

Toward a Constructivist Approach to Web-based Instruction in Software Engineering

Said Hadjerrouit
Agder University College
Department of Mathematics
N-4604 Kristiansand (Norway)
Said.Hadjerrouit@hia.no

Abstract

Software engineering education today faces two challenges. One challenge comes from the changing perceptions of what learning is about, a shift from objectivist learning - which views learning as the transmission of knowledge from the teacher to the learner - to constructivist learning which regards learning less as the product of transmission than a process of active construction. The second challenge comes from the new learning opportunities that Web technologies now afford. Because of its versatility and accessibility, the Web can help to shift the focus from knowledge-as-transmission to knowledge-as-construction. This paper describes a framework for applying the constructivist learning theory in the design of a Web-based courseware in the field of object-oriented software engineering.

1. Introduction

One of the most crucial prerequisites for successful implementation of Web-based instruction is the need for careful consideration of the underlying learning theory, and how learning takes place online. In practice, unfortunately, this is the most neglected aspect in any effort to implement Web-based instruction. However, as theories of learning have developed and teachers have gained more experience of using the WWW, there has been a shift from the objectivist learning model to a constructivist learning model. For many teachers, constructivism offers far more scope for realizing possible learning benefits of using Web-based technology. Similarly, because of its versatility and accessibility, the Web can help to shift the focus from objectivist learning to constructivist learning.

The remainder of this article is organized as follows. In section 2, we give an overview of the constructivist model of learning, followed by a description of current course Web site development practices and research in relation to constructivism. In section 3, we describe a set of instructional principles that guide the design of constructivist learning. In section 4, we outline the steps in designing a constructivist approach to object-oriented software engineering. Section 5 describes the software engineering course Web site. Then, in section 6, we present the results from the evaluation of the course. Finally, some remarks on further work conclude the article.

2. Constructivist Learning

A great deal of education builds on the objectivist model of learning which views learning as the passive transmission of knowledge from the teacher to the learner since its metaphor of mind is that of a blank page, a tabula rasa, waiting to be filled with knowledge. The objectivist model is criticized for stimulating surface learning, and knowledge reproduction. In stark contrast to the objectivist view, the constructivist perspective regards learning less as the product of passive transmission than a process of active construction. Learning is considered as an active process in which learners construct new knowledge based upon their prior knowledge. The constructivist perspective clearly diverges from the objectivist model of learning which presumes that knowledge can be put directly into the learner's head. Interested readers should consult [Von Glasersfeld 1994, Kafai & Resnick 1996, Phye 1997, Steffe & Gale 1995, Spivey 1997, Wilson 1998, Ben-Ari 1998] for a detailed description of the constructivist view of learning.

The constructivist model of learning described above suggests a set of instructional principles that can guide the practice of teaching and the design of learning environments [Hadjerrouit 1999, Hadjerrouit 1998b, Wilson 1998, Meeno-Seco & Forcada 1996, Duffy & Lowwyck & Jonassen 1993]. The design principles are as follows:

- Knowledge in a given domain must be actively constructed by learners, not passively transmitted by teachers.
- Students' prior knowledge must be taken into account by the construction of new knowledge.
- In order to be useful for problem solving, knowledge must be related to each other. The process of constructing interrelated knowledge structures requires higher-order thinking skills.
- Multiple representation of knowledge is crucial because problems cannot be solved in a simple way.
- To get students actively involved in knowledge construction, the learning activities should focus around a set of realistic, intrinsically motivating problems that are situated in some meaningful real world tasks. Rather than applying knowledge to solve abstract problems, knowledge must be constructed in real and useful contexts.
- Learning should take place in a learning environment that involves social interaction and negotiation through discussion and collaboration.
- Assessment procedures should be embedded in the learning process, and focus on authentic tasks and take into account learners' individual orientations.
- Teachers serve primarily as guides and facilitators of learning, not as transmitter of knowledge.

3. Constructivism and Current Course Web Site Development Practices and Research

The growing importance of constructivism in software engineering education has become

increasingly evident within the last five years. Discussions of current development practices within computer science and software engineering education can be found in [Bradly & Oliver 2002, Brahler & Peterson & Johnson 1999, Van Gorp & Grissom 2001, Krischner & Paas 2001, Marwell & Brooks 2002, Nulden 2001, Pullen 2001, Soendergaard & Gruba 2001, Fowler & Armagero & Allen 2001, McCormack & Jones 1998]. Basically, the introduction of constructivist ideas in software engineering education can be seen in the development of teaching and learning environments, as well as in pedagogical frameworks, but rarely in the construction of Web-based courseware. As a result, although software engineering education has been influenced by constructivism, most course Web sites are still static and rarely updated, diminishing the indisputable character of the Web, and, moreover, they do not reflect the dynamic character of learning. It appears that there is a lack of understanding and guidelines of how constructivist principles can be applied to the development of Web-based instruction. The identification of constructivist principles is indeed difficult, mostly because constructivism refers to learning and teaching molded by epistemological and ontological considerations associated with a constructivist philosophy with different orientations (radical constructivism, socio-cultural constructivism, etc.). Given the complexity of the constructivist philosophy, it may be difficult to construct an adequate pedagogical model rooted in constructivism. In addition, developing instructional Web sites rooted in the constructivist model of learning requires a sound scientific, software engineering and disciplined systematic approach as it undergoes the phases of analysis, design, implementation, testing, and maintenance [Lowe & Eklund 2002]. Furthermore it may be necessary to specify a set of quality attributes which must be addressed in the development process to obtain a quality product. Unfortunately, the construction of course Web sites has been ad hoc, resulting in poor quality, particularly in the early stages of the development process - the analysis and design stages [Lowe & Eklund 2002]. Clearly, researchers and teachers need to rethink their learning models and adopt a systematic, software engineering approach in the development of instructional Web sites. In addition, they need a strong understanding of the constructivist paradigm, that is a detailed and realistic description of its main principles, as well as a clear specification of constructivist characteristics and guidelines that provide a baseline for further refinement and decomposition in sub-characteristics in order to be useful for the development of course Web sites.

4. Steps in Designing a Constructivist Approach to Object-Oriented Software Engineering

Software engineering is taught during the fifth semester of the 3-year study programme in Computer Science at the Faculty of Mathematics and Sciences [Hadjerrouit 2002a, Hadjerrouit 2002b]. The primary objectives of the course were: (1) to allow students to familiarize themselves with the object-oriented software development methodology and the Unified Modelling Language (UML); (2) to gain enough practical experience through involvement in real-world projects, and finally (3) to acquire communicative, reading, and writing skills. The course was re-designed for the first time in the fall semester of 2002 according to the constructivist principles described above. The design process involved the following stages:

- a) Identify students' prior knowledge required.

- b) Identify the skills needed for constructing object-oriented knowledge.
- c) Describe realistic, real-world problems to be solved.
- d) Specify the collaborative model.
- e) Specify assessment procedures.
- f) Define the role of the teacher.

4.1. Students' Prior Knowledge

When running a course of any type, it is essential to start with a reasonably homogenous student group, especially in terms of prior knowledge. The course announcement should therefore include clearly defined prerequisite knowledge. This is consistent with the constructivist learning model which asserts that students' prior knowledge must be taken into account by the construction of new knowledge. In our case, third-year students are supposed to have sufficient background in the following subjects before taking the software engineering course. Firstly, Java and the key concepts of object-oriented programming, including HTML, JavaScript, and CGI scripts [Hadjerrouit 1998a]. Secondly, data structures, computer graphics, data communication, and advanced Java programming, including database development using the JDBC, SQL, and Java Servlets. These subjects are very important to lay a solid foundation for the development of object-oriented software.

4.2. Software Engineering Skills

The second step in designing a constructivist approach to object-oriented software engineering consists of identifying the skills needed by software engineers. We identified two groups of skills: The first group: analysis, design, reuse, programming, testing, and critical thinking skills provide the ability to perform key software engineering tasks during the software lifecycle. The second group: writing, reading, and communicating skills are essential for software engineers, in particular in a work situation. To summarize, software engineering skills are:

- Analysis skills, such as understanding and comprehending the problem domain, describing, refining, and representing problem situations using object-oriented concepts, as well as users requirements, system analysis, information gathering, simulation of scenarios.
- Design skills, such as architectural, component and logical design, cohesion and coupling issues, relationships between objects, elaboration of requirements, design criteria.
- Programming and testing skills such as implementation and program coding, testing, evaluating and debugging of the evolving code solution, and consistency checking.
- Reuse skills, such as comparing, contrasting, recognizing similarities and differences between new problem situations and previous solutions, including modifying, adapting, customizing, and reusing existing software design and program code solutions.
- Critical thinking skills, such as evaluating, explaining, and justifying software engineering solutions.
- Writing and reading skills, such as writing and formatting technical documentation and reading texts and documents.
- Communicative skills such as communicating with other learners and working in teams.

A constructivist approach to knowledge acquisition must take place in an environment that involves discussion and collaboration. Then, it must stress the crucial relationship between new knowledge and what is already known. Further, software construction must rely on motivating problems that are situated in some meaningful real-world tasks. Finally, some evaluation criteria are needed to reflect on the software solution process. To summarize, the acquisition of software engineering skills is a function of the following components :

- The prerequisite knowledge and technologies that the skill requires.
- The new knowledge to be learned in terms of the main steps of the process in which the skill is required.
- Previous software engineering solutions that can be used to solve the new software engineering problem
- The new software engineering problem in terms of a realistic, intrinsically interesting problem situation.
- Evaluation criteria needed to reflect on the software engineering solution process

Table 1 presents a sample skill specification.

<p><i>Description of the skill:</i> Developing use-case diagrams</p> <p><i>Required knowledge and technologies:</i> [1] Requirements analysis [2] Object-oriented modelling with UML [3] Rational Rose</p> <p><i>New knowledge and needed:</i> [1] Use-case modelling with UML [2] Assignments of requirements to actors and use-cases [3] Developing use-case diagrams [4] Documenting use-case with scenarios</p> <p><i>Previous use-case solutions and examples:</i> Link to previous use-case solutions and examples.</p> <p><i>Software engineering problem to be solved:</i> The software engineering problem to be solved consists of developing use-case diagrams in three steps. First, students are recommended to read carefully the requirements definition. Second, they must identify the main actors and use-cases. Finally, they must specify scenarios that describe the uses-cases and their actors.</p> <p><i>Evaluation criteria:</i> Readability, simplicity, and usability of use-case diagrams.</p>
--

Table 1. A sample skill specification

4.3. Project Work

The constructivist paradigm of learning recommends to focus on realistic, intrinsically motivating problems that are situated in some meaningful real-world tasks. Project work is particularly suited to implement this principle. Our projects were divided up into three stages, corresponding to the analysis, design and programming skills. Each group worked on a different project. The emphasis was on project management and software development process from analysis to the final product, with a set of deliverables and deadlines throughout the semester. Students were required to submit three written reports covering the analysis, design and implementation phases, and a final report including all phases. The teacher provided detailed written feedback. Reports were then rewritten based on the teacher's feedback, and submitted once again. Finally, each group had to present the final report orally to the whole class.

4.4. Collaboration

The constructivist paradigm of learning emphasizes social interaction and collaboration as a basis for knowledge construction. Collaboration is indeed essential for project work since acceptable project results can only be delivered if a group of students works in collaboration. Thus, the model we used for the organization of the projects was one in which our students were recommended to work in groups of 2-4 people, developing an object-oriented software system for solving a real-world problem. The organization of the projects in groups of 2-4 students allowed the supervision to be achieved through regular group meetings in collaboration with the teacher. Collaboration between students and the teacher was also supported by email.

4.5. Assessment

According to constructivism assessment procedures should be embedded in the learning process, focus on authentic tasks and take into account learners' individual orientations. To implement this principle we required a group and individual assessment. Group assessment covered project documentation, the final software product, and an oral presentation of the project. Students were required to submit three written reports covering the analysis, design and implementation phases, as well as a final report covering the entire project. Project work counted for 40 % of the final degree result. The individual assessment counted for 60 % of the final degree result, and was based on 3-hour written individual examination. The goal was to assess students' contribution to the projects, in particular how much they contributed to group work, as well as their understanding of the key topics of software engineering.

4.6. Teacher

The role of the teacher in the proposed constructivist learning environment was not to transmit knowledge but to help students to acquire software engineering skills. Thus, the teacher served primarily as guide and facilitator of knowledge. Basically, the teacher's work consisted of supporting the students in their search and supply of relevant information, coordinating the students'

presentations of individual milestones of their projects, moderating discussions, providing feedback and responses to queries, and communicating with students via email, etc. In addition, the teacher suggested project tasks in agreement with the students according to their interests and motivations.

5. Course Web Site

Instructional designers believe that the Web is particularly suited to be used with the constructivist learning environment [McCormack & Jones 1998, Nulden 2001]. This is because it can provide students with the capability of freely exploring material that is considered relevant for learning tasks. Our course Web site was designed to promote constructivist learning in three different ways. First, the course Web site was used as information source to provide useful knowledge for fulfilling project tasks. Students were therefore expected to check the course Web site frequently. Second, the course Web was used as a repository of students' project documents. Finally, the Web was used as communication medium to support collaborative work team. The course Web site contains the following components [Hadjerrouit 2002a, Hadjerrouit 2002b]:

- Online course notes which are organized in self-contained blocks according to the software engineering skills described above.
- Links to textbooks and other educational material
- Description of current projects
- Description of the recommended software engineering methodology
- Specimen of past students' projects with their solutions
- Link to related issues

6. Course Evaluation

A first version of the constructivist approach to e-Learning in software engineering was taught and evaluated in the autumn semester 2002. Because it was the first time such a course was being attempted it was not easy to identify all the elements that must be taken into account in the evaluation of the course compared with traditional courses. This is due to the collaborative character and greater flexibility of constructivist learning, along with the wider range of online resources offered to the students. Consequently, it is much harder to develop a complete methodology for evaluating our experience. Such an effort is therefore beyond the scope of this paper. Instead, we concentrate on the following aspects that are of particular significance from a constructivist point of view:

- Students' previous knowledge and eventually any previous experience in software engineering
- Team work and collaboration activities
- Real-world projects
- Project outcome and the ability to acquire software engineering skills

The evaluation was based on our observations, discussions with students, students' project performance, the written individual examination, and students' evaluation with questionnaires.

6.1. Prior Knowledge

Regarding the issue of prior knowledge we observed that background knowledge in particular from the object-oriented programming language Java, including Java Servlets in combination with the database MySQL, was necessary but not sufficient in order not to get lost in the new subject - software engineering. Prerequisite knowledge in programming was indeed very important to implement a valuable object-oriented software solution, but it was not sufficient to acquire analysis, design and reuse skills. It was therefore not a surprise that many students appeared to be more concerned about programming tasks rather than analysis, design and reuse skills. Nevertheless, the evaluation of the course showed that by identifying students' prior knowledge, their needs and motivations when designing a constructivist learning environment, it was easier to provide proper conditions of learning for each student and group of students. This observation is clearly compatible with the constructivist paradigm of learning which asserts that students construct new knowledge based upon their prior knowledge.

6.2. Team Work and Collaborative Activities

Helping students with different knowledge backgrounds, evaluating their project performance, providing immediate feedback, helping them to explore past students' solutions, acquiring software engineering skills to deal with the new projects demanded not only a lot of the teacher's and the students' time, but required also much interaction between students and the teacher. Clearly, it was a challenge to develop a collaborative work environment in order to help student teams learn how to work effectively and collaboratively. Thus, in the form presented here, collaborative work can be suggested only for rather small groups, say up to 20-25 students. Experiments with larger groups would be required to test this criterion.

In addition to the fact that it was a challenge to develop a collaborative work environment, development and management of skill specifications as described above and online course material on the Web prepared in accordance with the constructivist paradigm of learning were more complex and time intensive than conventional courses based on abstract and context-free problem solving.

Fortunately, students performed better when the teacher provided help and support. However, when not prompted by the teacher, students tended not to explore extra readings, past students' solutions, as well as online software engineering resources that were available to them.

6.3. Real-world Projects

An interesting observation regarding this issue was that students were very enthusiastic about the real-world character of the projects, even if they tended to spend significantly more time for their projects than in comparable, conventional course. The real-world project tasks clearly created a proper context for online discussion and conversation for many students. One group, however, reported difficulties working together in a group. The major complaints were members not working, and the problem of communicating. Encouraging and motivating students in that group to engage in conversation with their team members and other groups, was therefore not easy. Nevertheless, our

experience was 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

6.4. Project Outcome and the Ability to Acquire Software Engineering Skills

Regarding the issue of project performance, students performed better than before the course was redesigned. But, the evaluation has also shown that acquiring software engineering skills was hard for some students as it required a great effort and consumed a lot of time. The reason was that beginning students were usually not aware of the importance of software engineering skills. They focused on implementation issues rather than on analysis, design and reuse aspects. In many ways software engineering was regarded as an art rather than a discipline with principles methods, techniques and skills which guide the development of user-oriented, readable, modular, extensible, and reusable software. As a result, acquiring software engineering skills to deal with the new projects demanded a lot of the students' time.

7. Conclusions and Further Work

In this article, we presented a constructivist approach to Web-based instruction in object-oriented software engineering. Although we have only been able to outline the constructivist approach in broader terms, we are convinced that constructivism offers - by defining learning as an active process of construction - a potentially powerful way to rethink the pedagogical practice in software engineering education. Indeed, according to the evaluation results, students performed better in terms of software development process and software product quality than before the course was redesigned. Nevertheless, it was hard to implement the constructivist principles for at least two reasons. Firstly, it was time-consuming and more demanding in terms of communication, collaboration, as well as provision of intrinsically motivating learning activities that are situated in some meaningful real-world tasks. Secondly, it required Web technology to facilitate online learning, as well as collaborative work team, discussions and communication.

We have also experienced that developing a Web-based courseware rooted in the constructivist model of learning is clearly a challenge and requires a sound scientific, software engineering approach as it undergoes the phases of analysis, design and implementation, testing, and maintenance. Moreover it may be necessary to specify a set of quality attributes which must be addressed in the development process to obtain a quality product. Thus, our development approach, at this first phase, focused on building a Web site that will be usable, easily modified, and updated. We soon plan to develop and test a new prototype of the Web site with a variety of new course material. Then we will be able to draw some useful conclusions of the constructivist approach to Web-based instruction in the field of software engineering.

Summarizing, the constructivist view is clearly a challenge for instructional designers and teachers as it requires to make a radical shift in their thinking and to develop a rich environment to Web-based instruction that helps to translate the philosophy of constructivism into actual practice. Nevertheless, we believe that the constructivist learning model together with Web technologies offer

far more scope for realizing possible learning benefits than traditional instruction.

References

- Ben-Ari, M. (1998). Constructivism in Computer Science. *Proceedings of the 29th SIGCSE Technical Symposium on Computer Science Education*, 257-261.
- Bradly, C. & Oliver, M. (2002). Developing E-learning Courses for Work-based Learning. *Proceedings of the Eleventh International World Wide Web Conference*, Honolulu, Hawaii, USA 7-11. <http://www2002.org/CDROM/alternate/703/>
- Brahler, J. & Perterson, N. & Johnson, E. (1999). Developing On-line Learning Materials for Higher Education: An Overview of Current Issues. *Educational Technology & Society* 2, 1-8.
- Duffy, T. M. & Lowyck, J. & Jonassen, D. H. (1993). *Designing Environments for Constructive Learning*. Springer-Verlag.
- Fowler, L., & Armarego, J. & Allen, M. (2001). CASE- Tools: Constructivism and its Application to Learning and Usability of Software Engineering Tools. *Computer Science Education*, Vol. 11, No.3, 261-272.
- Hadjerrouit, S. (1998a). Teaching Java as First Programming Language: A Critical Evaluation. *SIGCSE Bulletin*, Vol. 30, No. 2, 43-47.
- Hadjerrouit, S. (1998b). A Constructivist Approach for Integrating the Java Paradigm into the Undergraduate Curriculum. *Proceedings of the 3th Annual Conference on ITiCSE*, 105-107.
- Hadjerrouit, S. (1999). A Constructivist Approach to Object-Oriented Design and Programming. *Proceedings of the 4th Annual Conference on ITiCSE*, 171-174.
- Hadjerrouit, S. (2002a). Software Engineering Course. http://fag.hia.no/kurs/inf2450/www_docs/
- Hadjerrouit, S. (2002b). Software Engineering Project. http://fag.hia.no/kurs/inf2470/www_docs/
- Kafai, Y. & Resnick, M. (Editors) (1996). *Constructionism in Practice: Designing, Thinking, and Learning in a Digital World*. Lawrence Erlbaum Associates.
- Kirschner, P. A. & Paas, F. (2001). Web-enhanced Higher Education: A Tower of Babel. *Computers in Human Behavior* 17, 237-353.
- Lowe, D. & Eklund, J. (2002). Client Needs and the Design Process in Web Projects. *Proceedings of the Eleventh International World Wide Web Conference*, Honolulu, Hawaii, USA 7-11. <http://www2002.org/CDROM/alternate/678/>
- Markwell J. & Brooks, D. (2002). Broken Links: The Ephemeral Nature of Educational WWW Hyperlinks. *Journal of Science Education and Technology* 11, 2, 105-108.
- McCormack, C. & Jones, D. (1998). *Building a Web-Based Education System*. Wiley

Computer Publishing.

Mereno-Seco F. & Forcada, M.L. (1996). Learning Compiler Design as Research Activity. *Computer Science Education* 7, 73-98.

Nulden, U. (2001). E-Education: Research and Practice. *Journal of Computer Assisted Learning* 17, 363-375.

Phye G. D. (Editor) (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*, Vol. 11, No. 3, 189-202.

Steffe, L. P. & Gale J. (Ed.) (1995). *Constructivism in Education*. Lawrence Erlbaum Associates.

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

Soendergaard, H. & Gruba, P. (2001). A Constructivist Approach to Communication Skills Instruction in Computer Science. *Computer Science Education*, Vol. 11, No. 3, 203-209.

Van Gorp, M.J & Grissom, S. (2001). An Empirical Evaluation of Using Constructive Classroom Activities to Teach Introductory Programming. *Computer Science Education*, Vol. 11, No. 3, 247-260

Von Glasersfeld, E. (1994). *Radical Constructivism in Mathematics Education*. Kluwer Academic Publishers

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