Informatics in Education, 2005, Vol. 4, No. 2, 167–192
© 2005 *Institute of Mathematics and Informatics, Vilnius*

Object-Oriented Software Development Education: a Constructivist Framework

Said HADJERROUIT
Agder University College, Faculty of Mathematics
Serviceboks 422, N-4604 Kristiansand, Norway
e-mail: said.hadjerrouit@hia.no

Received: June 2005

Abstract. The paper argues for the importance of the constructivist learning theory to software development education. Constructivism frames learning less as the product of passive transmission than a process of active construction whereby learners construct their own knowledge based upon prior knowledge and experience. Now that a number of software development courses offer project-based teaching, it seems that the importance of a constructivist perspective has been implicitly well-taken in the current practice. What these approaches explicitly lack is a concrete methodology of how to carry out the constructivist perspective and its consequences for learning. This paper reports on a constructivist approach to object-oriented software development at the undergraduate level. It explores methodological aspects of the approach and discusses the results from its evaluation.

Key words: constructivism, learning cycle, object-oriented software development, online resources, unified modeling language.

1. Introduction

To educate skilled software developers, teachers have tried to make software development education more realistic and project-based (Saiedian, 2002). To meet this goal, many approaches have been developed, particularly the Software Engineering Studio (Sebern, 2002), the Software Development Laboratory (Tomayko *et al.*, 2002), the Student-Enacted Simulation Approach to Software Engineering Education (Blake, 2003), and the Unified Process to Education and Training (Halling *et al.*, 2002). Now that a number of pedagogical approaches are concerned with trying to make software development education more realistic and project-based, it seems that the importance of a constructivist perspective has been implicitly well-taken in the current practice. But, these approaches do not explicitly integrate learning paradigms and associated pedagogical innovations, such as constructivism and related approaches. As a result, they do not sufficiently take into consideration the human aspect of the learning process – students' thoughts, misconceptions, behavior, difficulties – that is of particular importance for helping novice students entering the field of software development. Clearly, what current approaches to software development lack is a concrete methodology of how to carry out the constructivist perspective of learning.

The remainder of this article is organized as follows. First, the paper gives an overview of the constructivist learning theory and how to translate it into practice. Then, the paper describes a methodology of how to carry out a constructivist perspective in object-oriented software development. The next section describes the content and structure of a two-semester course in object-oriented software development at Agder University College. This is followed by the description of the online learning environment of the courses. Then, the article presents, in three sections, the results from the evaluation of the approach. Finally, some remarks on further work conclude the paper.

2. The Constructivist Learning Theory

Important to the design of a constructivist approach to software development is a pedagogical foundation built on solid learning theory and appropriate instructional strategies. The constructivist learning theory has its roots in the movement of constructivist epistemology and philosophy with three orientations: individual constructivism, radical constructivism and social constructivism. Its central figures include (Bruner, 1990; Kelly, 1995; Piaget, 1969; Von Glasersfeld, 1993; Vygotsky, 1978).

2.1. *Constructivist Principles*

Constructivism suggests a set of principles that may be used to design constructivist learning environments (Ben-Ari, 1998; Duffy *et al.*, 1993; Duit *et al.*, 2001; Gros, 2002; Honebein *et al.*, 1993; Matthews, 2002; Phye, 1997; Spivey, 1997; Steffe and Gale, 1995; Staver, 1998; Tam, 2000; Tynjaelae, 1999; Wilson, 1998; Young and Collin, 2004):

- Constructivism frames learning less as the product of passive transmission than a process of active construction whereby learners construct their own knowledge based upon prior knowledge and experience. Therefore, the constructivist model calls for learner-centered instruction, because learners are assumed to learn better when they are forced to discover things themselves rather than when they are instructed.
- The process of constructing knowledge requires meta-cognitive and higher-order thinking skills, such as analogical reasoning, reflection, and self-evaluation. Analogical reasoning is a key skill of learning processes with a constructivist perspective: every learning process includes a search for similarities between what is already known and the new, the familiar and the unfamiliar.
- Constructivist learning requires learners to demonstrate their skills by constructing their own knowledge when solving real-world problems. Rather than applying knowledge to solve abstract problems, knowledge must be constructed in real contexts. Real-world problems have enormous potential for learning, because knowledge construction is enhanced when the experience is authentic.
- In a constructivist setting, teachers serve primarily as guides and facilitators of learning, not as transmitters of knowledge. Teachers must learn how to understand

students so that they can interpret responses better and guide communication more effectively in order to facilitate learning.

- Constructivist learning will be appropriately implemented only if students are evaluated constructively. Such evaluation requires methods that are embedded in the learning process and approaches that take into consideration the learners' individual orientations.
- Learning emerges through interaction of learners with other people, e.g., instructors, fellow learners. Learning occurs as students construct, verify, test, and improve their knowledge through discussion, dialogue, collaboration, and information sharing. Thus, constructivism involves a communication process in which learners are actively and reciprocally engaged in the process of knowledge construction, creating, and sharing meaning and problem-solving situated in authentic tasks.

2.2. *The Learning Cycle*

There are many applications of constructivist principles in Computer Science and Software Development Education (Ben-Ari, 1998; Ben-David Kolikan, 2001; Booth, 2001; Fowler *et al.*, 2001; Hadjerrouit, 1998a; Hadjerrouit, 1998b; Hadjerrouit, 1999; Mereno-Seco and Fordaca, 1996; Pullen, 2001; Soendergaard and Gruba, 2001; Von Gorp and Grissom, 2001). What these applications lack is a generic pedagogical framework that may be used to bridge the gap between the constructivist principles and software development education. Such a framework would facilitate the transfer of constructivist insights into educational practice within different contexts.

Mayes and Fowler offer a model that may be used to bridge the gap between the relevant theoretical insights of constructivism and academic disciplines. Mayes and Fowler propose a three-stage model, or learning cycle, in which they identified three types of learning: conceptualization, construction, and dialogue. Accordingly, learning develops in three phases, beginning with conceptualization, progressing through construction to dialogue. This describes learning as a cyclical dynamic feedback process (Fig. 1). Conceptualization is characterized by the process of interaction between the learners' pre-existing framework and new knowledge. The construction phase – the intermediate phase of learning - requires the building of new conceptualizations through the performance of meaningful tasks. Dialogue is the final stage of learning where learners test the new

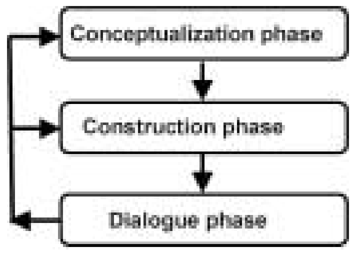


Fig. 1. Mayes and Fowler's model of the learning cycle.

conceptualizations during conversation with fellow learners and instructors. Dialogue emerges through collaborative learning.

3. Constructivism and Software Development: Methodological Considerations

The starting point for applying the learning cycle to software development was to split the learning process into three phases: a starting phase, an intermediate phase, and a final phase of learning. The goal of the first phase (conceptualization phase) is to initiate the interaction between the students' pre-existing framework and new knowledge. The goal of the intermediate phase (construction phase) is to teach the skills needed to build new conceptualizations through the performance of meaningful software development tasks. The final phase of the learning cycle is the dialogue phase where learners test the new conceptualizations during conversation with fellow learners and instructors. Dialogue emerges through collaborative learning and project activities.

3.1. *Conceptualization Phase: Interaction between the Learners' Pre-existing Framework and New Knowledge*

The conceptualization phase refers to the learner's "initial contact with other people concepts. This involves an interaction between the learner's pre-existing framework of understanding and a new exposition" (Mayes and Fowler, 1999, p. 6). This phase requires being clear about the prior knowledge and skills of the students before entering the field of object-oriented software development, because subsequent learning depends on. This observation agrees with constructivism which asserts that students' prior knowledge must be taken into consideration by the construction of new knowledge, because understanding a student's behavior requires an understanding of the student's prior knowledge. The prerequisite skills that were required before entering the field of object-oriented software development were sufficient experience with object-oriented programming with Java and database development with the JDBC, MySQL, and Java Servlets.

3.2. *Construction Phase: Performance of Meaningful Tasks and Acquisition of Software Development Skills*

The construction phase "refers to the process of building and combining concepts through their use in the performance of meaningful tasks. Traditionally these were tasks like laboratory work, writing, preparing presentations. The results of such process are product like essays, notes, handouts, laboratory reports and so on" (Mayes and Fowler, 1999, p. 6). In the field of object-oriented software development, the construction phase requires the building of object-oriented concepts through their use in the performance of software development tasks. These are normally requirements analysis, design, coding, testing, and reuse of components. Through the performance of these tasks, students acquire specific skills needed by software developers to perform their work during the software lifecycle (Seffah and Grogono, 2002; Tegarden and Sheetz, 1999). The focus of the construction phase is therefore the acquisition of specific skills:

- analysis skills, such as understanding, describing, refining, and representing the problem domain using object-oriented concepts, including user requirements, use cases; and simulation of scenarios;
- design skills, such as architectural, package, component, and deployment design; cohesion and coupling issues, including collaboration design and elaboration of use cases;
- coding and testing skills, such as program coding, testing, evaluating, and debugging of the evolving code solution, and consistency checking;
- reuse skills, such as modifying, adapting, customizing, and reusing existing objects, including the reuse of solution ideas, such as analysis, design, and program code solutions.
- critical thinking skills, such as evaluating, explaining, and justifying software development solutions.

3.3. *Dialogue Phase: Project Work, Reuse of Previous Experiences, and Collaborative Learning*

The third step refers to "the testing and tuning of conceptualizations through use in applied contexts" (Mayes and Fowler, 1999, p. 6). These conceptualizations are tested and further developed during conversations and dialogue with both fellow students and instructors and in the reflection on these. Such conversations may benefit from resources that were used by previous learners. This involves recording real-world experiences of previous learners in similar situations and making these accessible for new learners. In addition, the constructivist paradigm recommends to focus on realistic, intrinsically motivating problems that are situated in some real-world tasks. In this regard, project work is particularly suited to get students actively involved in the testing of conceptualizations (Blake, 2003; Sebern, 2002; Tomayko *et al.*, 2002).

Through project activities, students acquire two types of generic skills. First, writing and reading skills, such as writing and formatting project documentation and reading texts and documents. Second, collaborative skills, such as interacting with other learners and working in teams. Each member of a team must be assigned a role with team responsibilities that contribute to the final project solution. Generic skills are essential for software developers in a work situation (Seffah and Grogono, 2002).

4. The Learning Cycle and Software Development Education

Normally, software development is taught during the fifth and the sixth semester of the Bachelor Study Program in Computer Science at the Faculty of Mathematics (Hadjerrouit, 2003a; Hadjerrouit, 2003b). However, because of curriculum changes due to the implementation of the Quality Reform in Norway in 2003, software development was not taught in 2004.

The subject matter was re-designed for the first time in the fall semester of 2002 according to Mayes and Fowler's learning cycle. To apply the learning cycle, the subject

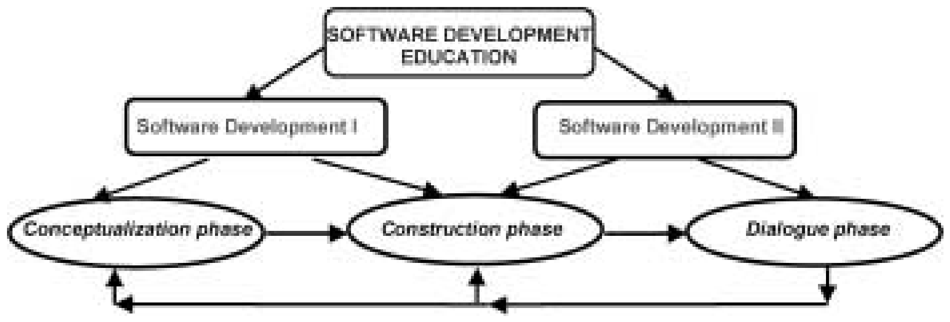


Fig. 2. Mayes and Fowler's model applied to software development education.

matter was split into two closely related courses: Software Development I and Software Development II.

The goals of Software Development I are twofold. First, to initiate the interaction between the learners' pre-existing knowledge (object-oriented programming skills, database development) and the new knowledge to be learned, that is object-oriented software development with UML (conceptualization phase). Second, to teach specific skills (analysis, design, analogical thinking, critical thinking, etc.) needed to construct software applications (construction phase).

The goals of Software Development II are threefold. First, to apply the concepts and skills acquired in Software development I. Second, to help students acquire generic skills, that is reading, writing, and collaboration skills, needed to perform project activities. Finally, to gain practical experience through involvement in real-world projects. Software Development II is thus concerned with the realization of the dialogue phase of the learning cycle (Fig. 2).

From a constructivist point of view, the cornerstone of the learning process is the Unified Modeling Language (UML). UML differs from other modeling languages such as Data Flow Diagrams. It enables to use a common set of terms and concepts throughout the whole modeling process and, therefore, makes the modeling process easier, for at least three reasons (Maciazeck, 2001). First, students are supposed to possess sufficient prerequisite knowledge in object-oriented programming. Thus, it may be easier for them to move from object-oriented programming to object-oriented modeling with UML. Second, UML facilitates the reuse of objects and analogical thinking, because of the similarities of objects and their collaborations. Finally, UML is favored in industry, and this is a major motivation for students for pragmatic considerations.

4.1. *Software Development I*

Software Development I was given in the form of four hours of lectures and four hours of laboratory work per week during a 14-week semester (Hadjerrouit, 2003a).

To realize the conceptualization and the construction phases of the learning cycle, the course content was divided into seven blocks (Table 1). The main focus of block 1 is to introduce object-oriented concepts and the basic principles of object-oriented software

Table 1

Software development I: weekly schedule

Timeframe	Block	Focus
Week 1	Block 1	Object-oriented concepts.
Week 2		Principles of object-oriented software development. Software process models.
Week 3	Block 2	Basic principles and diagrams of the Unified Modeling Language (UML).
Week 4	Block 3	Requirements determination and specification.
Week 5		Class and use case modeling.
Week 6		Interaction and state chart modeling.
Week 7	Block 4	Software design. Reuse of components
Week 8		Package, component and deployment design.
Week 9		Collaboration design.
Week 10	Block 5	User interface design.
Week 11		Database design.
Week 12	Block 6	Program and test design. Verification and validation techniques.
Week 13	Block 7	Repetition. Solutions of lab assignments and past exams.
Week 14		Repetition. Solutions of lab assignments and past exams.

development. Block 2 explains the basic concepts and diagrams of UML. Block 3 focuses on analysis modeling with UML. Block 4 covers design modeling with UML. Block 5 focuses on user interface design and database design. Block 6 presents program and test design, verification and validation methods and techniques. Finally, block 7 is devoted to repetition of the material covered during the semester, and to solutions of lab assignments and past exams. Block 1 and 2 are concerned with the conceptualization phase and block 3, 4, 5 and 6 are concerned with the construction phase of the learning cycle.

Laboratory work consists of doing lab assignments in groups of two to four students. There are six assignments each semester. In order to guarantee that every participating student acquires a minimum of object-oriented software development skills, only students who submitted well-designed solutions to the lab assignments were allowed to continue with Software Development II.

4.2. *Software Development II*

The intention of Software Development II is concerned with applying the concepts and the skills that the students acquired in Software Development I for the performance of project activities (Hadjerrouit, 2003b). Accordingly, the major goal was to work in teams and to go through the whole software life cycle from requirements analysis to delivery of a useful and well-documented piece of software.

The dialogue model which had been suggested according to the learning cycle, was one in which students were recommended to work in small groups of two to four people,

Table 2

Software Development II: Project phases, duration, focus and deliverables

Project Phase	Duration	Focus and Deliverables
Analysis modeling	4 weeks	Project milestones. Requirements elicitation, class diagrams and use case modeling. Delivery of analysis report.
Design modeling	4 weeks	Architecture, component, and deployment design. User Interface design. Prototyping. Improvement of the analysis document. Delivery of design report.
Coding and testing	2 weeks	Start coding and testing. Improvement of design document.
Final report	2 weeks	Complete project report. Finish coding and testing. Delivery of project documentation.
Oral presentation	2 weeks	Open presentation to fellow students, instructor, and stakeholders. Discussions.

cooperatively developing a software system. The organization of the projects in small groups allowed the supervision to be achieved through regular group meetings in dialogue with the instructor.

To create an atmosphere of reality according to the constructivist learning theory, students were encouraged to specify their own projects in collaboration with an external organization. This approach would allow students to develop applications that are marketable and important for their professional career.

To benefit from resources of previous versions of the course, students were recommended to reuse the learning experiences of previous students. This requires the acquisition of analogical reasoning, such as searching for similarities and differences between their own project work and past project solutions that might be modified, extended, and reused to meet the requirements of their own projects.

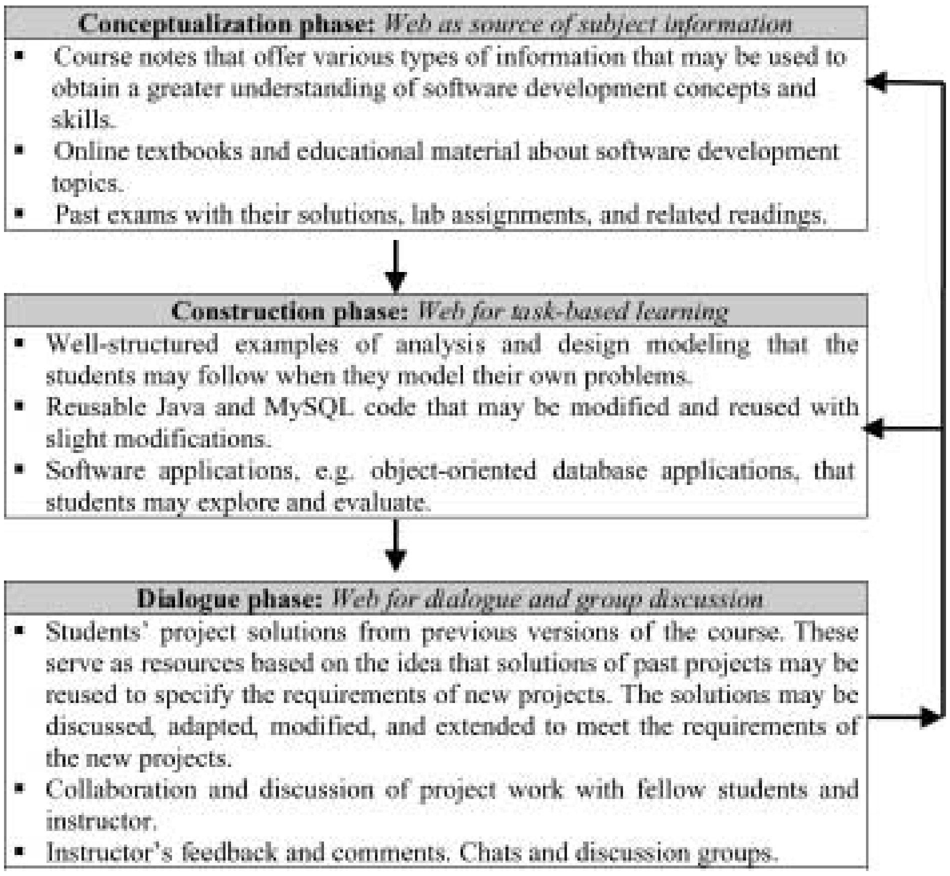
Software Development II was given in the form of eight hours project work, in small groups, per week. During a 14-week semester, team students were required to submit three written reports covering the analysis, design, and implementation phases, including a final report covering the entire project. Working in small teams, students were given four weeks to complete analysis modeling, four weeks for design modeling, two weeks for coding and testing, and two weeks for delivering the entire project report. The last two weeks were devoted to oral presentation and discussion of the projects (Table 2).

5. Online Learning Environment

The online learning environment of the software development courses was redesigned in 2002 and improved in 2003 to promote the learning cycle in three phases: conceptualization, construction, and dialogue.

Mayes and Fowler (1999) describe a framework which they can use to distinguish three types of courseware and their mapping to the three-stage of learning. Primary courseware is intended mainly to present the subject matter. Secondary courseware describes the environment and a set of tools by which the learners perform learning tasks. Tertiary courseware is material which was used by previous learners and that may be reused by new learners to solve their own problems and the testing of the solutions in dialogue with instructors and fellow students. According to Roberts (Roberts, 2003), this model has been adapted to categorize three uses of the Web (Fig. 3):

- First, to support the conceptualization phase, the Web was designed as a source for subject information to get either a greater understanding of software development concepts or to obtain further information about them (Hadjerrouit, 2003a). The most important criteria that have to be considered when designing Web-based resources for conceptualization are a well-structured presentation of the subject matter and easy accessibility of the information available on the Web.
- Second, to support the construction phase, the Web was designed to help students benefit from well-structured examples of analysis and design modeling that the students may follow when they model their own problems, as well as well-designed



Conceptualization phase: *Web as source of subject information*

- Course notes that offer various types of information that may be used to obtain a greater understanding of software development concepts and skills.
- Online textbooks and educational material about software development topics.
- Past exams with their solutions, lab assignments, and related readings.

Construction phase: *Web for task-based learning*

- Well-structured examples of analysis and design modeling that the students may follow when they model their own problems.
- Reusable Java and MySQL code that may be modified and reused with slight modifications.
- Software applications, e.g. object-oriented database applications, that students may explore and evaluate.

Dialogue phase: *Web for dialogue and group discussion*

- Students' project solutions from previous versions of the course. These serve as resources based on the idea that solutions of past projects may be reused to specify the requirements of new projects. The solutions may be discussed, adapted, modified, and extended to meet the requirements of the new projects.
- Collaboration and discussion of project work with fellow students and instructor.
- Instructor's feedback and comments. Chats and discussion groups.

Fig. 3. The learning cycle and associated online resources in software development.

object-oriented software applications and reusable Java and MySQL code that may be modified and reused with slight modifications (Hadjerrouit, 2003c).

- Finally, the Web was designed to support the dialogue phase of the learning cycle, enabling students to test their project solutions through email and Web-enabled discussions with the instructor and fellow students. In this case, the Web was used as a medium for dialogue to support collaborative learning. The testing may also benefit from previous students' learning experiences and project work from past versions of the course. Usually, these contain project documentation, reusable program code, well-designed analysis and design solutions that may be modified and reused to meet the requirements of the new projects (Hadjerrouit, 2003b).

6. Formative vs. Summative Evaluation

The concern of constructivist assessment "is not the mastery of a test but rather the ability to function successfully in the environment. This includes the ability to notice when particular skills and information are called for, to be able to recall or find that information, and to be able to apply those skills and that knowledge to solve a real world problem." (Honebein *et al.*, 1993, p. 90). As a result, constructivists advocate evaluations that focus on the authentic use of information and skills to solve real-worlds problems. Such evaluations require methods that are integrated into the learning process, so that, as learners are acquiring knowledge in authentic tasks, instructors can evaluate what the learner is learning. Hence, constructivist learning will be appropriately implemented only if students are evaluated constructively. To assess students' learning according to constructivism, it is thus important to collect data related to the three-stage model of the learning cycle: conceptualization, construction, and dialogue. Accordingly, the following issues should be evaluated:

1. The process of interaction between students' pre-existing knowledge and the level of difficulty, scope, and depth of the subject matter (1. phase of the learning cycle).
2. The degree of support provided by the construction phase to acquire software development skills (2. phase of the learning cycle).
3. The extent to which the dialogue phase supports collaborative learning among students and instructor (3. phase of the learning cycle).

Assessment of performance by means of oral/written exams may be considered as summative evaluation methods. Summative assessment is the attempt to summarize student learning at some point in time, say the end of a course. Most standardized tests are summative. In constructivist learning environments where the emphasis is the acquisition of critical skills, summative assessments like oral/written exams have advantages when it comes to assess factual recall and memorization, but they are not completely consistent with the learning cycle that takes place in those environments. Obviously, summative assessment does not automatically encourage students to think constructively when used in written/oral exams, because it is quite possible to pass an exam or test without using constructivist problem-solving techniques (Taxen, 2004; Lambert and Lines, 2000). Therefore, they should not be the only form of assessment.

By contrast, formative assessment occurs when teachers feed information back to students in ways that enable the student to learn better, or when students can engage in a similar, self- reflective process. Formative assessment is asking questions in order to determine the learners' current understanding, so that they can make adjustments if necessary (Beverly and Bronwen, 2002). It is based on the principle that the evaluation of learning should not be separated from the learning process. Hence, assessment should be embedded in the learning process and spread out over the duration of the course.

Assessment by means of project work may be considered as formative assessment, because the evaluation of project work is embedded in the learning process and is grounded in authentic tasks that are spread over the duration of the course. In addition, the emphasis of project-based work is on the acquisition of software development skills. This is in line with constructivism.

In addition, formative assessment may include issues of qualitative character to assess a greater portion of the learning cycle, such as motivational aspects, effort and time required to perform project activities, students' learning difficulties, teamwork and collaborative learning, students' perceptions, meanings and beliefs, etc. Many of these issues may be difficult to evaluate with standard assessment methods alone. Evaluation methods, such as semi-structured interviews and dialogue with the students, teacher's observations in the classroom, and students' comments and feedback are more appropriate to evaluate qualitative issues (Salomon and Perkins, 1998).

Thus, in an attempt to provide a consistent evaluation of the learning cycle, the author advocates a combination of two methods:

1. Assessment of project work performance to measure the degree of implementation of the learning cycle.
2. Qualitative evaluation of the learning cycle based on students' feedback and comments, semi-structured interviews, and teacher's observations.

7. Evaluation of the Learning Cycle through Project Performance Assessment

In line with constructivism, assessment of project work differs from conventional tests. First, unlike conventional measures that tend to evaluate student's possession of knowledge as well as factual recall and memorization of facts, project work assessment evaluates students' ability to apply software development knowledge and skills to solve real-word problems. Hence, the evaluation of project work was based on the constructivist idea that it is not possible to separate the evaluation of learning from the learning process. Accordingly, assessment of students' project work must be embedded in the learning process and spread out over the duration of the course. To assess their learning, students were required to submit three written reports covering the analysis, design, and implementation phases, and a final report covering the entire project. The instructor, then, monitored students' project work by examining reports, replying to e-mail, providing oral and written feedback as necessary in order to help students revise and improve their project work, and ensure that they meet the course objectives. Monitoring project work and providing feedback is an iterative process that occurs throughout the entire learning process. Finally, students had to present the project work orally to the whole class. The evaluation consisted of assessing:

a) the understanding of key software development concepts (1. phase of the learning cycle);
b) project work performance and the quality of the submitted project reports (2. phase of the learning cycle);
c) the presentation of project results to the whole classroom (3. phase of the learning cycle);
d) the active participation, collaboration, and contribution of students to group work (3. phase of the learning cycle).

Grades were based on a six-point scale from A to F, where F was coded as the lowest and A as the highest. Score E was required in order to pass the subject matter. Before 2003, grades were based on a six-point scale from 1 to 6, where 6 was coded as the lowest and 1 as the highest. Score 4 was required in order to pass the subject matter. This scale does not affect the interpretation of the evaluation results.

Based on students' performances, it appears that project work was an effective method for developing students' software development skills and project work activities. The scores exhibited by 3 student teams in 2003 were: 2 teams received an "A" and 1 received a "B". In 2002, 1 team received an "A" (1.3 in a six-point scale) and 2 teams a "B" (1.8 and 2.1 in a six-point scale).

These grades indicate that the overall performance of the students in 2003 and 2002 was clearly higher compared to the previous versions of the course in 2001, 2000 and 1999. All the students who completed the course indicated that the use of project work assessment to determine their grades was appropriate, especially considering that they were given the opportunity to submit and revise their work many times throughout the semester.

A possible interpretation of the positive results is that project work encourages students to think constructively in order to successfully perform project activities, because the emphasis is not on factual recall and memorization of facts or the mastery of a test, but on activities that are embedded in the learning process. The positive grades could be, to some degree, attributed to the fact, that the students received a greater amount of guidance in project work during the whole semester. Thus, it may very well be that students felt that constructivist-oriented learning activities are appropriate to achieve deeper learning. Thus, it seems that project work has many advantages to assess quantitative aspects of the learning cycle, but it is not entirely sufficient to evaluate qualitative aspects of constructivist learning, such as students' thoughts, misconceptions, behavior, difficulties, that are of particular importance for novice students entering the field of software development. Therefore, it should not be the only form of assessment (Salomon and Perkins, 1998).

8. Qualitative Assessment through Students' Feedback and Teacher's Observations

Considering that project work assessments alone are not sufficient to give a reliable picture of the learning cycle, the evaluation was extended to include issues of qualitative character of the learning process. Thus, the concern of the qualitative evaluation was to assess a greater portion of the learning cycle. The evaluation was based on experiences from the academic year of 2002/2003. The participants were 8 students from the academic year of 2002/2003.

8.1. *Data Collection and Analysis Methods*

With an emphasis on understanding the students' meanings, experiences, thoughts, perceptions and beliefs about teaching and learning software development, data collection methods that are consistent with constructivist principles and the learning cycle were required. Given this consideration, particular attention was devoted to the following data collection strategies:

a) defining the subjects of inquiry and associated issues that fit the learning cycle;
b) semi-structured interviews and formal and informal dialogue with the students;
c) teacher's observations in the classroom over a three-month time period (scheduled time);
d) comparing the data that was collected in the academic years of 2000/2001 and 2001/2002 with data collected in the academic year of 2002/2003;
e) when possible, finding evidence in the research literature that supports or contradicts the data collected.

The method used for data analysis consisted of finding diverse pieces of evidence from four different perspectives: teacher's perspective, students' perspective, the perspective of the research literature, and the perspective of the data collected before the evaluation was performed. To ensure the rigor and validity of analysis, data sources were triangulated through overlapping of diverse pieces of evidence and perspectives (Teacher Education Research, 2005), and an active search for conforming and disconfirming evidence was made through new dialogue and conversations with the students. These conversations consistently resulted in deeper understanding of the students' experiences.

To facilitate the analysis of the data collected, the subjects of inquiry and associated questions of the semi-structured interviews and dialogue with the students were closely aligned with the teacher's observation criteria. To define the subject of inquiry according to the learning cycle, data collection was divided into three categories:

1. Category one was related to the first phase of the learning cycle, and included the following subjects of inquiry: Prerequisite knowledge, course objectives, content understanding, and knowledge level.
2. Category two was related to the second phase of the learning cycle, and included the following subjects of inquiry: Analysis and design modeling, reuse of components and analogical reasoning, coding and testing, and critical thinking.

3. Category three was related to the third phase of the learning cycle, and included the following subjects of inquiry: Reading and writing, dialogue with instructor, teamwork and collaboration, and motivation.

Each of these categories was associated with 4 types of issues (Table 3, 4, and 5). Category 1 addressed issues 1–4. Category 2 was concerned with issues 5–8, and category 3 with issues 9–12.

8.2. *Evaluation of the First Phase of the Learning Cycle*

The evaluation of the first phase of the learning cycle was concerned with the students' initial contact with the concepts of the subject matter. This phase requires being clear about the prior knowledge and skills of the learners before entering the field of object-

Table 3
First phase of the learning cycle: subjects of inquiry and associated issues

Subjects of inquiry for the conceptualization phase of the learning cycle	**Associated Issues**
1. Prerequisite knowledge	1. Do you agree that the course framework takes into consideration the students' prerequisite knowledge and experience?
2. Course objectives	2. Do you think that the intended learning objectives of the course are achieved?
3. Content understanding	3. Do you believe that the course framework helped you understand the knowledge, concepts, and methods of the subject matter?
4. Knowledge level	4. What do you think about the level of difficulty, scope, and depth of the course? Is it appropriate? Difficult? Easy?

Table 4
Second phase of the learning cycle: subjects of inquiry and associated issues

Subjects of inquiry for the construction phase of the learning cycle	**Associated Issues**
5. Analysis and design modeling	5. Do you believe that the course framework provides appropriate support to help you perform analysis and design modeling? Yes/no. Please, explain why.
6. Reuse and analogical reasoning	6. Do you think that the course framework provides sufficient support to help you acquire reuse and analogical thinking skills? Yes/no. Please, explain why.
7. Coding and testing	7. Do you agree that the course framework provides sufficient support to help you perform coding and testing?
8. Critical thinking	8. Do you agree that the course framework helped you develop critical thinking skills? Yes/no. Please, explain why.

Table 5

Third phase of the learning cycle: subjects of inquiry and associated issues

Subjects of inquiry for the dialogue phase of the learning cycle	Associated Issues
9. Reading and writing activities	9. Do you think that reading and writing skills are important to software development education? Do you believe that the course framework sufficiently supports reading and writing activities?
10. Dialogue with instructor	10. Was the dialogue with the instructor effective in terms of facilitating your learning? Did it help you reflect on the strengths and limits of you own knowledge of software development?
11. Teamwork and collaboration	11. Do you agree that the course framework provides appropriate support for teamwork and collaboration among students and other people?
12. Motivation	12. Did the course framework and content (project activities, real-world tasks) support your motivation and engagement in the subject matter? Yes/no. Please, explain why.

oriented software development, because subsequent learning depends on. The subjects of inquiry of the first phase of the learning cycle were: Prerequisite knowledge, course objectives, content understanding, and knowledge level. These subjects were related to Software Development I**.**

Analysis of the responses to issues 1, 2, 3, and 4 show that students were globally positive about these issues.

First, most students' believed that the course framework takes into consideration the students' background knowledge, because the construction of object-oriented software development build upon the object-oriented programming language Java and database development with the JDBC, MySQL, and Java Servlets.

Second, the majority of the students felt that the intended learning objectives of Software Development I are achieved, namely to initiate the interaction between the students' pre-existing knowledge and object-oriented software development with UML, and to acquire specific skills (analysis, design, analogical thinking, critical thinking, etc) needed to construct software applications. These results show that the course framework in Software Development I was well-designed to bridge the gap between the students' background knowledge and the new knowledge. In correlation with this issue, the majority of the students also found that the course framework helped them to gain knowledge and understanding of software development concepts and skills.

Finally, over 50 % of the students agreed that the knowledge level of the course is acceptable due to an appropriate balance between conceptual understanding of software development and practical lab assignments.

8.3. *Evaluation of the Second Phase of the Learning Cycle*

The evaluation of the second phase of the learning cycle – the construction phase – was concerned with the acquisition of specific skills, that is analysis and design modeling, reuse and analogical thinking, coding and testing, and critical thinking. This evaluation was mainly related to Software Development II.

8.3.1. *Analysis and Design Modeling*

Basically, most students agreed that Software Development II provides sufficient support to analysis and design modeling (issue nr. 5). But, students also reported that analysis modeling with UML (requirements determination and specification, class and use case modeling, interaction and state chart modeling) was a relatively difficult task, essentially due to insufficient experience with modeling problem situations. The instructor agrees that analysis modeling is a challenging task for many novice students entering the field of software development as it requires a radical change from lower–order thinking skills, characterized by the tendency to focus on programming issues, to higher-order thinking skills required by software engineers to perform analysis modeling on the basis of previous experiences with solving problems in similar situations. This observation agrees with students' experiences in analysis modeling. By contrast, students found that design modeling (package, component and deployment design, collaboration design, user interface design and database design) was relatively easier compared to analysis modeling, for essentially two reasons. First, moving from analysis modeling to design modeling does not require a radical change of the object-oriented methodology with UML (Maciazeck, 2001; Stevens and Pooley, 2000). Second, the reuse of past project solutions helped them to perform design modeling. Most students agreed with these observations.

8.3.2. *Reuse and Analogical Reasoning*

From a software development point of view, it is not necessary to develop object-oriented software from the ground, since the reuse of objects is an essential element of object-orientation. Similarly, the reuse philosophy of this work relies on the basic idea that project activities are similar. Hence, solutions of past projects may be adapted and reused to meet the requirements of new projects. Thus, in order to achieve effective reuse, students need to acquire some experience in analogical reasoning, such as recognizing similarities and differences between past and new projects. Analogical thinking includes a search for similarities between what is already known and the new, the familiar and the unfamiliar (Duit *et al.*, 2001).

To help students acquire some experience in analogical thinking, the instructor used Web-based applications as examples to demonstrate the use of analogies in terms of structural similarities. Analogical thinking and reuse are potentially relevant for building e-commerce applications and instructional Web sites, since both have a number of similar components, e.g., user interface, shopping card, product catalogue for e-commerce applications; reusable learning objects for instructional Web sites (Hadjerrouit, 2005). To apply analogical thinking, students must be encouraged to take an active role in constructing their own understanding of project solutions from previous versions of the course in order to reuse some components of the projects.

However, analogical reasoning is not quite evident for novice students in software development. It takes time and effort, even for proficient students, to learn to distinguish between deep similarities (based on structural and architectural features of the applications) and surface similarities, such as graphical user interface appearances (colors, layout, style). Not surprisingly, then, that most students reported that the reuse of components during the phase of analysis modeling was quite difficult due to insufficient experience in analogical thinking (issue nr. 6).

Clearly, students must learn to carefully read reports from previous versions of the course in order to reflect on and discover the structural characteristics of the software solutions. These activities, in turn, may help them to effectively improve their analogical thinking skills and the quality of the projects at the end. As a result, even if analysis modeling remains a problem, analogical thinking provides a great potential for improving the project quality.

8.3.3. *Coding and Testing*

Students generally agree that Software Development II provides support to achieve coding and testing, in form of reusable Java and MySQL code available on the Web and from previous versions of the course (issue nr. 7). But the support provided by the course would not be sufficient without prior knowledge from the programming language Java and database development with the JDBC, MySQL, and Java Servlets, since the coding and testing of object-oriented software depends on. Students agreed that prior experience from object-oriented programming and database development was crucial for implementing object-oriented software, but not without some problems. Students reported that coding and testing were quite short but nevertheless did work. This because the major part of the code was written and tested in the last two weeks before delivery, resulting many times in late night work. Another problem was that the testing phase was not realized as originally planned. Often, students did not manage to stop coding at the proper time, in order to be able to systematically test the software. As a result, the quality and completeness of the test cases were rather of moderate quality for some teams, but this did not produce major problems for the delivery of the software, because it was not required to deliver a complete system, but just a prototype that realizes the main functionality.

8.3.4. *Critical Thinking*

As a result of passive listening to lectures, taking notes, assessing information from textbooks, students generally do not sufficiently know the value of critical thinking before entering the field of software development. Since software development was being taught for the first time in 1999, the instructor observed that students often focus, from the very beginning, on the software product rather than on the solution process. They often conceive a solution just as a solution that works for them, rather than a solution that is readable for others (Mereno-Seco and Forcada, 1996; Hadjerrouit, 1999). This problem is particularly visible in the phase of analysis modeling.

Not surprisingly, then, most students do not sufficiently know what it means to acquire critical thinking skills when asked whether the course framework helped them develop critical thinking skills. Their responses reflect their beliefs about learning, which are clearly mediated and impacted by a passive transmission view of knowledge acquisition (issue nr. 8).

To dislodge this misconception, students need to learn to reflect on their own solutions. It is however quite difficult to teach critical thinking skills within a one-semester course, because of the high workload of the projects and the short time frame.

Fortunately, the reading of well-structured reports and software solutions from previous versions of the course and the reflection on these helped some students, within a relatively short time, to understand the value of critical thinking. They learned to ask critical questions about their own work: What is a software product vs. software process? What difficulties problems were encountered in developing software? What cause the difficulties? How will they overcome them? What general principles may be extracted from the learning experiences? What patterns have they perceived in the problems? Clearly, learning experiences like these often give students the necessary pieces of how to develop well-structured software and documents that are readable and understandable for others.

8.4. *Evaluation of the Third Phase of the Learning Cycle*

The evaluation of the third phase of the learning cycle – the dialogue phase - was concerned with the quality of project work, the reading and writing of reports, teamwork and collaboration, dialogue with the instructor, and motivational aspects.

8.4.1. *Reading and Writing Activities*

Students believed that the consideration of writing and reading activities is a key issue of software development education. This is reflected in their responses when asked about the value of reading and writing documentation (issue nr. 9). They think that these skills would help them understand and reflect on the writings of other people and to write documents that are understandable for others (Tynjaelae, 1999). This observation supports previous research, wherein Spivey (Spivey, 1997) found that the process of reading reports written by other students in order to write their own reports can produce rich interferences and elaborations. They believed also that the ability to write and read both in Norwegian and English is an advantage for any student and a prerequisite for a successful professional life.

However, despite the availability of well-structured reports from previous versions of the course, students reported that the elaboration of the analysis document, and, in less degree, design report, took more time than expected due to the amount of work required to write substantial analysis and design project reports. Students reported that they had difficulties to write a stable document, and were forced to change it many times. As a result, project work had to be adjusted several times due to some misjudgments about time consuming, particularly in the analysis phase, which was often longer than planned. This created some problems during the design and coding phases, which were not rigorously planned and prepared as expected, and some deficiencies in the resulting documentation.

Nevertheless, the instructor was satisfied with the documentation delivered by the students given the fact that they were given only three months, a very short period of time, to write substantial analysis and design reports with over 450 pages. Clearly, proficiency in reading and writing requires considerable efforts, and can, therefore, only be acquired through active involvement and engagement with project work over a long time.

8.4.2. *Dialogue and Scaffolding*

Students agreed that Software Development II places a large emphasis on dialogue between the students and the instructor (issue nr. 10). They believed that dialogue enables to express a point of view and reflect one's own learning when solving problems with the instructor.

In order to facilitate dialogue, the instructor also had to learn how to understand students so that they can interpret their responses better and guide communication more effectively (Miller and Luse, 2004; Soendergaard and Gruba, 2001). Thus, it was not surprising for the instructor that dialogue with the students turned out to be more time consuming and challenging than originally anticipated. Dialogue may therefore be a challenge for inexperienced instructors as constructivism requires to make a radical change in their thinking and practice. In a constructivist setting, the instructor does not just convey information or supply facts, but must act as a mentor, facilitator, guide, coach, and mediator. Such a drastic change of attitude is difficult for any teacher, and certainly for university faculty members who are not educational researchers.

On the other hand, scaffolding (Robert-Jan Simons, 1993; Dunlap and Grabinger, 1998), that is when the instructor guides the learners towards a solution to a software problem, essentially allows adjusting the level of learning and helps the students (and their teams) according to their abilities. When adjusting the help to the students, the instructor felt that those who had initially been weaker had improved during the course, and had achieved relatively good results.

Despite the challenges of dialogue with the students and the time and effort required to act as a guide and facilitator of learning, scaffolding was an interesting experience from the instructor's perspective and a good way to experience practical problems of managing the student's learning. But, in the form presented here, scaffolding and dialogue can be suggested only for rather small groups, say up to 20 students, essentially because it is time consuming, challenging, and expensive for the university.

8.4.3. *Teamwork and Collaboration*

From a constructivist point of view, teamwork is one of the most important characteristics of software development, since group work decisions, expressing the point of views of the members, are better than individual decisions when it comes to develop god software (Blake, 2003; Frank *et al.*, 2003; Miller and Luse, 2004). Teamwork involves a lot of activities: working with students with different backgrounds and experiences, sharing out project tasks, organizing project activities, collecting and distributing information, writing up protocols of meetings, nominating a leader, and many other issues which affect the group decision process. It is thus evident that in order to achieve effective learning in a team, students must be trained in teamwork before performing project activities, because a random collection of students does not necessarily make for an effective team (Frank *et al.*, 2003). Unfortunately, many students were not trained in teamwork before entering the field of software development.

Not surprisingly, then, that some students were not prepared to achieve effective collaboration. This is reflected in their responses to issue nr. 11. As a result, they did not manage to solve problems between members of their team, resulting sometimes in frustration and insufficient collaboration, and this affected the effectiveness of their team. Effective collaboration may be a challenge when some students are more proficient than other members of their team. Two students reported that they felt that teamwork – a constructivist activity that is supposed to help students play active role in solving problems - is a waste of time and is in conflict with the goal at hand – deliver a successful project work within a limited period of time. Thus, there is a danger that students see software development education as exclusively the acquisition of performance skills and neglect the value of teamwork.

It is the responsibility of the teacher to help students understand the value of teamwork, both from a theoretical and practical point of view. Hence, the instructor has to intervene in order to help students overcome their problems whenever students have difficulties with teamwork. According to the teacher's experience the last five years, most students performed better when the teacher provided help and support. However, when not prompted by the teacher, students tended not to solve their problems by their own.

8.4.4. *Motivation, Real-World Projects, and Professional Career*

A major goal of the constructivist approach to software development is to try to be as close as possible to reality, and to involve as many people as possible in project activities and discussions. But still, in contrast to real-world projects from the industry, there were only three months time to deliver the software product – just enough to go through a full life cycle of small software projects. Therefore one could not expect students to design a complete software product. Therefore, the learning goal was to let students experience the challenges of developing small, but well-structured and documented software. The instructor believes that this goal has been achieved.

Furthermore, despite the limited timeframe, students were very enthusiastic about the real-world character of the projects, which clearly created a proper context for discussion for many students. Most students felt that this course increases their motivation to learn and to make greater efforts in the right direction. Thus, according to the students' experiences, it is evident that the course developed their engineering skills, increased their motivation, and made them feel that they were responsible for the learning process.

Students reported that their motivation is directly related to the marketability of the projects, the profession of software development, and future professional career. When asked about the relevance of marketability for their professional career, students were very clear in their responses. They reported that software development combined with real-world projects is highly motivating (issue nr. 12) and relevant to them as it can help them to gain practical experience and knowledge in software development and to increase their marketability. Thus, the motivational aspect should not be underestimated, since marketability seems to be very important for students.

Clearly, motivation was one of the most important factors for project's success (Green, 1998; Hadjerrouit, 1999; Hadjerrouit, 2001). High motivation of all participants in a team resulted always in god working atmosphere which was very positive for the performance of project activities. This experience is consistent with the constructivist point of view, which asserts that real-world projects are motivating to get students actively involved in knowledge construction and skill acquisition. Clearly, motivating the students is an essential element of constructivist learning.

8.5. *Evaluation of the Online Resources*

Finally, the evaluation included students' perceptions of the online resources based on feedback made through email, face-to-face discussions, and informal conversations. In contrast to the issues discussed above, the evaluation of the online resources refers to all phases of the learning cycle. Students were asked three questions:

- How did you use the online resources?
- How much have the online resources helped you to learn software development and perform project activities?
- How would you improve the online resources?

Students reported that they used the online resources in three different ways. First, as information source to gain a greater understanding of software development issues. This reflects the conceptualization phase of the learning cycle. The second use of the resources concerned the reuse of previous students' projects and their solutions to perform new project activities. In the opinion of the students, the resources available on the Web provided useful support for the construction and dialogue phases of the learning cycle. From the data collected, it seems that reuse was a relevant aspect of the online resources. The third use of the online resources involved the testing of project solutions through dialogue, particularly by means of email discussions with the instructor.

The most common and useful use concerned mainly questions and comments about project work. Otherwise, dialogue happened rather in face-to-face discussion. Quite few used group discussion forum to engage in dialogue with fellow students.

Students made also suggestions for improving the resources. First, they wanted that there should be more information and study material in the Norwegian language. In addition, they suggested improvement of the multimedia elements of the online resources to enhance look and feel of the Web pages. They also suggested to improve the dialogue component related to group discussion. Finally, they recommended to solve technical problems for ensuring a trouble-free interaction with the resources, in particular during the implementation phase, which caused some troubles.

9. Conclusions and Future Research Work

The goal of this work was to determine whether a constructivist-oriented pedagogy is suitable for software development education. In evaluating the approach after two years of experience, the instructor can draw the following conclusions.

Even if it is impossible to draw any general conclusions from the evaluation of the approach, it is clear that the majority of the students were positive about the implications of the constructivist approach. The approach appears to encourage students to think constructively and become involved in real-world project activities. The evaluation results also show that the constructivist approach to software development was proven to be beneficiary to many students. For instructors, constructivism holds important lessons for how to design environments to support active learning. It gives teachers a framework for understanding students' needs and motivations. It helps teachers to expose students to many aspects of the subject matter that are of crucial importance for the profession of software development. It allows to focus on what really matters for students – the acquisition of critical skills, authentic tasks, motivational aspects, teamwork and collaboration, reading and writing skills, formative assessment, etc.

Second, the form of this model can be suggested for rather small groups around 20 students and 3–5 student teams, because one instructor would have difficulty handling larger groups. Thus, the model needs to be studied in varied instructional settings, for example for larger groups over 20 students and more than one teacher to confirm and support the findings of this work. In addition, applying the constructivist learning theory may be a challenge for inexperienced instructors as it takes time and considerable effort to translate the philosophy of constructivism into practice and to make significant pedagogical changes.

Third, if constructivism is the appropriate pedagogical philosophy for software development, it is quite evident that assessment should not exclusively be exam-oriented, for two reasons. When students are forced to choose between passing of an exam through memorization of facts and solving problems constructively, most students will likely be to choose the former alternative (Taxen, 2004). Written exams are clearly not entirely adequate to evaluate constructivist learning, because it is difficult to measure it through performance assessment methods alone. Thus, alternative evaluation methods are necessary to assess constructivist learning. The author advocates a combination of methods that include not only project work evaluation, but also complementary methods that are embedded in the learning process.

Finally, several things could be done better, and some open questions remain. First, the instructor believes that students should have more than three months to accomplish project tasks. One suggestion is to start the projects earlier in order to give students more time for analysis modeling, which was the most difficult task. Second, it is important to write god documentation. The value of god documentation, however, often surfaces during the maintenance phase, which was not included in the project time. One way to partially solve this problem is to allow students to work with project tasks of previous versions of the course in order to improve their quality. This would make software development more realistic. Third, reading, writing, and oral skills must be improved, because they enable students to reflect on the writings of other people, to write documents that are understandable for other, and to express a point of view in the course of project work. Third, applying a constructivist approach to learning is an iterative and evolutionary process that progresses through a series of experimentations, evaluations, and redesigns over many years. Hence, significant pedagogical changes in software development education require considerable time and effort. This work is thus a long-term research work on the application of the constructivist learning theory in software development education. The constructivist model will be further developed through continuous cycles of design, experimentations, evaluations, and research directions.

References

Ben-Ari, M. (1998). Constructivism in computer science. In *Proceedings of the 29th SIGCSE Technical Symposium on Computer Science Education.* Atlanta, Georgia. pp. 257–261.

Ben-David Kolikan, Y. (2001). Gardeners and cinema tickets: high school students' preconceptions of concurrency. *Computer Science Education*, **11**(3), 221–245.

Beverly, B.F., and C. Bronwen (2002). *Formative Assessment and Science Education.* Kluwer Academic, London.

Blake, M.B. (2003). A student-enacted simulation approach to software engineering education. *IEEE Transactions on Education*, **46**(1), 124–132.

Booth, S. (2001). Learning computer science and engineering in context. *Computer Science Education*, **11**(3), 169–188.

Bruner, J. (1990). *Acts of Meaning*. Harvard University Press, Cambridge, MA.

Duffy, T.M., J. Lowyck and D.H. Jonassen (1993). *Designing Environments for Constructive Learning.* Springer-Verlag, New York.

Duit, R., W.-M. Roth, M. Komorek and J. Wilbers (2001). Fostering conceptual change by analogies – between Scylla and Charybdi. *Learning and Instruction*, **11**, 283–303.

Dunlap, J.C., and R.S. Grabinger (1998). Rich environments for active learning in the higher education classroom. In B.G. Wilson (Ed.), *Constructivist Learning Environments*: *Case Studies in Instructional Design*. Educational Technologies Publications, Englewood Cliffs, New Jersey.

Fowler, L., J. Armarego and M. Allen (2001). CASE-tools: constructivism and its application to learning and usability of software development tools. *Computer Science Education*, **11**(3), 261–272.

Frank, M., I. Lavy and D. Elata (2003). Implementing the project-based learning approach in an academic engineering course. *International Journal of Technology and Design Education*, **13**, 273–288.

Green, A.M. (1998). Project-based learning: moving students toward meaningful learning. In L.P. Steffe and J. Gale (Eds.), *Constructivism in Education*. Lawrence Erlbaum Associates, New Jersey.

Gros, B. (2002). Knowledge construction and technology. *Journal of Educational Multimedia and Hypermedia*, **11**(4), 323–343.

Halling, M., W. Zuser, M. Koehle and S. Biffl (2002). Teaching the unified process to undergraduate students. In *Proceedings of the 15th Conference on Software Development Education and Training,* (*CSEET'02*). pp. 148–159.

Hadjerrouit, S. (1998a). A constructivist perspective for software engineering education. In *Software Development Symposium* (*SEES'98*). Scientific Publishers. pp. 91–96.

Hadjerrouit, S. (1998b). A constructivist approach for integrating the Java Paradigm into the Undergraduate Curriculum. In *Proceedings of the 3th Annual Conference on ITiCSE'98*, Dublin. pp. 105–107.

Hadjerrouit, S. (1999). A constructivist approach to object-oriented design and programming. In *Proceedings of the 4th Annual Conference on ITiCSE'99*, Cracow. pp. 171–174.

Hadjerrouit, S. (2001). Web-based application development: a software engineering approach. *SIGCSE Bulletin*, **23**(2), 31–34.

Hadjerrouit, S. (2003a). *Software Development I.* `http://fag.hia.no/kurs/inf2450/www_docs/`

Hadjerrouit, S. (2003b). *Software Development II.* `htttp://fag.hia.no/kurs/inf2470/www_docs/`

Hadjerrouit, S. (2003c). *Database Development.* `http://fag.hia.no/kurs/inf2490/www_docs/`

Hadjerrouit, S. (2005). Designing a pedagogical model for web engineering education: an evolutionary perspective. *Journal of Information Technology Education*, **4**, 115–140.

Honebein, P.C., T.M. Duffy and B. Fishman (1993). Constructivism and the design of learning environments: context and authentic activities for learning. In T.M. Duffy, J. Lowyck, and D.H. Jonassen (Eds.), *Designing Environments for Constructive Learning*, Springer-Verlag, New York. pp. 88–108.

Kelly, G.A. (1955). *The Psychology of Personal Constructs*. Norton, New York.

Lambert, D., and D. Lines (2000). *Understanding Assessment*: *Purposes, Perceptions, Practice.* Routledge Falmer, London.

Maciazeck, L.A. (2001). *Requirements Analysis and System Design*: *Developing Information Systems with UML*. Addison-Wesley, London.

Mereno-Seco, F., and M.L. Forcada (1996). Learning compiler design as research activity. *Computer Science Education*, **7**, 73–98.

Matthews, M.R. (2002). Constructivism and science education: a further appraisal. *Journal of Science Educational Technology*, **11**(2), 121–134.

Mayes, J.T., and C.J. Fowler (1999). Learning technology and usability: a framework for understanding courseware. *Interacting with Computers*, **11**(5), 485–497.

Miller, R.A., and D.W. Luse (2004). Advancing the Curricula: the identification of important communication skills needed by IS staff during systems development. *Journal of Information Technology Education*, **3**, 117–131.

Piaget, J. (1969). *Judgment and Reasoning in the Child*. Routledge & Kegan Paul, London.

Phye, G.D. (Ed.) (1997). *Handbook of Academic Learning*: *Construction of Knowledge*. Academic Press.

Pullen, M. (2001). The network workbench and constructivism: learning protocols by programming. *Computer Science Education*, **11**(3), 189–202.

Roberts, G. (2003). Teaching using the web: conceptions and approaches from a phenomenographic perspective. *Instructional Science*, **31**, 127–150.

Robert-Jan Simons, P. (1993). Constructive learning: the role of the learner. In T.M. Duffy, J. Lowyck, and D.H. Jonassen (Eds.), *Designing Environments for Constructive Learning*. Springer-Verlag, New York. pp. 291–313.

Saiedian, H. (2002). Bridging academic software engineering education and industrial needs. *Computer Science Education*, **12**(1–2), 5–9.

Salomon, G., and D. Perkins (1998). Individual and social aspects of learning. In P. Pearson and Iran-Nejad (Eds.), *Review of Research in Education*, Vol. 23. American Educational Research Association, Washington DC. pp. 1–24.

Sebern, M.J. (2002). The Software Development Laboratory: incorporating industrial practice in an academic environment. In *Proceedings of the 15th Conference on Software Development Education and Training* (*CSEET'02*). pp. 118–127.

Seffah, A., and P. Grogono (2002). Learner-centered software engineering education: from resources to skills and pedagogical patterns. In *Proceedings of the 15th Conference on Software Development Education and Training*. pp. 14–21.

Soendergaard, H., and P. Gruba (2001). A constructivist approach to communication skills instruction in computer science. *Computer Science Education,* **11**(3), 203–209.

Spivey, N.N. (1997). *The Constructivist Metaphor*: *Reading, Writing, and the Making of Meaning.* Academic Press.

Steffe, L.P., and J. Gale (Eds.) (1995). *Constructivism in Education*. Lawrence Erlbaum Associates, New Jersey.

Stevens, P., and R. Pooley (2000). *Using UML*: *Software Development with Objects and Components*. Edison-Wesley, London.

Staver, J.R. (1998). Constructivism: sound theory for explicating the practice and science education. *Journal of Research in Science Education*, **35**(5), 501–520.

Tam, M. (2000). Constructivism, instructional design, and technology: implications for transforming distance learning. *Educational Technology & Society*, **3**(2), 50–60.

Taxen, G. (2004). Teaching computer graphics constructively. *Computer & Graphics*, 393–399.

Teacher Education Research (2005). *Triangulating Your Evidence*.
`http://gse.gmu.edu/research/tr/TRtriangulation.shtml`

Tegarden, D., and S.D. Sheetz (2001). Cognitive activities in OO development. *International Journal of Human-Computer Studies*, **54**, 779–798.

Tomayko, J.E., S. Kuhn, O. Hazzan and B. Corson (2002). The software studio in software engineering education. In *Proceedings of the 15th Conference on Software Development Education and Training* (*CSEET'02*).

pp. 236–238.

Tynjaelae, P. (1999). Towards expert knowledge? A comparison between a constructivist and a traditional learning environment in the university. *International Journal of Educational Research*, **31**, 357–442.

Van Gorp, M.J., and D. Grissom (2001). An empirical evaluation of using constructive classroom activities to teach introductory programming. *Computer Science Education*, **11**(3), 247–260.

Von Glaserfeld, E. (1993). *Radical Constructivism. A Way of Knowing and Learning*. Routledge Falmer, London.

Vygotsky, L.S. (1978). *Mind in Society*: *The Development of Higher Psychological Processes*. Harvard University Press, Cambridge M.A.

Wilson, B.G. (Ed.) (1998). *Constructivist Learning Environments: Case Studies in Instructional Design*. Educational Technologies Publications. Englewood Cliffs, New Jersey.

Young, R.A., and A. Collin (2004). Introduction: constructivism and social constructionism in the career field. *Vocational Behavior*, 373–388.

S. Hadjerrouit received the MS and PhD degrees in software engineering and artificial intelligence from the Technical University of Berlin (Germany), in 1985 and 1992, respectively. He joined Agder University College, Kristiansand (Norway) in 1991. He is currently an associate professor of computer science at the Faculty of Mathematics. He has been in the teaching profession for 23 years. He has extensive experience teaching object-oriented programming, Web design, database development, and software engineering. His research interests include computer science and software engineering education, didactics of informatics, e-learning, Web engineering, and object-oriented software development with the Unified Modeling Language (UML). Hadjerrouit has published over 30 papers in international journals and conference proceedings.

Objektinis informatikos mokymas: konstruktyvistinis požiūris

Said HADJERROUIT

Straipsnyje nagrinėjama konstruktyvistinės mokymosi teorijos svarba mokant informatikos, tiksliau, programinės įrangos kūrimo darbų. Konstruktyvizmas mokymąsi apibrėžia ne kaip pasyvaus žinių perdavimo rezultatą, – veikiau, kaip aktyvų konstravimo procesą, kurio metu besimokantieji konstruoja savo pačių žinias, remdamiesi anksčiau įgytomis žiniomis bei patirtimi. Šiuo metu daugelio informatikos (programinės įrangos kūrimo) kursų metu taikomas projektinis mokymas, tad atrodytų, jog konstruktyvistinės perspektyvos atveriamos galimybės yra pakankamai suprantamos ir plačiai taikomos. Vis dėlto, daugeliu atvejų pasigendama konkrečios metodologijos, skirtos konstruktyvistinės perspektyvos taikymo bei jos įtakos mokymuisi klausimams. Straipsnyje supažindinama su konstruktyvistine perspektyva, taikoma objektiniam informatikos mokymui (object-oriented software development) dirbant su baigiamojo kurso studentais. Straipsnyje aptariami metodologiniai minėtos perspektyvos aspektai, atliekama jos analizė.