Software Engineering Using Experimental Learning

Asbjørn S. Danielsen

Abstract—**This paper reports on an experimental approach on teaching Software Engineering (SE) and the results recorded during two years at Narvik University College. The course described in this paper uses experimental pedagogy and problem-based learning to give the students experience in close to real-life work environment, demonstrating social and problem complexity of Requirements Engineering (RE) and Software Development (SD). The course uses social simulations rather than software simulations, making the students learn through interactions with real people and confronted with the complexity of social relationships.**

Index Terms—**Requirements engineering, software engineering, experimental learning, problembased learning.**

I. Introduction

Traditional lecturing in SE tends to focus on the theoretical aspects of SE, the different processes involved in SE, and the different properties of these processes. “When students try to understand a problem in only one way, especially when that way conveys no conceptual information about the problem, they do not understand the underlying systems in which they are working” [1] p. 364. In traditional teaching methods that use a linear presentation of materials (e.g. textbooks, lectures), students gain knowledge at the most basic level and memorize scientific facts without understanding the underlying concepts [2]. As a result, misconceptions about these concepts can develop. Misconceptions can be strongly held ideas, and these ideas are often difficult to change with traditional instructional methods.

Avoiding such misconceptions is an important endeavor, both within RE and SD. It is important that RE practices and RE education is emphasized because the industry has a relatively poor understanding of RE practices and their benefits. In an ideal world, a course in RE should be provided at university level, before the students become software developers. This is however not so in most cases [3]. When they do, these courses are often given in the traditional lecture/exercise format, and only few reports exist on other types of pedagogy used in such courses, i.e. [4], [5], and [6].

It is a common understanding that requirements represent the expression of peoples desires [7]. To understand the desires of people and to be able to express these desires is essentially a social construction. Consequently, much social wisdom is packed into RE methods. Further on, it is unrealistic to expect students with little organizational experience to understand this body of knowledge and to appreciate even the need for RE methods, much less to be able to use them. In the software engineering discipline, it is essential that software engineering students understand the latest accepted methods and practices in use today in the design of complex computing systems.

If we want to provide more than just a shallow understanding of RE and SD to the students, we need to provide more than just lectures about RE and SD methods and academic problems. We need to create a platform in which they can exercise the knowledge they have been provided in our traditional teaching environment. The experiences we seek to impart to students are directly linked to the issues found in the workplace when we understand what the business is about and what the desires of people are. A short list of these issues is: dealing with ambiguity, uncertainty, confusion, fear, time pressure, collaboration, corporate politics etc. In sum, what some call the “messy” part of organizations [8]. The messy part, recognized by scientists and mathematicians as wicked problems, exemplify the differences between classroom and workplace problems.

To be able to handle the messy part, one needs to combine both emotions and techniques in one combined effort. Whereas the use of specific techniques and algorithms can be learned through lectures and exercises, emotions can only be learned through real life experiences.

This paper describes the requirements engineering and software development parts of a Software Engineering course at Narvik University College. In this paper the reasons for creating the course, its pedagogical features, and the experiences giving the course will be presented. The course is a 10 point course, spanning from spring to autumn, and is now running in its second year. This paper mainly describes the experiences during 2009 and 2010.

The course was a result of a major restructuring of the computer science education program at Narvik University College. It was designed to create a realistic organization in order to provide the students with an opportunity to experience the “messiness” they can expect in the workplace. Situations which one may be faced with in a real workplace was created, and framed the problems presented to the students in an uncertain and confusing reality, often relying only on verbal, word-of-mouth communication as this is an important part of design and management information transfer in reality.

The system the students were working on was a stock-exchange game using close to real time data from the Oslo Stock Exchange combined with locally defined stocks. The approach was to immerse the students into a more realistic social environment with real data instead of simulated data, real events instead of simulated ones,

Manuscript received August 31, 2011; revised September 20, 2011.

Asbjørn S. Danielsen is with Faculty of Technology, Narvik University College, Norway (email-asd@hin.no).

TABLE I: DIFFERENCES BETWEEN CLASSROOM AND WORKSPACE EXPERIENCES.

Experience	Classroom	Workplace
Problem definition	Well defined	Ill-defined
Problem approach	Strongly indicated by most recently presented classroom material. Problems tend to be carefully designed to reinforce specific methodologies.	Few hints as to how to approach the problem. May be required to invent new methods.
Problem solution	Professor always knows the solution.	A solution to the problem will only be apparent when accepted by management.
Problem scope	Many problems are "scoped" so that they can be solved by one person (student) in a few days or weeks.	The scope of the problem is unclear, and in general, problems require a team of several people working over a period of many months.
Social environment	Working as an individual with implied competition.	Working as a team member, cooperation being essential to success.
Information levels	Accurate, well defined, explicitly stated.	Vague, unrecognizably ambiguous. Occasional hidden agendas.
Solution methods	Given by an authority figure, usually to reinforce material recently presented.	Veracity and efficacy never an issue. May have to invent a new method as part of the problem solving process.
Design team	Same group of members from beginning to end of project.	New members join the team and old, experienced members leave the team.
Stability of problem statement	Once stated, the problem statement is rarely, if ever changed.	The problem statement changes frequently as new information becomes available.
Information channels	Heavy use of well-documented, written form.	Some documentation exists, but most information is found during informal conversations.

interviewing real people, using real planning and reporting tools.

In the next section of this paper a number of typical education related challenges will be presented. Further on, the basics of experimental learning, problem-based learning, and how these theories were adopted into the course is presented. Finally experiences lecturing the course are presented, including student's evaluation, before presenting related work and summarizing the contribution to the area of expertise.

II. CHALLENGES

Traditional education tends to be separated into separate disciplines or courses. Few, if any, courses seek to integrate disciplines. The result is that students going through the educational system acquire much factual knowledge about specific areas but little synthesis is provided. This pattern is also present in courses where each course is divided into topics which is lectured in a sequential manner, one by one, with the implication that there is little, if any, relationship between topics, and without any or just a few references to other disciplines.

Another challenge is related to wicked problems and the fact that these problems tend to be difficult to define, and the definition, if it exists, tend to change over time. Their solution is not known at the beginning and whether they were correctly solved will not be known often until long after a solution is proposed or implemented. Solving one problem often brings about a number of other problems that could not be foreseen before the solution was implemented, and so on. Wicked problems have the following properties [9]:

1) cannot be easily defined so that all stakeholders agree on the problem to solve
2) have no clear stopping rules
3) have better or worse solutions, not right and wrong ones
4) have no objective measure of success
5) require iteration – every trial counts
6) have no given alternative solutions
7) require complex judgments about the level of abstraction at which to define the problem
8) often have strong moral, political or professional dimensions which cannot be easily formalized.

The differences between the classroom experience and the real time workspace are striking. [10] presents a number of differences between the classroom and the real-life workplace experience applicable to SE courses, where the most common differences are summarized in TABLE I.

In addition to the problems in TABLE I, three additional syndromes tend to occur. The first of these are the "free-rider syndrome" [11]; when a group is graded as a whole, not all students contribute equally and work equally hard.

The second is the "Wyatt Earp syndrome" [12], which make collaboration more difficult and occur when a student presents a new accomplishment or idea to the lecturer without having first discussed it with the other members of the group.

Finally we have the all too familiar "student syndrome" [11], i.e. many students does not engage themselves fully in an assignment until the last possible moment before deadline.

III. PROBLEM BASED LEARNING

Problem-based learning (PBL) is generally based on ideas from the early 20th century, and then nurtured by different researchers such as Dewey [13], Piaget [14], and many others. PBL, as it is known today, originated in the 1950s and 1960s, and grew from frustrations and disagreements related to the common medical education practices in Canada [15]. Today PBL has developed and been implemented in a wide range of domains. In spite of the many variations of PBL that have evolved, a basic definition is needed to which other educational methods can be compared. Six core characteristics of PBL are distinguished in the core model described by [15]. These are:

1) Learning is student-centered.
2) Learning should occur in small student groups under the guidance of a tutor.
3) The tutor role is as a facilitator or guide.
4) Authentic problems are primarily encountered in the

learning sequence, before any preparation or study has occurred.

5) The problems encountered are used as a tool to achieve the required knowledge and the problem-solving skills necessary to eventually solve the problem.
6) New information needs to be acquired through self-directed learning. It is further generally recognized that a seventh characteristic should be added:
7) Essential for PBL is that students learn by analyzing and solving representative problems.

This definition makes PBL well suited to handle ill-structured problems such as the problems which may be found in a real-world related to software engineering. Further on, PBL are related to experimental learning, and offers ideas on how experimental learning may be conducted.

IV. Experimental Learning

The theory of experimental learning is generally attributed to Kolb [6]. Kolb developed this theory based on the previous pioneering work of educators such as Dewey [13], Piaget [14], Lewin, and Freire. He proposed a model of the underlying structures of the learning process based on research in psychology, philosophy, and physiology, and bases its typology of individual learning styles and corresponding structures of knowledge in different academic disciplines and careers on this structural model.

His model of experimental learning is divided into four modes, usually assembled into the experimental learning cycle presented in Fig. 1.

According to Kolb, the learners "must be able to involve themselves fully, openly, and without bias in new experiences (CE). They must be able to reflect on and observe their experiences from many perspectives (RO). They must be able to create concepts that integrate their observations into logically sound theories (AC), and they must be able to use these theories to make decisions and solve problems (AE)."

Kolb notes that this cycle is an ideal that is difficult to achieve because learners cannot easily reconcile these modes, which require different ways of interacting with one's environment and thinking about it. Further on, in the model, the modes are "dialectically" opposed along two dimensions. The first dimension, pretension, opposes Concrete Experience of events (apprehension) and Abstract Conceptualization that seeks to make generalizations of these events (comprehension). The second dimension, transformation, opposes Reflective Observation about experience (intension) and Active Experimentation that seeks to make decisions about future experience (extension). For Kolb, the level of learning is determined by the way the learner can resolve the conflicts present in these two dimensions.

Another important aspect of experimental learning has been described by Lev Vygotsky as the zone of proximal development (ZPD). ZPD is the difference between what a learner can do without help and what he or she can do with help, and is defined as "the distance between the actual developmental level as determined by independent problem solving and the level of potential development as determined through problem solving under adult guidance, or in collaboration with more capable peers" [16].

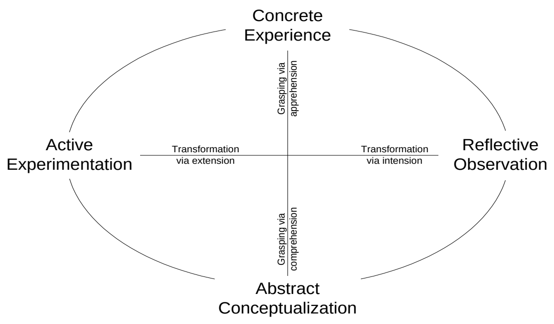
Concrete Experience
Grasping via apprehension
Active Experimentation
Transformation via extension
Transformation via intension
Reflective Observation
Grasping via comprehension
Abstract Conceptualization

Fig. 1. Kolbs model of the experimental learning cycle [6].

As already described in the previous sections, RE and SD consists of a number of wicked problems that need an iterative process to be gradually refined. During the process, the students should become more familiar with the problem domain in question, business rules and hopefully the students should be able to comprehend the nature of RE, SD and business problems through their apprehension of the concrete classroom experience.

Instructional scaffolding is based on Vygotsky's theory where he defined scaffolding instruction as the role of teachers and others in supporting the learner's development and providing support structures to get to that next stage or level. An important aspect of scaffolding instruction is that the scaffolds are temporary. As the learner's abilities increase the scaffolding provided by the more knowledgeable is progressively withdrawn. Finally the learner is able to complete the task or master the concepts independently [17]. Therefore the goal of the educator when using the scaffolding teaching strategy is for the students to become independent and self-regulating learners and problem solvers. As the learner's knowledge and learning competency increases, the educator gradually reduces the supports provided until the students complete the tasks without interference from the teacher.

V. Scrum

SCRUM is not a methodology. SCRUM is an iterative, incremental framework for project management [18], often used in agile software development.

SCRUM organizes the development process into a predefined framework of a plan, a SPRINT, which should be completed within a relatively short timeframe, usually 2–4 weeks. The SPRINT involves a number of meetings with predefined purposes, such as the SPRINT Planning meeting, the SCRUM meetings, and SPRINT review and the SPRINT retrospective meetings. During the execution of the plan a number of artifacts are produced and updated. These include a product backlog (what should be built, ordered by importance), a SPRINT backlog (list of work the team must address during the SPRINT – not assigned to persons), and a burn-down chart (updated every day and answers what remains of the SPRINT). In addition to these artifacts, SCRUM preaches a set of SCRUM core values of importance to the development process; commitment, focus, openness, respect and courage [18], pp. 147-154.

The presented properties of SCRUM makes it well adapted to be used in a Software Engineering course like the one presented in this paper. SCRUM is agile and adaptable to new requirements. It is a very good platform in which to deploy experimental learning, problem-based learning, and scaffolding strategies.

VI. The Course

The Software Engineering course was designed following the experimental learning cycle, incorporating the idea of instructional scaffolding. The course grade was defined to be Passed or Not Passed. First the class was divided into two distinct categories; those with no prior knowledge of software development, and those with some experience in OO or programming. This separation was done because it was expected that the student with no prior experience would need more assistance than the other group.

The typical course session during spring-term consisted of two times two periods of forty-five minutes every week. In the beginning of the term, the course focused on initial analysis and Business Process Modeling Notation (BPMN). Then we started working with RE phase of the course. RE is at the interface between the business understanding and the product or service to be created, and the students need to understand the business function before they can start on RE process of conducting interviews, drafting requirements, etc.

A typical lesson in the RE phase would include a presentation of a number of real-life examples, e.g. business game or a RE interview, followed by a discussion on different social and technical issues of the examples. The lesson would then be followed by a shuffle of people within groups into teams of 3 – 7 people. The shuffling of people within the group was done to provide a kind of uncertainty related to people attending the group and to be able to reduce the predictability of the working environment for the students. Each team was given identical exercises in form of similar activities as those being presented during lecture. The exercise was usually in the form of a RE interview related to a limited number of problems within a system. Most experience sessions where followed by a 45 minute reflective observation phase in which emotional and technical debriefing was performed. After the debriefing was performed, the groups exchanged documents developed during the process and commented on the other groups work. To finish the session off, a lesson to sum up the experiences with the exercise was held. The typical week doing RE-lessons looked as described in TABLE II.
P12e450

TABLE II: Spring Lecture Session Sequence.

Session	Seq.	Subject
Session 1	4	Summary lecture and discussions
	1	Introducing a RE subject, exemplifying with real-life examples
Session 2	2	Shuffling group-members into work-groups Perform exercise, create RE descriptions
	3	Reflective Observation Phase Emotional and technical debriefing Change of RE work between groups

After having lectured after this model for five weeks, a new shuffle of the members of the groups where performed creating a project group which should be stable for the rest of the course. Project management techniques were lectured and the class was at a stage in which they could embark on a journey into doing a larger RE exercise resulting in the requirements for a stock-exchange game, and using instructional scaffolding in the process. Each of the teams was organized as different companies, competing in the same market, developing competing systems.

The system to be developed was described as an internet-based game which should be as close to real-life experience as the real stock exchange. The game period should be limited to a number of months after startup. All buying and selling within the game should be done using virtual money, and should include actual stocks on Oslo Stock Exchange (OSE), real-time stock rates and availability. The game should also support virtual companies and their stocks which also should be tradable. All selling and buying should be done using a broker. Further on the system should incorporate some kind of security to minimize the problem that a single user may hold several identities within the game, making it possible to do illegal trades.

All groups got the same information. Within the first two weeks, the students had to deliver a business game document on how stock trading is performed in Norway, different techniques of stock trading, which stock types exist and what is the difference, what laws apply, and to some extent give an explanation on a number of financial and economical terms used in stock trading. They also had to deliver the first project plan, and their first project description.

As a result of severe time pressure, each group struggled to understand the problem domain, describe and document it, and create a project plan. It could be observed that in most cases the students neglected the text given in the exercise, and that there was no overall picture on what was going on. Frustration and emotions surfaced quite often, and some students even expressed doubts about the competence of the lecturer. The planning aspect was put out until last minute and some groups didn't even manage to come up with a plan. After this exercise, an emotional and a technical debriefing were performed, evaluating the results. Then all groups exchanged business game documents, and commented on differences. This period was then followed by a 7 week module focusing on RE and specifications. At the start of this period the students got a document from the company's sales- and marketing departments about sales and marketing problems, and the teams was instructed to deliver a simplified Software Requirement Specification (SRS) using BPMN as modeling notation where applicable. They were also instructed that further information was available from the CEO (lecturer), CIO (student assistant #1), customer (student assistant #2) as well as one outside stakeholder (lecturer imposing as an advertising interest). A document was prepared for each role in the game, ensuring that the same main messages and information were delivered to all groups during interview. During the whole process, except during interviews, the lecturer performed instructional scaffolding as described.

After the first round of interviews, each group wrote its own simplified version of the SRS and presented it to the CEO of the company. The CEO did not approve the SRS, and the students was instructed to do further investigations, especially into the feasibility and validity part of their SRS. The students concluded they needed more interviews and a second interviewing session was set up.

At the completion of the 7 week module, a new and updated version of the SRS where delivered to the CEO, which this time accepted and approved it. This document was later used to design and implement a prototype of the system.

When autumn started, the student had lectures related to project management, project reporting and project execution in general, and SCRUM in particular. Further on lecturing focused on the different phases of software development and the relevant contexts in each phase. Finally different agile software development methodologies was lectured, such as extreme programming (XP), test-driven development (TDD), lean software development (LD) and rapid application development (RAD). All lecturing was completed within the first 3 weeks of the autumn semester period (3 lectures per week).

At this point the student groups used during the RE-phase was revived, and each group was instructed to develop a first version of the stock-trading game based on their own SRS-document, using SCRUM and one or more agile software methodologies (XP, TDD, LD, or RAD). Since SCRUM suggests the development group to be self-organizing and self-led, they had to decide the length of each SPRINT (a minimum of 2 SPRINTs had to be completed), and how the project group would organize and decide on who should be the SCRUM Master. Final delivery date of software was set 9 weeks into the future, and the team where instructed that the lecturer would pose as Product Owner as well as doing instructional scaffolding. Further on, the project teams was informed that student assistants would take on different roles as "chicken" and try to influence the project in different ways.

The class was now at a stage in which they could embark on a journey into doing a larger and more complex SD exercise resulting in a first version of a stock-exchange game.

The first assignment during the autumn-term was to within one week of startup deliver a project organization overview, a project plan on how to execute the project – identifying each SPRINT and delivery point, decide which agile software development methodology (XP or some other) to use and explain why, give a plan of the first SPRINT, decide when to have the "daily SCRUM meeting", create the product backlog, do the first SPRINT Planning meeting, and make a

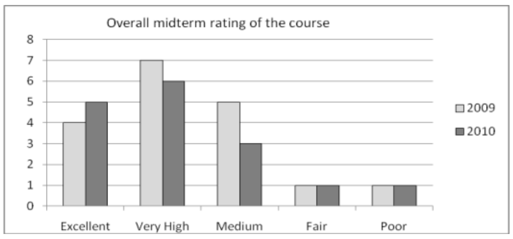

Fig. 2. Overall midterm rating of the course.

SPRINT Backlog. Lecturer was present at all meetings, changing roles between "Product Owner" and lecturer doing instructional scaffolding, guiding the students through the experience.

As a result of severe time pressure, each group struggled, once more, to understand the problem domain and to perform the tasks that had been defined as the first assignment. The next assignments were tailored to fit each project group, based on how many SPRINTs the group would perform.

During the execution of the development project, debriefing sessions were held. They were partly related to each groups SPRINT retrospective meetings (time for after thoughts on what went well and what could be improved), and partly in the form of separate sessions in which the groups exchanged experiences and discussed the situations. Further on, lecturer and student assistant attended most SCRUM meetings, and some programming sessions.

VII. Experiences

The course was given two spring-terms with 15 and 18 students in each year. The course itself was mandatorily for all students in the Bachelor of Engineering study of Computer Science at Narvik University College, and the course has been well evaluated during mid-term and end-term in both years.

A. Course rating

In both years the course has been evaluated by the students. The overall midterm evaluation performed at end of the RE-part of the course, is presented in Fig. 2. Fig. 3 shows the overall students evaluation at the end of the course, while Fig. 4 gives a student subjective overview on how much time the students has invested in the course.

B. Learning Style

The course is problem-based and the approach towards learning is completely different from other courses at campus. This had to be explained frequently during the course. At the beginning of the course several students found the course to be disorganized and found it difficult to accept this kind of pedagogy. However, at the end of the course most students found the problem-based approach to be fruitful. No students quit the course.

C. Syndromes

The "free-rider" and the "student" syndrome could be observed to some extent in all groups. The "free-rider" syndrome is indicated by Fig. 4 by those students that did not invest much time in the course. The "student" syndrome was observable as well since the project burndown increased drastically when the SPRINT deadline came closer.

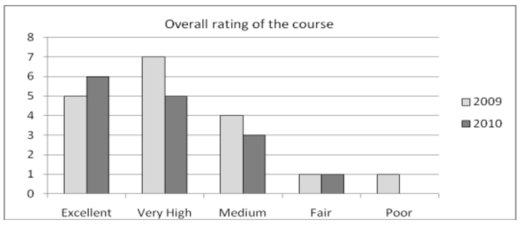

Fig. 3. Overall (endterm) rating of the course.

Minimizing the occurrences of the "free-rider" syndrome may be done by defining a web based scoreboard where all participants in the group could see the scores of everyone else in the group. Each participant earns 5 points for every meeting and programming sessions he or she attended. In addition, the student earns 10 points for every finished backlog item, and -10 points for not being present at meetings or programming sessions. At the end of a SPRINT or the course, it will be evident who has attended and contributed and who has not.

D. Group Dynamics

Group dynamics was challenging. Even though the work was done in groups, the most common scenario was that two or three persons in the group executed the work while the rest observed. It was also evident that when all members of the group worked on separate parts of the project, they did not communicate well with the other parts of their group working on related material. This resulted in incompatible models and specifications. The latter situation occurred more often during time pressure and was more evident earlier in the course during RE.

The SCRUM meetings, performed 2 times a week in the autumn-term, made the group communicate better. They found the experience embarrassing at first, but after 3-4 meetings they learned to appreciate the information and the openness expressed during the meetings.

Most groups used XP and pair-programming during the start of the project, but tended to leave this strategy when the end of the SPRINT got closer, partly due to the time boxing SCRUM preaches, partly due to the overhead pair-programming result in, and the “student syndrome”.

Frequent reporting on time used and progress had the effect that all participants in the group became more conscious of what they were doing, how much time they had used and how unsecure they were about how much time had to be used before they could complete the task at hand. This situation also made the students more aware of how much the other group members had left of their work.

Each student would get a grade in the course as Passed or Not Passed, dependent on the work of the group, and how much the student had participated in the groups work. Some students found this to be unfair due to the fact that some individuals contributed more than others (some degree of “free-rider” syndrome). This challenge may be solved by changing the grade to an A-F scale, making the completion of the project mandatorily and having a final oral exam. Further on, the use of a web based scoreboard may help document the degree of contribution by each member of a team.

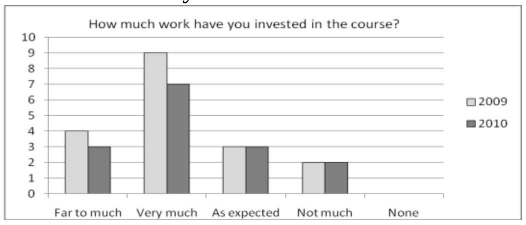

Fig. 4. Subjective student workload in course.

E. Credibility

It is of utmost importance that a course like this which differs drastically from other university courses has credibility. As the course puts the student in a stressful situation, the student might infer that the course is disorganized. With adequate communication, the students eventually believe that the course puts them in a situation related to the “real life” situation and learn from their experience.

The following strategy was used to raise the credibility of the course:

1) Used own industrial experience as examples
2) Used real-life project examples to demonstrate problems
3) Open discussions
4) Open scenarios based on student experiences

It is preferable to have guest speakers from the software industry to do some talks on different aspects of software development. It would also be preferable that more lecturers shared a course like this, partly because of workload, but also by increasing the industrial experience and by so broadening the horizon of the students.

F. SCRUM Characteristics

The SPRINT definition was a problem during execution of the project. The problem was twofold. Partly due the idea that SCRUM presumes that system development is the single process each SCRUM team member is involved in, and partly due to the presumption that all members of the SCRUM team has identical time schedules. This was not the case since the course is a 10 point course spanning two semesters, from spring to autumn and only being a small part of a student’s schedule. Further on, differences in student time schedule were present.

The SCRUM meetings should be held daily, but because of student schedules, the meeting was performed twice a week. Programming sessions was usually 2 – 4 times a week. The effect was that the progress rhythm in the SPRINT varied.

G. Student Workload

Several students claimed that the course was time-consuming related to other courses on site (Fig. 4). Every student maintained a timesheet summing up hours used in the project, defining which activity in the project the student was involved in, and how much time the student had spent.

In average, each student put 312 hours into the RE- and SD-project in addition to lectures. The workload on each student is larger than what should be expected in a 10-point course. To adapt the workload to the course, an increase in study points to 15 may be applicable.

H. Instructional Scaffolding

The student groups tended to delve into problems with wrong focus, e.g. during the RE-phase they tended to focus on descriptions on how some problem should be solved on an object-level rather than specifying the properties of the problem to be solved in relation to circumstances, consequences, process, and so forth. Lecturer repeatedly had to re-establish RE goals in the group, guiding the group out of its OO-sphere and back into the game.

In this context, the lecturer and assistants need to be more involved in the early stages of RE to establish focus on RE rather than OO-specific modeling. Similar observations were done during the SCRUM development phases.

VIII. Related Work

There seems to be few publications on experimental pedagogy and problem based learning based on projects and teamwork in software engineering. [19] report on student teams which perform an industrial software development project and presents the main educational problems encountered in such real-life projects. [20] reports on project based learning and how scaffolding may be used within the project management body of knowledge, while [21] focuses on a global software development project where extended teams of students distributed across two to three countries experienced the roles of developers, auditors and testers. Carnegie Mellon's West Coast Campus offers a Software Engineering course [22] which in many aspects seems to be the closest fit to the course lectured at Narvik University College.

There seems to be few publications on experimental pedagogy related to RE as well. Some publications were published at the REET 2005 workshop, e.g. [3], [4], and [5]. [4] discuss challenges encountered teaching across universities, different cultures and time zones, while [5] focus on an investigation-based approach towards RE. [10] presents an immersive and problem based approach closely related to instructional scaffolding. It further on use low-tech and social simulations rather than computer based.

IX. Conclusion

In this paper an experimental approach toward teaching software engineering has been presented with its theoretical foundation. The course was designed with two major objectives in mind; prepare the students for the real-world experience in which they will embark after a completed study, and, create an arena in which the students may place their functional knowledge obtained during their studies.

Doing a course like this require more planning and more resources than a standard course. It is also more difficult to teach since it involves wicked problems and consequences of such problems. The course uses more man-hours than other courses.

The effect of the course is evident in the student evaluations of the course, and it is possible to see the effect in B.Sc. theses of students completing the course. The B.Sc. thesis of students who has completed the course seems to have a more balanced approach towards software engineering in general. The main component contributing to this is most likely the experience each student gets by working with wicked problems and agile software development in the course discussed.

We think we have succeeded with providing a context in which the students may place their functional knowledge obtained during their studies.

References

[1] D. H. Jonassen, "Designing research-based instructions for story problems", *Educational Psychology Review*, vol. 15, no. 3, pp. 267-296, 2003.

[2] S. Cepni, E. Tas, and S. Kose, "The effects of computer-assisted material on students' cognitive levels, misconceptions and attitudes towards science", *Computers and Education*, vol. 46, pp. 192-205, 2006.

[3] B. Berenbach, "A hole in the curriculum", in *Proceedings 1st International Workshop on Requirements Engineering Education and Training (REET 2005)*, Paris, 2005.

[4] D. Damian, B. Al-Ani, D. Cubranic, and L. Robles, "Teaching Requirements Engineering in Global Software Development: A three-University Collaboration", in *Proceedings 1st International Workshop on Requirements Engineering Education and Training (REET 2005)*, Paris, 2005.

[5] N. H. Madhavji and J. Miller, "Investigation-based Requirements Engineering Education", in *Proceedings 1st International Workshop on Requirements Engineering Education and Training (REET 2005)*, Paris, 2005.

[6] G. Regev, D.C. Gause, and A. Wegmann, "Requirements Engineering Education in the 21st Century, an Experimental Learning Approach", in *Proceedings of the 2008 16th IEEE International Requirements Engineering Conference*, Washington, DC, 2008, pp. 85-94.

[7] D. Gause and G. Weinberg, *Exploring Requirements: Quality Before Design*, New York, NY, Dorset House Publishing, 1989.

[8] P. Checkland and S. Holwell, *Information, Systems and Information Systems – making sense of the field*, New York, NY, John Wiley and Sons Ltd, 1997.

[9] S. J. B Shum, A. MacLean , V.M.E. Bellotti, N.V. Hammond, "Graphical Argumentation and Design Cognition", *Human-Computer Interaction*, vol. 12, no. 3, pp. 267-300, 1997.

[10] G. Regev, D. C. Gause, and A. Wegmann, "Requirements Engineering Education in the 21st Century, an Experimental Learning Approach", in *Proceedings of the 2008 16th IEEE International Requirements Engineering Conference*, Washington, DC, 2008, pp. 85-94.

[11] E. M. Goldratt, *Critical Chain*, Great Barrington, MA, North River Press, 1997.

[12] Boehm B.W. and Port D., "Educating software engineering students to manage risk, in *Proceedings of the 23rd International Conference on Software Engineering*, Toronto, ON, 2001, pp. 591-600.

[13] J. Dewey, *How we think*, Boston, Health and Co, 1910

[14] J. Piaget, *The construction of reality in the child*, New York, NY, Basic Books, 1954

[15] H. S. Barrows . (August 1996). Problem-Based Learning in Medicine and Beyond: A Brief Overview, *New directions for teaching and learning term*. [Online]. 68, pp. 3-11, Available: http://onlinelibrary.wiley.com/doi/10.1002/tl.37219966804/pdf

[16] L. S. Vygotsky, *Mind in Society: The Development of Higher Psychological Processes*, Harvard University Press, 1978, p. 86.

[17] K. Chang, I. Chen, and Y. Sung, "The effect of concept mapping to enhance text comprehension and summarization", *The Journal of Experimental Education*, vol. 71, no. 1, pp.5-23., 2002.

[18] K. Schwaber and M. Beedle, *Agile Software Development with SCRUM*, Prentice Hall, 2001

[19] L. van der Duim, J. Andersson, and M. Sinnema, "Good Practices for Educational Software Engineering Projects", in *Proceedings of the 29th International Conference on Software Engineering(ICSE 2007)*, Washington, DC, 2007, pp. 698-707

[20] S. W. van Rooij. (January 2009). "Scaffolding project-based learning with the project management body of knowledge", *Computers and Education. [Online]. 52(1)*, pp. 210-219, Available: http://www.sciencedirect.com/science/article/pii/S0360131508001139

[21] C. Scharff, "Guiding global software development projects using Scrum and Agile with quality assurance", in *Proceedings of the 24th IEEE Conference on Software Engineering Education and Training (CSEEandT)*, Waikiki, Honolulu, 2011, pp. 274-283.

[22] R. Bareiss and M. Griss, "A story-centered, learn-by-doing approach to software engineering education", in *Proceedings of the 39th SIGCSE technical symposium on Computer science education*, Portland, OR, 2008, pp. 221 – 225.

Asbjørn S. Danielsen holds a cand.scient. degree from University of Oslo, Institute of informatics from 1999 in computer science.

He is an Associate Professor at Narvik University College where he teaches Software Engineering and Open Distributed Systems.