsmanship* (1 ed.). Prentice Hall PTR.

[17] A. Meier, M. Kropp, and G. Perellano. 2016. Experience Report of Teaching Agile Collaboration and Values: Agile Software Development in Large Student Teams. In *2016 IEEE 29th International Conference on Software Engineering Education and Training (CSEET).* 76–80.

[18] M. Missiroli, D. Russo, and P. Ciancarini. 2016. Learning Agile Software Development in High School: An Investigation. In *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C).* 293–302.

[19] Danilo Toshiaki Sato, Hugo Corbucci, and Mariana Vivian Bravo. 2008. Coding Dojo: An Environment for Learning and Sharing Agile Practices. In *Proceedings of the Agile 2008 (AGILE '08).* IEEE Computer Society, 459–464.

[20] John R. Savery. 2006. Overview of problem-based learning: definition and distinctions, The interdisciplinary. *Journal of Problem-based Learning* (2006), 9–20.

[21] Christiane Gresse von Wangenheim, Rafael Savi, and Adriano Ferreti Borgatto. 2012. DELIVER! - An Educational Game for Teaching Earned Value Management in Computing Courses. *Inf. Softw. Technol.* 54, 3 (2012), 286–298.

[22] Alf Inge Wang and Bian Wu. 2015. *The Use of Game Development in Computer Science and Software Engineering Education.* CRC Press, Taylor and Francis.