Introducing Critical Thinking to Software Engineering Education

Article *in* Studies in Computational Intelligence · January 2014

DOI: 10.1007/978-3-319-00948-3-12

2 authors:

Oumout Chouseinoglou
Bilkent University
25 PUBLICATIONS **142** CITATIONS
SEE PROFILE

Semih Bilgen
Istanbul Okan University
93 PUBLICATIONS **449** CITATIONS
SEE PROFILE

Introducing Critical Thinking to Software Engineering Education

Oumout Chouseinoglou and Semih Bilgen

Abstract. Software and its development processes are changing continuously pervading our daily life, new and diverse techniques and approaches are being proposed and the software industry is eager to adopt the ones that will provide competitive advantage. The diversity of these new techniques and approaches and the diversity of clients and contexts in the software industry, requires software developers to have the ability to judge correctly and to discriminate successfully among these. These skills need to be taught to software developers in the course of their formal undergraduate education. However, traditional approaches in software engineering education (SEEd) are mostly inadequate in equipping students with these unusual and diverse skills. This study, as part of a larger study aiming to develop a model for assessing organizational learning capabilities of software development organizations and teams, proposes and implements a novel educational approach to SEEd combining different methodologies, namely lecturing, project development and critical thinking. The theoretical background and studies on each approach employed in this study are provided, together with the rationales of applying them in SEEd. Student opinions and instructor observations demonstrate that the proposed course structure is a positive step towards the aforementioned goals.

Keywords: Software engineering education, critical thinking, practicum, SQ4R.

1 Introduction

Software systems evade our daily life in an increasing pace with diverse applications and the need for quality software is eminent. Different methodologies and

Oumout Chouseinoglou
Statistics and Computer Science Department, Başkent University, 06810, Ankara, Turkey
e-mail: umuth@baskent.edu.tr

Semih Bilgen
Electrical and Electronics Engineering Department, Middle East Technical University, 06531, Ankara, Turkey
e-mail: semih-bilgen@metu.edu.tr

R. Lee (Ed.): *SERA*, SCI 496, pp. 183–195.
DOI: 10.1007/978-3-319-00948-3_12

approaches in software engineering are being employed to provide the necessary levels of software quality, however, the quality of developed software is directly related to the supply of capable and up-to-date software developers [1], who have to cope with technical as well as non-technical issues [2], and who have to discriminate among the criteria for success by identifying the good solutions for the problem at hand [3]. In other words todays software developers need to "think out of the box" [3]. On the other hand, software engineers, in accordance with the characteristics of the engineering domain, are expected to reconcile conflicting constraints and to make deliberate selections among alternative designs with their judgments based on deep knowledge of the discipline [1], need to have social skills, and must be capable of evaluating competing values [4]. Thus, our belief is that software developers and engineers need to be equipped with the aforementioned skills and capabilities through their education with the use of specifically constructed software courses.

However, traditional and generic approaches in software engineering education (SEEd) are insufficient in helping students to keep their knowledge current, cannot prepare them for the intricacies of the domain and fail to produce the supply and quality of developers required by the industry [1][5]. Traditional learning approaches have been proven to be ineffective as students are not actively involved in the learning process but are merely passive listeners [6]. SEEd should first identify the critical ingredients that result in competence in the field and then the instructional models that will transform students to effective practitioners should be developed [7]. Moreover, new approaches should not only require students to study theory using text books but also should educate them to "learn how to learn" through state-of-the-art analyses [8], allow them to stay up-to-date regardless of rapid change, prepare them for different and new roles [1], and provide them with the experience of non-technical issues and practical know-how [2]. Similarly, in a detailed review of current trends in SEEd [6] the importance of self-directed learning (the ability to learn on their own) and higher order cognitive skills of application, analysis, evaluation and synthesis is emphasized. Surveying the current trends in SEEd within the pedagogical context, the authors of [9] point to the increasing importance of practice-based education and alternative ways of teaching such as empirical methods where students will acquire knowledge for evaluating and proposing technology and processes. SEEd is moving from lecture-format courses to team projects where students are expected to exercise the ideas they are learning [1][6], the so called practicum, a positive change bridging the academia-industry gap [6]. The efforts of transforming the existing knowledge to a curriculum have produced two important milestones for the SEEd, namely the Guide to the Software Engineering Body of Knowledge, which manifests the general perceptions on what a software engineer with bachelor's degree and four years of experience should know, and the Software Engineering 2004, which suggests curriculum guidelines for undergraduate software engineering degree programs [4]. However, experiences in project based SEEd have shown that students with little industrial involvement (i.e., undergraduate students) aren't mature enough to appreciate the importance of many software engineering topics [4], they lack the intuition to understand problems, ambiguity and hidden constraints of real projects and therefore find it challenging to apply

their acquired knowledge to a project [3], and focus more on programming issues and less on the development process and the associated software engineering issues [6]. Therefore, students often fail to appreciate the importance of tasks that software practitioners continuously conduct (e.g., requirements engineering, project management, cost estimation) [4], as they consider them theoretical and of very little use in future [6].

In [7], SEEd courses are investigated with respect to three distinctive knowledge categories, namely declarative (knowledge from textbooks), procedural (knowledge by doing) and metacognitive knowledge (planning, monitoring process and progress, changing when appropriate and reflecting). Software domain studies make it clear that each knowledge category must be addressed explicitly in instruction. This paper describes the practice of combining the approach of critical thinking, a metacognitive approach, to a project based SEEd course (traditionally encompassing the declarative and procedural approaches) with the aim of increasing the understanding capabilities of students with respect to software engineering practices, allowing them to appreciate these practices and equipping them with skills to evaluate and judge alternative approaches that they will face in their professional careers. The rest of the paper is organized as follows: First, we briefly review the related practices and the theoretical background on the subject. Section 3 gives the details of the developed course as a comparative study, and Section 4 outlines the lessons learned. The last section concludes the paper and overviews the planned future work.

2 Related Work

2.1 Software Engineering Practicum

Acknowledging that the skills to be effective software engineers are not limited to declarative knowledge and claiming that expertise is domain-specific and can only be acquired in the context in which it will be practiced, practicum is a procedural approach which uses a realistic environment, usually in the form of a project for an actual client, where students learn the skills they will use in the future by applying their knowledge in a real-world setting [7][10][11]. Especially it is important that these practicum approaches be team based, according to the CMM statement that in higher levels of maturity individual activities transform to team activities [12]. Practicum is closely related to constructivism, a learning theory that is learner centered, which states that students learn better if they construct knowledge for themselves and regards learning as a process of active construction [13]. The set of constructivist instructional principles and the skills needed by software engineers to solve real-world problems within the constructivist approach are given in detail in [13].

Numerous successful cases of SEEd courses incorporating theoretical knowledge and practical experience with the use of semester-long projects have been discussed in the literature. Surveys and descriptions of courses reflecting the realities and

focusing on specific areas of software engineering such as requirements engineering, supply chain development and global software development are presented in [14] and [15]. In [16] it is argued that the proposed approach taught inexperienced graduate students many software engineering principles, such as software verification and validation. An initiative towards restructuring an undergraduate software engineering class from lecture-based to lab-oriented by focusing on learning and personality types and emphasizing practical tools is given in [17]. A review of textbooks addressing the difficulties of learning by doing in the SEEd domain and the challenges faced by the universities is available in [2], and a university-wide example of learner centered approach by solving real problems is provided in [18]. In [13], the author surveys a vast number of approaches employed to render software education more realistic, pinpointing the most important ones that have provided valuable contributions. However, it is argued that the majority of these approaches do not explicitly integrate pedagogical innovations, they do not sufficiently take into consideration the human aspect of the learning process such as the emotions, behavior and thoughts of students and finally they do not expose students to all phases of software development [13]. Despite these successful cases in the literature, the practicum approach has several drawbacks, pointed out in detail in [7]: instructors use little effort on identifying the skills that a project should bring to the students, practicum does not help students to develop the skills necessary for deliberate practice and students mostly are able to apply what they have learned only to very similar situations [7]. In [11], observed issues with practicum are poor testing, ineffective teams, no documentation, no use of metrics, no measures of quality and failure of students to transfer knowledge from the formal curriculum to the practicum project.

Another successful practicum in SEEd is the CSCI577ab Software Engineering course [19][20] which is further described in detail in Section 3 as it has been pivotal in our study.

2.2 Personality and Learning Types

In [17] it is discussed that students can be classified with respect to their personality types using the Myers-Briggs type indicator and to their learning styles using Felder-Silverman model and a software engineering course should be constructed in a way that should appeal to most students. In Myers-Briggs personality types the students can be characterized with respect to four dimensions, namely as introvert vs. extrovert, sensing vs. intuitive, thinking vs. feeling and judging vs. perceiving. On the other hand, with respect to the Felder-Silverman learning styles, the students can be classified as active vs. reflective, sensing vs. intuitive, visual vs. verbal and sequential vs. global. The details of how these dimensions should be assessed are given in detail in [17]. However, it is evident that SEEd courses need to be designed in a fashion to contain elements that would address the majority of students. We believe that critical thinking is an approach that can address a wide number of students with different personality and learning types.

2.3 Critical Thinking in Education

The importance of judgment and decision making in engineering is studied in detail in [3], where it is clearly shown that methods, tools, processes, skills, heuristics, and other tools and techniques can be utilized in the search of good and cost-effective candidate solutions but cannot replace judgment. According to the engineering principle of striking a balance between conflicting goals, SEEd students should be taught judgment and the commitment to use it [3]. This, we believe, is closely related to critical thinking, a metacognitive approach. A survey of the conceptions of critical thinking is given in [21] where it is defined as "reasonable, reflective thinking focusing on task, people or belief"; involving abilities such as "identifying a problem and its associated assumptions, clarifying and focusing the problem, and analyzing, understanding and making use of inferences, inductive and deductive logic, as well as judging the validity and reliability of the assumptions, sources of data or information available". In [7] it is stated that when students are asked to think in a metacognitive fashion at every stage of a problem-solving process, not only they accomplish this task but they also develop a deeper understanding about it and achieve better performance on following problems. Moreover, studies in programming domain have demonstrated that students perform better both on declarative and procedural tasks when they reflect on what they are learning. This suggests that the metacognitive activity is the main reason for producing the improved performance [7].

In [3], a critical thinking approach for SEEd is presented where supplemental materials (mostly books) are incorporated to the course by having the groups read and report on the ideas from these materials via critical analysis and interpretation, and having the students to identify their association with the course. According to the authors [3] this activity highlights the value of engineering judgment by emphasizing critical evaluation and helps students learn to recognize external but relevant material, evaluate techniques on their own, and recognize how to use different techniques together. The course moves from the traditional memorize-and-recite-back to the critical application of content. The related theoretical framework and the application details of this approach are provided in detail in [3]. In [7] extreme programming practices are assessed within the metacognitive approach by focusing on how each practice may facilitate the acquisition of the metacognitive skills that are required for the development of enhanced competence in students.

In this study the Survey, Question, Read, Recite, Review, and wRite (SQ4R) technique is proposed as a critical thinking method. Details of SQ4R and how it was implemented are discussed in Section 3.

3 Comparative Study

3.1 General Structure of the Course

The İST478 Current Topics in Information Technologies course, which is offered in the Department of Statistics and Computer Science, Başkent University, Turkey,

is a course with flexible content and focuses on addressing current topics in information systems, software engineering and programming. In the 2011-2012 Spring term in which this study was conducted, İST478 was given as a Software Engineering Team Project Practicum course, as explained below, and was developed with the initial aims of: (a) to teach students the practical techniques and tools that are used in professional software development through regular lecture sessions and a practicum conducted in teams, and (b) to provide a test bed and a pilot study to validate whether a model for assessing the organizational learning capabilities of software development teams, namely AiOLoS (Assessing Organizational Learning of Software Development Organizations), developed by the authors [23], is actually applicable in real life teams. AiOLoS has been developed with the main aims of (a) providing a framework for comparison between software organizations with respect to their organizational learning capabilities, (b) allowing software organizations to identify their deficiencies and shortcomings, (c) offering the means for the measurement of the realized improvement in organizational learning and (d) providing a starting point for software process improvement. AiOLoS consists of three major process areas that map to the three major objectives of an Learning Software Organization [22], namely obtaining, using and passing knowledge. AiOLoS proposes that the organizational learning activity can be assessed with respect to 12 core processes that elaborate the 3 major process areas. The details of AiOLoS are available in [23] whereas an exploratory case study of AiOLoS conducted on the İST478 course is given in [24]. As explained in [9], when pilot studies such this one are carried out with students, they are required to have pedagogical value and both the researchers and the students have to perceive that value. Therefore, in order to enhance the pedagogical value of the course a critical thinking approach within the perspective of metacognitive knowledge was employed. However, as this critical thinking approach is also novel and was developed based on different practices in the literature not previously applied in the SEEd domain in conjunction, a comparative study was performed in order to assess its applicability and usefulness.

The subjects consisted of 15 undergraduate and 4 graduate level students who were enrolled in the İST478 course. All graduate level students and 6 of the 15 undergraduate level students had taken an introductory software engineering course. Four software development groups were formed of varying sizes, with each graduate student being assigned as a team leader (project manager) to each group. Moreover, each group had at least two students who had previously taken an introductory software engineering course. In order to achieve fairness in the workload, each group was assigned the development of systems similar in size and context, but with significant requirement and development differences. Specifically, each group was assigned the development of a score tracking software respectively for chess, tennis, basketball and football.

The course followed a customization of the outline provided by CSCI577ab Software Engineering [19], a graduate software engineering course at University of Southern California, being offered since 1996. CSCI577ab focuses on software plans, processes, requirements, architectures, risk analysis, feasibility analysis,

software product creation, integration, test, and maintenance with an emphasis on quality software production [20]. Moreover, CSCI577ab has been used as an experimental test-bed to deploy various research tools and approaches for validation of new methods and tools, leading to twelve PhD dissertations until 2008. As stated in [16], partially employing an already defined course outline and building the novel approaches of our study on top of that outline, ensures that our study is in accordance with published, well-grounded work and may encourage other instructors to use and employ the methodology that is proposed in this research with less effort. İST478 followed the Incremental Commitment Spiral Model (ICSM) [25][26][27], a new generation process model developed specifically for CSCI577ab and the architected agile approach for software development. İST478 covered the full system development life cycle of ICSM, which consisted of the Exploration phase, Valuation phase, Foundations phase, Development phase, and Operation phase. The deliverable deadlines and the items to be delivered for each of these phases were predefined. The tasks and artifacts to be developed by the students in İST478 were based on specific templates and they were described in detail in the Incremental Commitment Spiral process model - Electronic Process Guide (ICSM-EPG) [28]. Table 1 provides the list of conducted phases and the artifacts delivered by groups in each phase.

3.2 Implementing the SQ4R Approach

SQ4R [29] is a metacognitive approach to facilitate students' comprehension and memory specifically when reading science texts. Moreover, SQ4R has been implemented successfully in courses from a variety of areas, such as poetry analysis [30] or arts teaching [31]. SQ4R teaches learners to "attack" content in five sequential steps [32]: (i) Survey and (ii) Question where self-questioning and predicting occurs, (iii) Reading where learners check if their predictions are accurate, (iv) Recording where learners take notes regarding the content, Reciting where learners fill the gaps in their understanding based on their notes and finally (v) wRiting where learners write a brief summary to reflect what they have understood from the subject [33].

Fig. 1 The implemented SQ4R approach

Table 1 The ICSM phases followed in this study

Phase	Deliverable	Artifact
Exploration	Customer Interaction Package	Customer Interaction Report
Valuation	Valuation Commitment Package	Customer Interaction Package + Life Cycle Plan Operational Concept Description Feasibility Evidence Description
Foundation	Foundation Commitment Package	Valuation Commitment Package + System and Software Architecture Description System and Software Requirements Description Prototype Report Supporting Information Document
Development	Development Commitment Package	Foundation Commitment Package + Quality Management Plan Acceptance Test Plan and Cases Iteration Plan
Transition	Transition Readiness Package	Development Commitment Package + Iteration Assessment Report Training Plan User Manual Transition Plan Test Procedures and Results Functioning Product

Two randomly selected groups (groups 2 and 3) were assigned a differentiated development method of the ICSM which incorporated the SQ4R, to enhance their learning experience. The two groups implementing SQ4R were provided with prior knowledge of the phase they were conducting, the artifacts they were expected to develop and the deliverables to submit. During SQ4R, before working on and developing the deliverable, the students were given the deliverable name and were asked to conduct a small "survey" on the subject. After the survey, the team members were asked to write a brief reflection paper where they "questioned" and discussed why they thought the phase and the related deliverables are of importance for the software development process. Then all teams were given the guidelines and templates of the deliverables to be developed. The teams, while developing the deliverables, "read" the documents provided by the instructor and team members would "recite" to each other what they have understood on the material provided by the instructor. After the submission of the deliverable, the members of the teams undertaking SQ4R would conduct a "review" session with the instructor where they discussed their understanding of the process they have concluded/undertaken and the deliverable they have submitted. Finally they would write a closure paper, where they discussed what they have done, if they have understood it, what their initial thoughts and final thoughts were on the process, if they would change some or all parts of the deliverable or process, and their final comments/proposals. Figure 1 displays the SQ4R approach which was undertaken by the two randomly assigned groups in all five phases (depicted as "Milestone" in Fig. 1) of the software development lifecycle of İST478 course.

Similar to the experience in [34], during the course period one of the groups submitted no acceptable documents and deliverables (Group 4) and subsequently the members failed the course; thus no metrics or data were collected from this group. The authors in [34] argue that this experience is one of the most important lessons learned while conducting experiments in SEEd courses: surprises happen, and evaluations rarely turn out exactly as planned.

Among these three groups, only Group 1 did not undertake the SQ4R approach. The results of the AiOLoS research regarding the organizational learning capabilities of the assessed three teams are given in [24].

4 Lessons Learned / Experience and Evaluation

As explained in detail in [34], evaluation in the domain of education and especially SEEd is a challenging undertaking, as it is almost impossible to adequately isolate the effects of a new or proposed educational technique, there are difficulties in getting a statistically significant number of subjects, assessment of software engineering skills is less straightforward with respect to other disciplines, and comparative evaluations are difficult to be conducted due to the immaturity of the domain of software engineering. Due to these reasons in SEEd a new technique is usually intended to be a supplement to a curriculum.

The developed İST478 course was aimed to be as close as possible to reality and all five phases of the ICSM model were completed in a period of 16 weeks. All three groups that attended the course completed and submitted a working software artifact. However, as the developed software products were not meant for real-life usage, no payments or similar incentives existed; the incentives of students for developing the projects according to the expectations of the ICSM-EPG and SQ4R were credit points. Students also received credit points for attending the lectures.

After the conclusion of the course, the teams undertaking the SQ4R approach, a total of 11 students (both undergraduate and graduate), were asked to evaluate and assess the SQ4R approach and provide their opinions regarding the model and its results. The team members were asked five questions regarding the developed SQ4R model and they submitted their results using a Likert Scale. The questions and the Likert scores of the answers are given in Table 2. The major threat to the validity of that evaluation was the instructor-student relationship that existed between the assessor and the assessed team members. This relationship could force the students to alter their answers in the questionnaires to more favorable ones, believing that such answers would contribute to their grades. In order to resolve this, the students were informed that they would not be graded based on the answers they provide. Moreover, the survey answers were collected after the submission of the grades, so that students would not feel compelled to provide answers that do not depict their true opinions about the SQ4R model.

The frequency of the results regarding the answers given in the opinion questionnaires were:

- 8 out of 11 believed that the SQ4R approach mostly helped them to learn the course topics better (mode value being Mostly, median value being 4 out of 5),
- 6 out of 11 believed that the SQ4R approach mostly helped them to apply the course topics better to their project (mode value being Mostly, median value being 4 out of 5),
- 9 out of 11 believed that the SQ4R approach mostly provided them with an advantage in the development of the project (mode value being Mostly, median value being 4 out of 5),
- 6 out of 11 believed that the time spent for conducting the SQ4R approach mostly was worth it (mode value being Mostly, median value being 4 out of 5),
- 6 out of 11 believed that the SQ4R approach mostly would contribute to passing the acquired knowledge to their later professional life (mode value being Mostly, median value being 4 out of 5).

Moreover, the students felt that the SQ4R experience enhanced their academic curiosity by the questions they were asking and the level of their participation in classroom activities. Even though it was not measured with questionnaires, another important observation was that the attitude of team members towards the task at hand would usually shift positively after the conclusion of the SQ4R for that task. On the other hand, the majority of the students would complain regarding the extra work the SQ4R required. However, we believe that these complaints were mostly related to the fact that one development team was not undertaking the SQ4R approach.

Table 2 Student opinions regarding SQ4R approach

Question	Fully	Mostly	Somewhat	Very Little	Not at all
Q1) Do you think the SQ4R helped you to learn the course topics better?	2	8		1	
Q2) Do you think the SQ4R approach helped you to apply the topics better to your project?	3	6	2		
Q3) Do you think the SQ4R approach provided you an advantage in the development of your project?	1	9		1	
Q4) Do you think the extra time spent for conducting the SQ4R was worth it?		6	2	3	
Q5) Do you think the SQ4R approach will contribute to you passing the acquired knowledge to your later professional life?	4	6	1		

5 Conclusion

This study has been a part of a larger study aiming to develop the AiOLoS model for assessing organizational learning capabilities of software development organizations and teams. Within that perspective, a pilot study was constructed to test

whether AiOLoS can be actually implemented in software development teams. In conjunction, in order to propose a pedagogical contribution, a novel SEEd course structure has been introduced, implementing several different methodologies, namely lecturing, practicum with ICSM and critical thinking.

It is obvious that the number of subjects who participated in the comparative study described in this paper and the overall structure of the study are not conducive to statistically significant and definitive results, especially as one of the development teams did not continue the course. As confirmed in [34], the anecdotal usage alone of an education technique does not provide much information, instead a multi-angled evaluation approach which is partially rooted in educational theory, can be a useful solution.

Nevertheless, we believe that the combination of lecture and practicum, the use of a well-grounded software process model (ICSM), and the implementation of a critical thinking based learning approach to the process has contributed to the development of a successful learning environment. It is our opinion that, with the aforementioned amalgamation of approaches and methods, the course has (a) constituted a step forward towards the realization of an overall SEEd course that will arm students with the critical thinking and judgment capabilities the engineering approach requires, and (b) appealed to a wide variety of students with different personality types and learning styles. The student opinions given in Table 2 and the observations of the instructor are supplements of these claims. We have interpreted these results as an indication that the proposed approach of utilizing creative thinking in SEEd provides students with both the knowledge and the judgment capabilities that the software engineering industry requires.

We are currently analyzing the obtained results and the lessons learned from that first experience with this course in order to develop a more structured and well-defined course, that will be used to better evaluate and compare the educational attainments of the proposed model. It is our belief that by harvesting further data from students in the newly constructed course we will have a better understanding of the learning needs of SEEd students and we will be able to find solutions in filling the gap between the requirements of software industry and SEEd. Furthermore, we are also investigating the case of proposing the critical thinking approach to professional software development organizations as a technique of increasing the organizational learning of teams and individuals and enhancing AiOLoS in order to embody the critical thinking capabilities of software practitioners.

References

1. Shaw, M.: Software engineering education: a roadmap. In: Proceedings of the Conference on the Future of Software Engineering. ACM (2000)
2. Gnatz, M., Kof, L., Prilmeier, F., Seifert, T.: A practical approach of teaching software engineering. In: Proceedings of the 16th Conference on Software Engineering Education and Training, CSEE&T 2003, pp. 120–128. IEEE (2003)

3. Shaw, M., Herbsleb, J., Ozkaya, I., Root, D.: Deciding what to design: Closing a gap in software engineering education. In: Inverardi, P., Jazayeri, M. (eds.) ICSE 2005. LNCS, vol. 4309, pp. 28–58. Springer, Heidelberg (2006)
4. Van Vliet, H.: Reflections on software engineering education. IEEE Software 23(3), 55–61 (2006)
5. Blake, B.M.: A student-enacted simulation approach to software engineering education. IEEE Transactions on Education 46(1), 124–132 (2003)
6. Garg, K., Varma, V.: A study of the effectiveness of case study approach in software engineering education. In: Proceedings of the 20th Conference on Software Engineering Education & Training, CSEET 2007. IEEE (2007)
7. Williams, L., Upchurch, R.: Extreme programming for software engineering education? In: The Proceedings of the 31st Annual Frontiers in Education Conference. IEEE (2001)
8. Boehm, B.: A view of 20th and 21st century software engineering. In: Proceedings of the 28th International Conference on Software Engineering. ACM (2006)
9. Carver, J., Jaccheri, L., Morasca, S., Shull, F.: Issues in using students in empirical studies in software engineering education. In: Proceedings of the Ninth International Software Metrics Symposium. IEEE (2003)
10. Katz, E.P.: Software engineering practicum course experience. In: Proceedings of the 23rd IEEE Conference on Software Engineering Education and Training (CSEE&T). IEEE (2010)
11. Bareiss, R., Katz, E.P.: An exploration of knowledge and skills transfer from a formal software engineering curriculum to a capstone practicum project. In: Proceedings of the 24th IEEE-CS Conference on Software Engineering Education and Training (CSEE&T). IEEE (2011)
12. Favela, J., Feniosky, P.M.: An experience in collaborative software engineering education. IEEE Software 18(2), 47–53 (2001)
13. Hadjerrouit, S.: Learner-centered web-based instruction in software engineering. IEEE Transactions on Education 48(1), 99–104 (2005)
14. Gotel, O., Kulkarni, V., Neak, L.C., Scharff, C., Seng, S.: Introducing global supply chains into software engineering education. In: Meyer, B., Joseph, M. (eds.) SEAFOOD 2007. LNCS, vol. 4716, pp. 44–58. Springer, Heidelberg (2007)
15. Gotel, O., Kulkarni, V., Say, M., Scharff, C., Sunetnanta, T.: A global and competition-based model for fostering technical and soft skills in software engineering education. In: Proceedings of the 22nd Conference on Software Engineering Education and Training (CSEE&T 2009). IEEE (2009)
16. Hayes, J.H.: Energizing software engineering education through real-world projects as experimental studies. In: Proceedings of the 15th Conference on Software Engineering Education and Training (CSEE&T 2002). IEEE (2002)
17. Layman, L., Cornwell, T., Williams, L.: Personality types, learning styles, and an agile approach to software engineering education. ACM SIGCSE Bulletin 38(1), 428–432 (2006)
18. Nikolov, R., Ilieva, S.: Building a research university ecosystem: the case of software engineering education at Sofia University. In: Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering. ACM, Cavtat near Dubrovnik (2007)
19. Boehm, B., Koolmanojwong, S.: Software Engineering I - Fall 2011. USC Viterbi School of Engineering (August 12, 2011), `http://greenbay.usc.edu/csci577/fall2011/index.php` (cited June 30, 2012)

20. Koolmanojwong, S., Boehm, B.: Using Software Project Courses to Integrate Education and Research: An Experience Report. In: Proceedings of the 22nd Conference on Software Engineering Education and Training (CSEE&T 2009), Hyderabad, India (2009)
21. Pithers, R., Soden, R.: Critical thinking in education: A review. Educational Research 42(3), 237–249 (2000)
22. Ruhe, G.: Learning Software Organisations. In: Chang, S.K. (ed.) Handbook of Software Engineering and Knowledge Engineering, vol. 1, pp. 663–678. World Scientific Publishing (2001)
23. Chouseinoglou, O., Bilgen, S.: Assessing Organizational Learning in Software Development Organizations. Technical Report. METU/II-TR-2012-02, Department of Information Systems, Middle East Technical University, Ankara, Turkey (2012), http://www.baskent.edu.tr/~umuth/METU-II-TR-2012-02.pdf
24. Chouseinoglou, O., Bilgen, S.: A Model for Assessing Organizational Learning in Software Development Organizations. In: Winckler, M., Forbrig, P., Bernhaupt, R. (eds.) HCSE 2012. LNCS, vol. 7623, pp. 251–258. Springer, Heidelberg (2012)
25. Boehm, B., Lane, J.A.: Using the Incremental Commitment Model to Integrate System Acquisition, Systems Engineering, and Software Engineering. CrossTalk the Journal of Defense Software Engineering, 4–9 (October 2007)
26. Pew, R.W., Mavor, A.S.: Human-System Integration in the System Development Process: A New Look. National Academy Press (2007)
27. Boehm, B.: Some future software engineering opportunities and challenges. In: The Future of Software Engineering, pp. 1–32. Springer, Heidelberg (2011)
28. USC-CSSE: Instructional Commitment Spiral Model - Software Electronic Process Guide. USC Viterbi School of Engineering (2008), http://greenbay.usc.edu/IICMSw/index.htm (cited June 30, 2012)
29. Thomas, E.L., Robinson, A.H.: Improving Reading in Every Class: A Sourcebook for Teachers. Allyn & Bacon, Boston (1982)
30. Casebeer, E.F.: SQ4R in the Analysis of Poetry. College Composition and Communication 19(3), 231–235 (1968)
31. Applegate, M.D., Quinn, K.B., Applegate, A.J.: Using metacognitive strategies to enhance achievement for at-risk liberal arts college students. Journal of Reading 38(1), 32–40 (1994)
32. Glynn, S.M., Muth, D.K.: Reading and writing to learn science: achieving scientific literacy. Journal of Research in Science Teaching 31(9), 1057–1073 (1994)
33. Yakupoglu, F.: The effects of cognitive and metacognitive strategy training on the reading performance of Turkish students. Practice and Theory in Systems of Education 7(3), 353–358 (2012)
34. Navarro, E.O., Van Der Hoek, A.: Comprehensive evaluation of an educational software engineering simulation environment. In: Proceedings of the 20th Conference on Software Engineering Education and Training, CSEET 2007. IEEE (2007)