Software Kaizen: Using Agile to Form High-Perfomance Software Development Teams

Bernardo Estácio, Rafael Prikladnicki, Michael Morá
Computer Science School, PUCRS
Porto Alegre, Brazil
bernardo.estacio@acad.pucrs.br,
rafael.prikladnicki@pucrs.br,
michael.mora@pucrs.br

Gabriel Notari, Paulo Caroli
Thoughtworks Inc.
Porto Alegre, Brazil
gnotari@thoughtworks.com,
paulonotari@gmail.com

Alejandro Olchik
Ionatec
Porto Alegre, Brazil
aolchik@ionatec.com.br

***Abstract*— The process of teaching Software Engineering has undergone questions about the methods that have been used in training activities. Recent studies show that these methods involve traditional teaching strategies, such as presentation of theory and lectures. For this reason, students usually find in industry a different scenario than what is taught in the classroom. In parallel, other studies indicate that the emergence of agile methods in the 90s led to the formation of high performance teams with great level of knowledge in technical, business and behavioral domains. For this reason, we have proposed a training method called Software Kaizen, which provides temporary immersion of a team in a high-performance environment, based on agile methodologies. This paper presents the method and the results obtained from its application. We report on four replications of the method, with good results in learning, posture change and teamwork, some of the expected characteristics of high-performance teams.**

Keywords*— *software development, high-performance teams, agile methods training, university-industry collaboration.

I. INTRODUCTION

The development and training of qualified professionals are increasingly required nowadays. At the same time, the process of teaching Software Engineering (SE) has undergone questions about the methods that have been used in training activities. Recent studies show that these methods involve traditional teaching strategies, such as presentation of theory and lectures [1]. Other studies indicate that the emergence of agile methods in the 90s led to the formation of high performance teams, with great level of knowledge in technical, business and behavioral domains [2, 3].

Parker and Jackson understand that high-performance teams are formed by groups that rely on each other, are committed to planning and execution, base their actions on a common vision, and develop activities through open communication [4]. This definition is consistent with the principles and values proposed in the Agile Manifesto [5].

For this reason, and based on the growing demand for training of high-performance teams and the opportunity to pursue innovative training strategies in SE, the goal of this paper is to present Software Kaizen, a training method that provides temporary immersion of a team in a high-performance environment, based on agile methodologies. The method was proposed to be a pioneering training program in the context of an innovative recruitment process as well as a complementary course in the university curriculum. Through an immersion in an environment mentored by experienced professionals, the selected students become members of high-performance software development teams. The proposal is based on well-known agile methods such as Scrum, Kanban, and Extreme Programming (XP). During five two-week development iterations, the team develops skills in dimensions such as business, governance, technical and behavioral.

In this paper we describe the concepts and the program in details, including the results obtained in four consecutive instances (four months each) where Software Kaizen was applied to groups of students and professionals. In the first two instances, for example, the team velocity improved by 230% and 200% respectively. Overall the four instances presented above average results in terms of agile practice adoption, and team behavior. We also share lessons learned, presenting the main benefits and challenges identified.

The remainder of this paper is structured as follows. In Section II we present background information and related work. Section III describes our research methodology. The Software Kaizen method and its application are presented, respectively, in Sections IV and V. In Section VI we discuss the results and present lessons learned. Finally, we conclude in Section VII.

978-0-7695-5222-4/14 $31.00 © 2014 IEEE
DOI 10.1109/AGILE.2014.10

II. BACKGROUND AND RELATED WORK

A. High-perfomance teams

High-performance teams are formed by groups who rely on each other, base their actions on a common vision, develop their activities through open communication, build confidence, and have shared leadership, enabling innovation from individual differences [4]. This is complemented by the study of Roda [3], which presents a model of three levels for self-organizing teams: creating, practicing and transcending. The high-performance teams are at the last level and are characterized by technical and behavioral excellence, practicing and experimenting challenges continuously. A high performance team must have autonomy, attitude and more productivity than a traditional team and usually have great satisfaction in the work they do.

For this reason, the concept of high-performance team has a strong relationship with self-organization [6]. The literature indicates that the use of agile methodologies such as Scrum or XP requires the formation of high-performance teams [4]. Authors also reported that some of the agile practices such as retrospectives, daily stand-ups and the adoption of Kanban improved the autonomy and shared leadership, characteristics of high-performance teams [6].

B. Software development education and training

The literature reports that the common approaches in teaching agile concepts include lectures [7, 8], the use of games [1, 9], studio and Capstone projects (the execution of a project from start to finish) [10]. In professional agile training Agile is taught in short courses, some of them in the company environment and others are preparatory courses for certifications exams [9].

In academia, we have identified experiences with agile in teaching at both undergraduate and graduate levels [8, 9, 10], but few studies report on agile training using the concept of temporary immersion of project teams. Sutherland et al. [11] reported from an immersion training at two companies, MySpace and Jayway. The method is called Shock Therapy, and the goal is to bootstrap for high-performance teams in Scrum. In this study, an experienced coach created an immersion environment, and the results showed 240% of improvement in team velocity in a few weeks.

Our work is inspired by Shock Therapy, but it is different as it is not limited to Scrum and also combines academia (students, professors) with industry (company, professional, developers), promoting rich ecosystems not only for teaching, but also for researching.

III. RESEARCH METHODOLOGY

This research was conducted following the methodology for the incremental evaluation of a new process, proposed by Shull et al. [12]. The methodology has four mains activities: feasibility study, observational study, a case study in a real lifecycle and a case study in industry. Figure 1 presents an overview of this methodology.

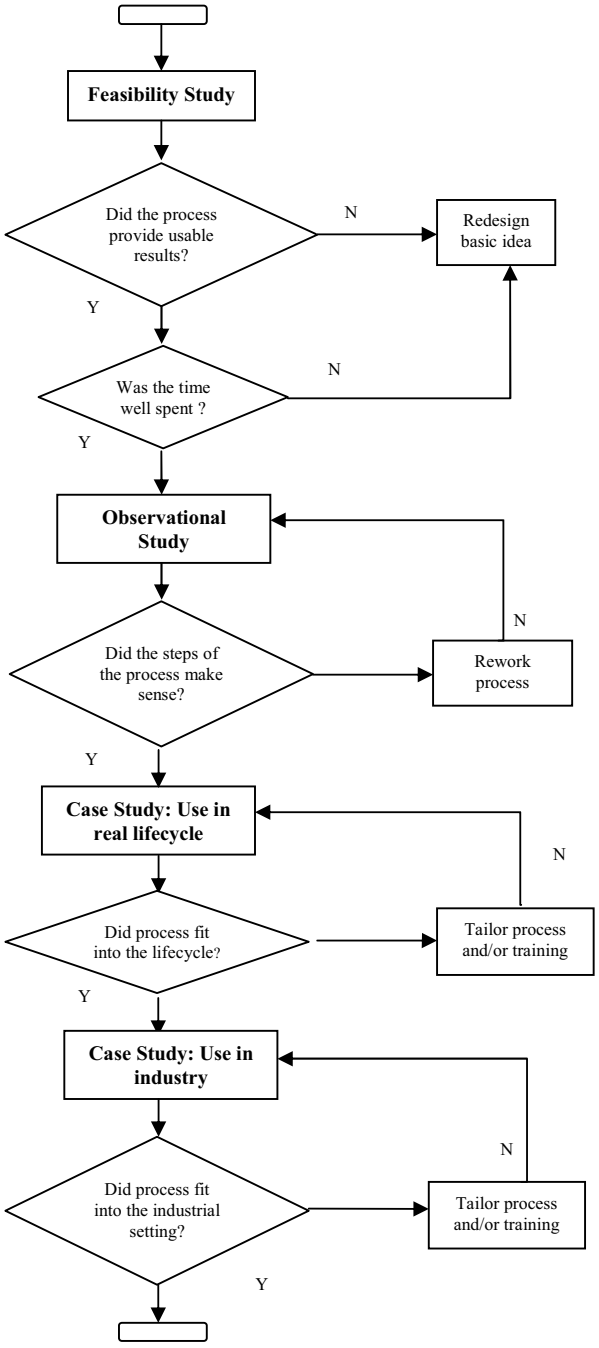

Figure 1. Methodology overview [12].

In the feasibility study the purpose is to evaluate if it is worthwhile to spend the resources required to continue through the methodology. The focus in this step is on generating rather than testing hypotheses about the new process and its usefulness.

In the observational study the main goal is to begin evaluating the steps in the new process to ensure that each one is effective and that the order in which they are executed makes sense. Observational techniques can be used to

understand current work practices that can be incorporated into the new process.

In order for the process to be useful, it has to be able to fit into a real development lifecycle. And in this case one can execute a case study. Once the process is tailored to be usable within a real development lifecycle, the next step is to use the new process in an industrial setting. This last step can also be executed through a case study. Figure 1 presents an overview of this methodology.

We executed our research as follows:

- Literature review and initial proposal of Software Kaizen: we performed a literature review in software engineering education and agile software development, in order to identify strategies in software engineering education and characteristics of training with high-perfomance teams. After that an initial version of the method was proposed;
- Feasibility study: we planned and executed a survey including 150 interviews from 100 IT companies in Rio Grande do Sul (a state in Brazil) aiming at evaluating the feasibility, benefits and risks of the Software Kaizen method.;
- Observational study: based on the feasibility study, we improved the Kaizen method and planned two observational studies with students;
- Case study in a real lifecycle: since the pilot study with students, we have planned and executed two case studies in a real lifecycle.

The research was executed in a collaborative initiative that includes a university, a technology park, and two partner companies, respectively, PUCRS, an university located in Porto Alegre, Tecnopuc, PUCRS's Technology Park, DBServer and ThoughtWorks, both leading companies with offices at Tecnopuc.

IV. SOFTWARE KAIZEN: THE METHOD

Kaizen is a Japanese word meaning continuous improvement and aims not only to increase productivity but also to harmonize the work environment and the systematic elimination of waste [13]. In this sense, Software Kaizen aims to assist in forming high-performing teams for software development, based on agile methodologies. The method is characterized by temporary immersion in a project team environment with experienced professionals who prepare participants in technical, behavioral, business and governance aspects. The method is based on well-known agile methods such as Scrum, Kanban, and XP. In five two-week iterations the team develops competencies in dimensions such as business, governance, technical and behavioral. The method is inspired by the immersion environment proposed by Jeff Sutherland called Shock Therapy, and provides learning, agile coaching and peer mentoring sessions. Through an open application process, the participants take classes made up of small teams.

We support them with mentoring, training and workshops for a definite period (usually three months), in exchange for building a non-profit application. While traditional teaching classes are often university-driven, generally require industry mentoring, and focus on standard learning, curriculum building, grading or traditional recruiting companies, the Software Kaizen is focused on a real software deliverable.

After receiving the results of two observational studies (the observation studies were executed in 1 preparatory iteration and 3 development iterations, with a duration of one week each), we improved the Kaizen method in a second version with Scrum as a framework for continuous improvement. We defined the method with 2 preparation iterations and 5 development iterations lasting two weeks that involve coaching in agile methodologies, training session and interaction with the support team. We then define the Kaizen method in 4 phases:

- Phase 1: Pre-evaluation: recruitiment process of a team with complementary skills in order to encourage learning between the students, providing experience in a multifunctional team. It has a duration of 3 three weeks.
- Phase 2: Iteration -1: this phase lasts for 2 weeks, and is when the environment is set up with the technology that will be adopted. This phase also includes a preparatory tecnhical training for the students, with teaching assistants and mentors supporting the team with Coding Dojos [14] and providing materials (tutorials, guides, etc.).
- Phase 3: Iterarion Zero: in this phase the team takes a test about their initial stage of agile method adoption [15]. Additionally, an initial training begin with topics such as goal alignment, backlog preparation, responsibility definition, and definition of the concept of being done. The initial training involves 36 hours of agile training with 4 six-hour sessions under the supervision of a teaching assistant.
- Phase 4: Iteration 1 to 5: iterations of 2 weeks, with a 4-hour coaching focusing on planning, review and retrospective of the work done; 2 hours of training and coaching focusing on the team's needs. In the last iteration (iteration 4) we have a retrospective of 2 hours to evaluate the training program.

The roles are defined as follows:

- Students (or Professionals): responsible for developing the product, executing coding activities, user interface, architecture, quality assurance. During the training, a student (or professional) is responsible for maintaining frequent communication with the product owner.
- Coach: responsible for pre-evaluation of the students, as well as conducting the planning, review and retrospective of each iteration. The coach also supports supervision of the teaching assistants and conducts the final evaluation of the course.
- Teaching assistant: responsible for helping the students to prepare the development environment, solving impediments, helping the team in technical questions and collaborating in the coaching activity.

- Mentor: reponsible for supporting the team during the project. The mentor could be a developer of a company that will often support and collaborate with the trainning.
- Product owner: responsible for defining the requirements list, prioritizing it and assessing the requirements delivered at the end of each iteration.

The product can be real or fictional, defined by an internal or external customer. Moreover, we defined the environment beforehand, including the technology used and the continuous integration tools, source code versioning, document sharing, static code analysis, among others.

During the training program, we collect a set of metrics in order to monitor the performance of all students. The final evaluation includes an assessment given to each individual student, pointing out strengths and opportunities for improvement, overall evaluation of the group and a final evaluation of the course by each of the students.

V. PUTTING SOFTWARE KAIZEN INTO PRACTICE

A. Feasibility study

We executed the feasibility study between May and September of 2011, in collaboration with DBServer, a Brazilian software development company (www.dbserver.com.br). We did 150 interviews with 100 companies, these being 60 IT companies and 40 companies that have an IT department. In relation to the training methods, 56% of the companies said that they plan their training outside the company, 33.3% said that they plan the training in company and 17.3% said that they plan pilot training project training in company. Regarding the pilot training projects, the main benefit reported was regarding the use of this initiative to invest in continuous improvement of the software development process.

When we asked about the problems that the companies face, 46% of the respondents said that they face a lack of skilled labor and 14% said that they have problems delivering projects on time. Furthermore, among the companies that use a prescriptive model for software development (such as a waterfall lifecycle), 41% said that the adoption of that model is a cultural aspect of the company and 10% said that it is a customer requirement. On the other hand, among the companies that use an adaptive model for software development (such as an iterative lifecycle), 22% said that it is a customer requirement, 17% said that it is necessary to get better results and 12% reported that the model embraces changes.

We also asked about the Software Kaizen, to which 81.1% responded that the model is innovative and interesting. Moreover, 78.3 % of the respondents suggested that the immersion could be applied to other topics (beyond agile software development) such as project management and product management. Regarding possible challenges to adopting the method, 50.7% mentioned possible high costs and 37.7% said that it would be difficul to have the team out of the workspace for a long period of time.

B. Observational Study 1

The main purpose of the observational study was to evaluate the application of Software Kaizen. We performed this study with a team of five students at PUCRS. The team had developers working on software development activities for external customers. The study included a teaching assistant, three mentors (two of them professors) in the coordination and a coach responsible for the training.

During four weeks, the team aimed at developing a minimal version of a system for evaluating students. The client was a University department. The technology platform was .NET/ MVC. At the beginning and end of the course the team answered a self-assessment of their adoption of agile methods. Figures 2 and 3 present the results and a significant improvement in the adoption of agile practices. They used a template provided at http://www.agileassessments.com/.

Figure 2. Initial assessment of the observational study 1

Figure 3. Final assessment of the observational study 1

The main indicator of team performance was the speed per iteration. As iteration zero has been devoted to the initial preparation of the team and to planning the backlog of the product, the speed was measured in iterations 1, 2 and 3. From iteration 1 to 3 one can observe an improvement of 233% in the speed (Table 1).

Table 1. Metrics collected during the observational study 1

	It. 1	**It. 2**	**It. 3**
Velocity	3	6.25	10
Code coverage	45%	81.64%	81.97%
LOC	1.072	1,985	2,840
Unit tests	50	112	146
Functional tests	1	2	2
Commits/week	62	72	70
Build duration (seconds)	10	23	30
Status of the continuous integration	72.73%	100%	100%
% stories done in pair	75%	0%	25%

In the first iteration the team still had no knowledge of the continuous integration process and had never implemented an automated deployment. The concept of done defined was: acceptance criteria met the story, story accepted by the Product Owner, story accepted by the coach; story free of

defects, 80% of code coverage and at least one story with an automated scenario. Regarding pair programming, as in the first iteration the team used this technique to develop several stories and was already facing major technical difficulties in learning other practices, the coach decided not to use pair programming as a mandatory practice. The result was that the group significantly reduced the use of the practice in the iterations 2 and 3. The decision to use pair programming was due to the type of story and the individual preference of each team member.

We also collect testimonials of the students at end of the study. One student commented about working in a team:

"When I began the course, I did not know how to work on a team. I did not have trust in teammates and I only cared about the defects that were coded by me. At the end of the course, I discovered that the Agile methodology greatly facilitates our in working together, I learned to trust in the team and to care about the project goal."

Software Kaizen also helped the students in quality improvement and in self organization of the team, as another student reported:

"The temporary immersion of the Kaizen method helped us to adopt a quality standard for the product we were developing. We had the opportunity to adopt a standard for team self-management as well."

C. Observational Study 2

After the first experience with Softwre Kaizen in an observational study, we planned and executed a case study with the purpose of tailoring the method and set it for the industry. Six students from PUCRS formed the team that also included a teaching assistant, a coach and two professors in the coordination of program training .

The training had 4 weeks of duration, the same as the first observational study. In this second edition, the team aimed to develop a minimal version of a system for government; the client was the regulatory agency for Brazilian graduate programs (Capes). The technology platform was .NET/ MVC. The team also answered a self-assessment of their level of agile methods adoption. Figure 4 and 5 present the initial and final agile assessment results, respectively.

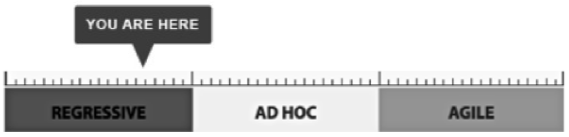

Figure 4. Initial assessment of observation study 2

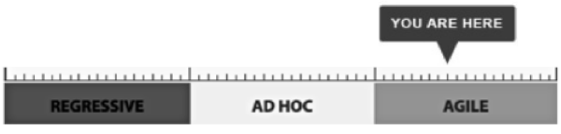

Figure 5. Final assessment of observation study 2

As in our first study, the main metric of performance was the velocity of the team. In this second edition we also considered only the range of iteration 1 to 3, because iteration 0 was to set up the environment. We observed an improvement of 200% in team velocity from iteration 1 to 3 (Table 2).

Table 2. Metrics collected during observational study 2

	It 1	**It 2**	**It 3**
Velocity	8	18	24
Code coverage	73.42%	99.67%	88.37%
LOC	603	1122	1565
Unit tests	30	100	146
Functional tests	0	0	1
Commits/week	74	59	80
Build duration (seconds)	51	75	138
Status of the continuous integration	72.73%	72.73%	72.73%
% stories done in pair	33%	15%	44%

In order to measure how the students felt about their experience in the project, we introduced an important metric: the happiness index [16]. We collected it at the end of each iteration and during the retrospective sessions; it indicates the team's sense in relation to the activities of the training. Figure 6 shows the happiness index collected.

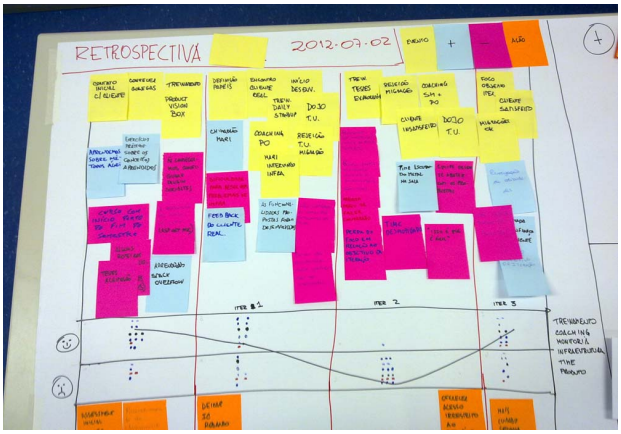

Figure 6. Happiness Index

Figure 6 shows the happiness index at the bottom. The index indicates a common behavior in building teams. While the first iteration presented a high happiness index, probably because of everything was new and the team had high expectations, we observed that during the second iteration this index fell significantly, indicating a possible cultural change and resistance, which ended up reversing in the last iteration. This behavior is commonly found in building teams theories, such as the Tuckman Theory [17].

In this second observational study we also collected testimonials to measure the impact of the training among the participating students. One of the students said:

"The course was a paradigm shift, because I was used to developing alone. Working as a team, I could see that the tasks can gain greater speed and quality, if they are well distributed and if the communication is clear and explicit."

The student who performed the Product Owner role said:

"Through the course, I learned about agile methods and mainly I learned about the role of Product Owner, that I could perform this role during the project."

D. Use in a real lifecycle: case study 1

In the third edition of the Software Kaizen we have planned and executed a case study in a real lifecycle. We received a total of 130 applications. For this edition we began a partnership with ThoughtWorks, a global software delivery and products company (www.thoughtworks.com). The company is closely associated with the movement for agile software development, and has contributed to a range of open source products. The students worked in the same environment and were immersed for several iterations. The team participated in developing a piece of a social impact project. The piece involved a feature of an open source project for managing hospitals in poor cities around the world. The project used Java as the main programming language and several frameworks and technologies such as Spring MVC, JQuery, Hibernate and others. In this third edition the students also answered the agile assessment. Figures 7 and 8 show the results at the beginnig and at the end of the course.

Figure 7. Initial assessment of the case study in a real life cycle 1

Figure 8. Final assessment of the case study in a real life cycle 1

Due to the complexity of the project, two developers from the company helped the students in technical aspects and two teaching assistants focused on monitoring evolution and solving impediments. The product owned role was played by a business analyst from the company. Two professors and a coach were reponsible for the training.

During the project, we faced several challenges related to dependencies. The main reason was the open source nature of the project. For instance, to develop a feature in the system, the students must understand the existing features and this process influenced the velocity of the team, although it reflected a real life cycle setting. Table 3 shows the results of the team.

Another metric that we collected was the Net Promote Score (NPS) [18]; we have chosen this metric in order to analyze the satisfaction level in relation to team performance from the point of view of the support team. The support team was considered to be customers of the students as well, so we classified the metric results in three categories: NPS of PO, NPS of Mentor and NPS of teaching assistants. Table 4 shows the results (the NPS range defined was 0-10).

Table 3. Metrics collected during the real lifecycle case study 1

	It 1	**It 2**	**It 3**	**It 4**	**It 5**
Velocity	0	0	21	10	8
Code coverage	20.4%	23.3%	22.3%	22.2%	22.2%
LOC	436	684	705	722	717
Commits/week	0	0	11	31	4
Build duration (seconds)	150	120	216	169	175
Status of the continuous integration	13%	38%	88%	88%	88%
Hapiness Index	3	4.2	3	3.2	3.2

Table 4. Table 4.NPS collected in the real lifecycle case study 1

NPS	**It 1**	**It 2**	**It 3**	**It 4**	**It 5**
PO	5	4	6	6	6
Mentors	-	4	5	5	6
Teaching assistant	-	8	8,5	7	3

We began to collect NPS from iteration 1, because from that iteration on we had deliverables for the project. As teaching assistants and mentors had little participation in iteration 1, they did not participate in the NPS. In the last iteration, the teaching assistants gave a low score to the team, as they reported a lack of commitment to the project.

In this study, we did not measure functional tests and unit tests. We focused on TDD in iteration 1, but the feature developed was a front-end requirement, so the students faced difficulties in developing that and they chose not to do so. In this study, all stories had to be done using pair programming, so we did not measure the % stories done in pairs. The happiness index presented a similar behavior to that of the previous studies.

At the end of this study, we also collected qualitative feedback from the students. One of them said:

"I have many difficulties in communicating with other people and agile methods need strong communication. With the Kaizen method, I learned not only about technical aspects but how my voice and participation was important to the team".

Another student reported:

"I learned a lot of TDD and about programming techniques. The experience of being in the same place every day in a week helped us to make not classmates, but friends."

E. *Use in a real lifecycle: case study 2*

The second real lifecycle study followed the same configuration as in the case study 1. We had improved the recruitment process, focusing more on collaboration aspects, so as a recruitment step we included a Coding Dojo to assess the candidates from a teamwork perspective. This time we also received a lot of applications, 160 altogether, and selected 7 students. In this study we transferred to the students the power to decide the project that they to wanted work on, so we executed an ideation, inviting the participants (students and mentors) to show and discuss ideas. The students selected a system for monitoring the draft laws from the Porto Alegre city hall.

We replicated the same number of iterations as in the previous study. The project used several tecnologies such as Ruby, Angular JS and Mongo Db. The students also answered an agile assessment, but in this edition we adopted the agile evaluation proposed by James Shores [19], because the other assessment had been discontinued. In this assessment there is a scale from 0 to 100 and there is a classification of the agility of a team in terms of: collaborating, developing, thinking, planning and releasing. Figures 9 and 10 show the results at the beginning and at the end of the study.

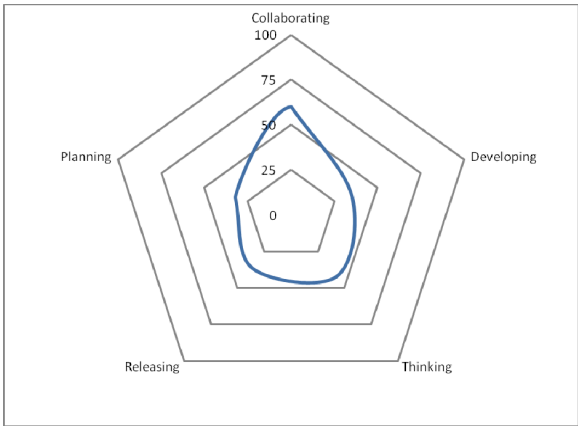

Figure 9. Initial assessment of the case study in a real life cycle 2

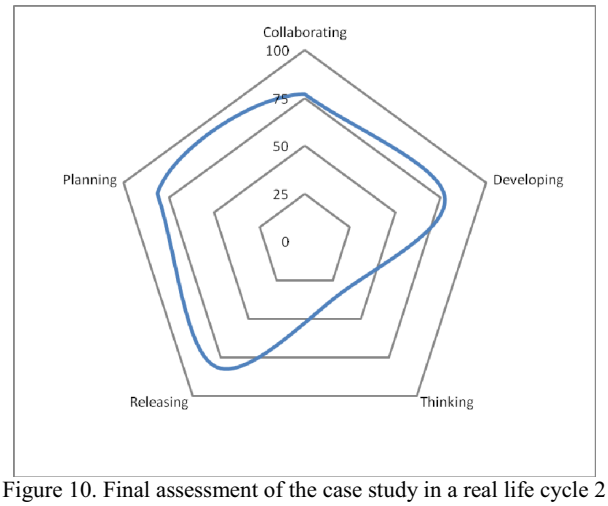

Figure 10. Final assessment of the case study in a real life cycle 2

The results indicates a evolution in all the dimensions, except in thinking. One of the reasons was that the team had challenges throughout the course with Test-Driven Development (TDD), and at the end they themselves evaluated that the thinking in TDD could be improved. Table 5 also summarizes the technical results of the team.

As in case study 1, we did not measure functional tests and unit tests because we have many front-end stories. Due to technology limitations, we collected code coverage only in iteration 1 and in the last iteration, but the results indicate significant improvement. In this study, all stories had to be done using pair programming, so we did not measure the % stories done in pairs. The happiness index presented a behavior similar to that of the previous studies and had the highest score in two iterations. We did not use NPS in this case study due the customer´s choice.

Table 5. Metrics collected during the real lifecycle case study 2

	It 1	**It 2**	**It 3**	**It 4**	**It 5**
Velocity	0	0	10	5	12
Code coverage	57.7%	-	-	-	79.9%
Commits/week	0	2	15	5	30
Status of continuous integration	38%	63%	63%	63%	63%
Hapiness index	4.8	5	4.5	5	3.8

The product owner in this study was also a business analyst from the company. In the coordination of the training program there were two professors and a coach. We also have two mentors and two teaching assistants who often supported in technical project aspects.

At the end of the course, we also collected qualitative feedback from the students. One of them said:

"I learned to understand the costumer value and the importance of a strong relationship with the Product Owner role."

Another student reported:

"I learned that communication is a key factor in software development, both among the team and with the PO. I believe that agile methodologies support not only the technical part, but also the interaction between all parties involved."

VI. Discussion and Lessons Learned

During the four studies we observed a significant growth of the students in aspects such as technique (an improvement in skills such as code coverage and continuous integration), governance (most of the students learned how to work in a team and to self-organize), business (the contact of a real project and a continuous interaction with a product owner) and behavioral (the students experienced how to collaborate better within a team using practices such as pair programming and coding dojo). In the observational studies we had possitive results in terms of team velocity. While in the first observational study the velocity increased 233%, in the second study the velocity increased 200%. In the first case

study in a real life cycle we observed that the velocity was impacted because the environment was less controlled and the project suffered from complex external dependencies. Based on these results, we concluded that for a real life cycle we have to be careful with the external dependencies in the project.

At the end, all teams were more focused and more organized. From the point of view of training, it could be argued that after the course students were able to form a team to manage their own work focusing on software product delivery. But more important than delivery capacity is the ability to gradually improve their performance, adopting the correct posture when working in a software development team. Future improvements of Software Kaizen will include a measure of retention. In other words, we will propose a way to measure how the participants perform later on, whether they still work as agile teams or if they reverted to how they worked previously, or if they were able to maintain constant velocity after the study period.

Another important aspect to report was the different roles adopted by the students during the course, which offered them the opportunity to understand, in a practical way, which role involves the effects of the failures and success in the performance of the team. The students were able to exchange ideas and experiences with professionals (mentors).

As it was a short course, the students felt free to participate in different ways, including open communication with participants from ThoughtWorks. We made it clear to them that the purpose of Software Kaizen was not to give them a grade, but immerse them in a real project context.

Communication was the most important aspect in the project. We understand that there are different perspectives and goals from the different partners (university, company, students and teaching assistants). For this reason, it is important to balance these goals and align a common vision for the project. This was a challenge, but regular communication helped us to resolve the differences.

We also identified several research opportunities in order to evaluate the effectiveness of Software Kaizen in an industry setting. While in the first four instances we applied the method having the university as the main customer, we understand that the method can be applied with real customers and with industry professionals. This will be explored in the fifth edition.

During the evaluation of the Software Kaizen method, we also learned important lessons that helped us to improve the process. Altogether we identified eight lessons that are listed as following:

Lesson 1: Pair programming from the first moment

During the first two studies, we observed that pair programming was not well adopted by the students; they tended to work alone. Pair programming is an agile practice important for learning and collaboration in our experience, as reported by several studies in the literature [19, 20]. To get the benefits of pair programming into the Software Kaizen method, we made pair programming part of the process. Therefore, from our case study in industry, during the entire process the students used pair programming. The pomodoro techinique [24] also supported PP in a beneficial manner to maintain the roles (driver and observer) with the students.

Lesson 2: Students have the flexibility to set up the environment

In the observational studies, the teaching assistant was responsible for preparing the environment. During the course, we observed that all the knowledge related to continuous integration and configuration aspects was centered on the teaching assistants and not on the students. Thus, beginning in the second observational study, the students engaged in the setup stage, and this helped them throughout the course.

Lesson 3: Retrospective sessions help the coach give the directions in each iteration

Retrospective sessions helped the coach obtain feedback about the training program and change the directions when needed. Moreover, at the end of each of the four editions a final retrospective session with the students was planned and executed. This retrospective session helped to improve the Software Kaizen method. Figure 11 and figure 2 show the team during the retrospective session with the coach.

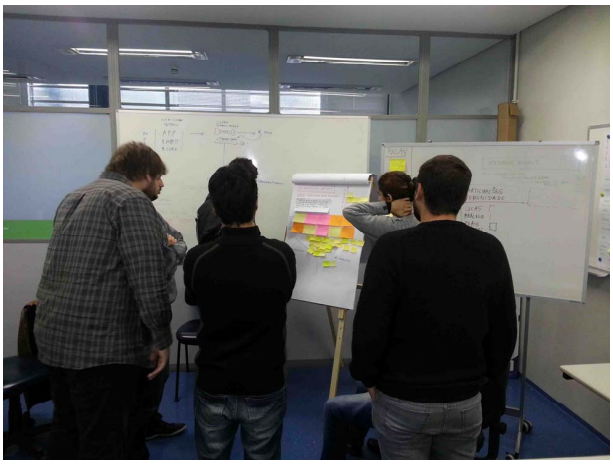

Figure 11. A retrospective performed by the coach (case study 1)

Figure 12. The final retrospective in case study 2

Lesson 4: It is important to make clear that the main goal of Software Kaizen is the learning outcomes

One of the challenges that we faced in the first case study in a real life cycle was the fact that the students thought that the Software Kaizen method was a hiring process for the company. This fact created competition between the team members and several relationship problems. Once we identified this challenge, we made it clear to the students that the main purpose was practical learning.

Lesson 5: An impediment board helps the teaching assistants to solve the impediments of the team

Beyond the daily meetings with the team, we decided to create an impediment board to help the teaching assistants to solve the impediments of the team. This board was continuously updated by both the team and teaching assistants when the impediment was solved. This strategy helped with the velocity of the team and the teaching assistants' work.

Lesson 6: The Software Kaizen method needs a specific infrastructure

Since the first instances, we prepared a specific room to support the Software Kaizen method. This room has boards, a flip chart, and workstations with two teaching assistants and two keyboards to allow for pair programming. In the last two instances we limited the number of workstations available to exactly fit the number of pairs. The reason was that in the first two instances we had more workstations available, and the students tended to do solo programming. Figure 13 shows a view of the Software Kaizen room.

Figure 13. Software Kaizen room

Lesson 7: Adopting coding dojo sessions in the recruitment process is a good practice

Coding dojo session [14] was an important step in the recruitment process to evaluate teamwork among all the candidates. Besides technical knowledge, coding dojo session presented an overview of how the candidates dealt with decisions.

Lesson 8: Coding dojo sessions help training with technical aspects

An approach that we used to solve technical impediments was the coding dojo session [14]. In this approach we also involved the teaching assistants and mentors in pairing with the students. We collected good results in relation to learning with dojo and the creation of a collaborative environment.

VII. CONCLUSION

In this paper we presented Software Kaizen, a training method which provides temporary immersion of a team in a high-performance environment, based on agile methodologies. An early version of the Software Kaizen method, with the results from the first observational study, was documented in a paper that was published, in Portuguese, and presented at the V Education Forum on Software Engineering (FEES), as part of the XXVI Brazilian Symposium on Software Engineering (SBES), in 2012 [23]. In the same year this early version was also presented at the Agile Brazil Conference, also in Portuguese, and included partial results from the second observational study. This presentation was recorded and was made available at the InfoQ Brazil [24].

In this paper we have described the current version of the methods, the concepts and the training program in details, including the results obtained in four consecutive instances (four months each) where the Software Kaizen was applied with groups of students and professionals in 2012 (in collaboration with DBServer) and 2013 (in collaboration with ThoughtWorks).

In the first two instances, for example, the team velocity improved by 230% and 200% respectively. Overall, the four instances presented above average results in terms of agile practice adoption, and team behavior. We also share lessons learned, presenting the main benefits and challenges identified. In future research, we plan to continue following the Shull methodology [12], executing and improving the Software Kaizen in order to apply it in industry.

ACKNOWLEDGMENTS

In 2013 the Software Kaizen method received an award for innovation in education, offered by the Union of Private Education of Rio Grande do Sul (SINEPE-RS). For the 2013 award, a total of 110 projects were evaluated in all categories, and Software Kaizen was the only IT related finalist project. We are thankful to all the students, professionals and mentors who have contributed to this project, which is partially funded by the research agreement signed between ThoughtWorks and PUCRS. We also thank DBServer, and CNPq (under projects 560037/2010-4, 550130/2011-0, and 309000/2012-2).

REFERENCES

[1] C. G. Von Wangenheim, , R. Savi, and A. F. Borgatto. 2013. "SCRUMIA - an Educational Game for Teaching SCRUM in Computing Courses," Journal of Systems and Software, vol. 86, pp. 2675-2687.

[2] Nearshore Americas, "Collaborate. Innovate. Accelerate. Creating successful software requires a new model of development and a new kind of development team". http://www.nxtbook.com/nxtbooks/nextcoast/nearshore_americas/#/1. [Online].

[3] Roda, R., "Self-Organizing Agile Teams: A Grounded Theory", PhD Thesis, Victoria University of Wellington, 2011.

[4] S. K. Parker, and P. R. Jackson, " The implementation of high performance work teams, Case Studies in Organisational Behavior and Human Resource Management, 2nd Edition (pp 42-56). London: Paul Chapman Publishing, 1993.

[5] K. Beck, et al., 2013, "Manifesto for Agile Software Development," [Online]. www.agilemanifesto.org.

[6] V. Oza, P. Kettunen, P. Abrahamsson, and J. Münch, "Attaining High-performing Software Teams with Agile and Lean Practices: An Empirical Case Study," Proceedings of the International Software Technology Exchange Workshop, 2011, Stockholm, Sweden.

[7] ACM, "Computer Science Curriculum," 2013. [Online]. http://www.acm.org/education/curricula-recommendations.

[8] V. Devedzic, "Teaching agile software development: a case study, IEEE Transactions on Education," vol. 54 (2), 2011, pp. 273–278.".

[9] C. G. VonWangenheim, F. Shull, " To Game or Not to Game?," IEEE Software, vol. 26, 2009, pp. 92-94.

[10] A.Goold, and P. Horan, "Foundation software engineering practices for capstone projects and beyond," Proc. 15th Conference on Software Engineering Education and Training, pp 140-146, 2002.

[11] J. Sutherland, S. Downey, and B. Granvik,. "Shock Therapy a Bootstrap for Hyper-Productive Scrum," Agile Conference, Experiente Report, 2009.

[12] F. Shull, J. Carver, G. H. Travassos, "An empirical methodology for introducing software processes," ACM SIGSOFT Software Engineering Notes, v. 26, n. 5, pp. 288-296, 2001.

[13] J. K. Liker, and M. Hoseus, .Toyota Culture: The Heart and Soul of the Toyota Way. ," Ed. Bookman, 2009.

[14] M. Bravo. and A. Goldman, "Reinforcing the Learning of Agile Practices using Coding Dojos," Lecture Notes in Business Information Processing, vol. 48, 2010, pp 379-380.

[15] ThoughtWorks Agile Assessment, 2012. [Online]. www.agileassessments.com/.

[16] J. Sutherland, "Happiness metric wave of future". http://scrum.jeffsutherland.com/2010/11/happiness-metric-wave-of-future.html, [Online], 2010.

[17] B. Tuckman, "Developmental sequence in small groups," Psychological Bulletin, vol. 63, pp. 384–99.

[18] Frederick, R, "One Number You Need to Grow," Harvard Business Review, 2003.

[19] J. Shore. The art of Agile Development. O'Reilly Media, 1st Edition, 2007.

[20] C. Mcdowell, L. Werner, H. Bulock. and J. Fernald,"The effects of pair-programming on performance in an introductory programming course, " SIGCSE Symposium on Computer Science Education, 2002, pp. 38–42.

[21] N. Salleh, E. Mendes, J. Grundy, "Empirical studies of pair programming for CS/SE teaching in higher education: A systematic literature review," IEEE Transactions on Software Engineering, vol.37-4, Jul-Ago.. 509–525.

[22] S. Nöteberg, Pomodoro Technique Illustrated: The Easy Way to Do More in Less Time. Pragmatic Programmers, Raleigh, N.C, 2009.

[23] R. Prikladnicki, M. Móra, B. Copstein, A. Olchik, M. Bastos, L. C. Parzianello, "Kaizen: Training High Performance Software Development Teams," V Education Forum in Software Engineering (FEES), as part of the XXVI Brazilian Symposium in Software Engineering (SBES) , in Portuguese, 2012.

[24] A. Olchik, R. Prikladnicki, "Software Kaizen: an innovative model for high performance teams," Agile Brazil Conference (www.agilebrazil.com), also available online at www.infoq.com/br/presentations/software-kaizen-equipes, 2012.