

Teaching a Software Development Methodology: The Case of Extreme Programming

Orit Hazzan

Department of Education in Technology and Science
oritha@tx.technion.ac.il

Yael Dubinsky

Department of Computer Science
yael@cs.technion.ac.il

Technion – Israel Institute of Technology
Technion City, Haifa 32000, Israel

Abstract

This paper focuses on the teaching of software development methodologies. It presents ten principles of teaching such a topic, while examining each from both a pedagogical and an organizational viewpoint. The teaching principles are demonstrated using the methodology of Extreme Programming (XP).

1. Introduction

Calls have been made recently to integrate software development skills into the curriculum of computer science and software engineering programs (Denning, 1992; Meyer, 2001). Parallel to this trend, the importance of the product (software) process is being highlighted, and topics such as productivity and quality are gaining increasingly more attention, especially as a result of recent financial circumstances¹. One way to group the aforementioned topics is under the heading of Software Development Methodologies (hereinafter abbreviated as SDM or SDMs). This paper focuses on teaching SDMs to undergraduate students. The paper presents ten principles for teaching an SDM and demonstrates them using the teaching of Extreme Programming (XP), an SDM that has been receiving increasingly more attention in industrial and academic discussions in recent years.

Section 2 describes the research background from which the teaching principles, described in section 3, have been derived. We conclude in Section 4.

2. Research background

The principles presented in Section 3 are based on an analysis of the teaching of the XP methodology in a course entitled ‘Operating Systems Projects’. This course has been taught for the past six years at the Technion’s Department of Computer Science by the second author of this paper. The Department of Computer Science is considered to be comparable to the ten leading Computer Science departments in the US. Approximately 1,300 undergraduate students and about 200 graduate students are currently studying at the Department. In addition to its general B.Sc. program, the Department offers 4 special undergraduate tracks in Software Engineering, Information Systems Engineering, Computer Engineering, and Bioinformatics. The Technion in general, and the Department of Computer Science in particular, are one the main suppliers of (software) engineers to the Israeli hi-tech industry.

Each year, approximately eighty students attend the ‘Operating Systems Projects’ course offered in the Winter and Spring semesters, and fifty students take the course in the Summer semester. As a computer-science project-based course (Sebern, 2002; Wilson 2001), the main requirement of the course is the development of an operating systems project. Prior to the introduction of XP into the course, projects were performed in teams of 2-3 computer/software engineering students. The students were required to submit documentation, mid-semester and final presentations and to participate in a project fair. The academic staff consisted of two instructors, who coordinated the teaching and administration of the course, and of several supervisors. If questions or other issues arose, the students could meet with the supervisors during their regular office hours.

¹ Look for example at the National Institute of Standards and Technology (NIST) New Release of June 28, 2002 at http://www.nist.gov/public_affairs/releases/n02-10.htm.

Starting in the 2002 Summer semester, XP has been introduced as the SDM used in the ‘Operating Systems Projects’ course. XP is one SDM of the family of agile SDMs, which includes methodologies such as SCRUM (Schwaber and Beedle, 2001) and the DSDM (Dynamic System Development Method²). Flower (2002) explains that “agile methods have some significant changes in emphasis from heavyweight methods.” Specifically, he mentions two major differences:

- *Agile methods are adaptive rather than predictive.* Heavy methods tend to try to plan out a large part of the software process in great detail for a long span of time. This method works well until things change. Therefore, the nature of such methods is to resist change. The agile methods, however, welcome change. They try to be processes that adapt and thrive on change, even to the point of changing themselves.
- *Agile methods are people-oriented rather than process-oriented.* Agile methods explicitly make a point of trying to work with peoples' nature rather than against them and to emphasize that software development should be an enjoyable activity.

XP, in specific, is described by Beck (2000) as being “a lightweight, efficient, low-risk, flexible, predictable, scientific, and fun way to develop software” (p. XVII).

In addition to the introduction of XP into the ‘Operating Systems Projects’ course, some studio elements were also introduced into the course. The studio is the basic learning method used in architecture schools and it has been adopted by some other design disciplines as a training environment. In such studios, students develop projects with close guidance of a tutor and with on-going reflection, both on what is created and on the creation process. Studio descriptions in Software Engineering education can be found in Tomayko (1996) and in Kuhn (1998). Analysis of how the studio can be implemented into software development environments is presented by Hazzan (in press, 2002). Integration of the studio with XP, in the ‘Operating Systems Projects’ course, creates a conceptual-physical stimulating development environment. For reasons of space limitation, we will not elaborate on this here. We hope that the spirit of this combination is reflected in our examples presented in Section 3.

The introduction of XP into the course consisted of two main phases: In Phase 1, prior to the onset of the course, a team of four supervisors was introduced to XP and trained to teach it. Table 1 presents the timetable for the supervision team training. The second author, who was the instructor in charge of the course, led the supervisor team sessions. All sessions were conducted with a teamwork orientation and all decisions were made jointly by the supervision team. In Phase 2, XP was implemented in the Summer semester in the form of four studios. Each studio worked on a different project, chosen by its supervisor, in teams of 10-12 students. Attendance of all sessions during the semester was compulsory. In the 2003 Winter semester (opening in mid October 2002) students will be required to select one of the following two options: Developing a project in the XP studio as part of a team of 10-12 students; or developing a project as part of a 2-3 student team, with remote guidance of a supervisor and in a traditional SDM, like the waterfall model. The option chosen by each student will be his or her development framework for the entire semester. Obviously, different projects of different scale will be developed in these two formats.

Session No.	Agenda -- Supervision Team Training
1	Introduction to XP Personal experience with XP practices
2	The studio structure
3	The supervisor’s role in the studio Teaching XP practices
4	Student evaluation policy

Table 1: Schedule for the Supervision Team Training

The following research tools were used in the gathering of data: Video tapes of the XP sessions in one of the studios, observations of students’ work and the social interaction in the studio, students’ weekly reflections, students’ homework assignments, supervisors’ reflections, and our on-

² The DSDM Consortium website is <http://www.dsdm.org>.

going reflection on the course progress. The principles presented in the following section were derived from an analysis of the data obtained using these research tools.

3. Principles for teaching software development methodologies

This section presents ten principles for the teaching of SDMs. Some of these principles are especially suited to situations in which an SDM is introduced into a Computer Science department for the first time. Based on our experience described above, each principle is illustrated through the teaching of XP. We believe, however, that the principles can be implemented in the teaching of other SDMs as well.

Principle I addresses the *course* structure; Principles II-VI address the actual *teaching* of SDMs; Principles VII-IX address the *people* involved in the educational environment – the students and the supervision team; the last principle – Principle X - is a *meta* principle which suggests action over preaching. Before presenting the principles in depth, we present them briefly in Table 2. In order to facilitate their remembrance, we have given each of them a one-word nickname.

Principle No.	Category	Nickname	Single Sentence Description
I	Course	Project-based	Course structure: Project-based team-based course
II	Teaching	Cognition	Teaching an SDM: What aspect to emphasize?
III		Adjustment	Adjustment of the SDM to a university course framework
IV		Projection	Projection of the SDM's notions into the project development environment
V		Connectivity	The SDM in the context of the world of software development
VI	People	Evaluation	Reflection of the SDM orientation in students' evaluation policy
VII		Listening	Listening to students' objections
VIII		Reflection	Students' reflections and progress diaries
IX		Feedback	Supervisors' hesitations and feedback
X	Meta	Inspiration	Inspiring the SDM rather than preaching it

Table 2: 'Teaching SDM' Principles in a Glance

I. Course structure: Project-based team-based course: This principle suggests teaching an SDM in a project-based course. In such a course, students develop, within a single-semester time frame, software projects that are suited for development using the SDM in question. This setting contrasts with the traditional teaching method, in which SDMs (or any other topic) are presented in the lecture hall, with students passively listening to the philosophy of the SDMs, its process, guidelines and practices. We argue that it is important to teach SDMs in environments that imitate real life situations, projects and software development environments, as closely as possible. Indeed, gaining such experience can prepare the students for their future work in the software industry.

This principle stems from the importance attributed to providing learners with the experimental basis needed for learning complex concepts. This stands in line with the constructivist cognitive perspective, the origins of which are rooted in Jean Piaget's studies. The constructivist paradigm examines learners' mental processes that lead to mental constructions of knowledge based upon learners' knowledge and experience. We believe that an SDM is a complex concept, which can be better understood if learners are afforded the opportunity to experience it. We do not claim that no frontal teaching should be used at all; Some aspects of the SDM can be taught by lecturing. However, we believe that the more students experience the SDM and reflect upon it, the more they will improve both their understanding of the SDM's essence and orientation and their professional skills. As is described in Section 2, this principle is implemented in our experience of teaching XP. Further details are provided in the next principles.

II. Teaching SDM: While dealing with the teaching of an SDM, a need arises to integrate some teaching/learning modules into the course, which address the essence of the SDM being taught. Since an SDM is, by nature, a complex topic, several questions must be addressed: What aspects of the SDM should be emphasized? Should one teach all of the SDM ideas? Perhaps only part of them? Which ones? In order to answer such questions, we suggest analyzing and mapping the SDM ideas, in such a way that highlights those aspects of the SDM that form its core, and, for each specific SDM idea, reflects the complexity of teaching it. One way to do this is illustrated in the following mapping.

The main ideas of the SDM are mapped along two dimensions: ‘Aspect’ and ‘Cognitive Awareness’. Specifically, on one dimension, the SDM ideas are divided into either a code/technical aspect or a human/social aspect. The second dimension maps the main ideas of the SDM according to the level of cognitive awareness (or cognitive complexity) required to implement each of them throughout the development process. We believe that such a mapping can be suitable for the mapping of the main ideas of any SDM, and can thus assist lecturers in organizing their courses. Clearly, the mapping presented above is not the only mapping possible. If other mappings are appropriate, they should, of course, be implemented.

The case of XP: XP consists of 12 practices that form the core of the methodology. Table 3 presents these practices according to the two-dimensional mapping suggested above. In this mapping, we roughly assume that some of the practices are easily understood (e.g., 40-hours a week), some are hard to understand (e.g., simple design or refactoring), and others are at an intermediate level of cognitive complexity (e.g., coding standards).

We believe that when XP is taught, such a mapping enables the instructor to choose which aspects to focus on, in such a way that students will not be overwhelmed with too many hard-to-grasp concepts in too short a period of time. For example, one might emphasize the technical practices (of both high and low cognitive complexity) and pay less attention to the social practices.

Cognitive Awareness \ Aspect	Code/Technical Perspective	Human/Social Perspective
High	Refactoring Simple design	Metaphor Collective ownership
Intermediate	Coding standards Testing	Pair programming Planning game
Low	Small releases Contentious integration	Customer on site 40-hour a week

Table 3: Mapping of the XP Practices

III. Adjustment of the SDM into a university course framework: As far as the desire to teach a SDM in an environment that resembles, as much as possible, real software development environments is concerned, the traditional university course frameworks sometimes require adjustment of the SDM. This is due mainly to the fact that students take other courses in parallel and cannot dedicate the entire week to their work on the course project.

The case of XP: With respect to XP, two of the adjustments made in order to adapt XP to the framework of a university course are presented, as follows:

- The supervision team hypothesized that when playing the planning game (one of the XP practices) students might not add product features, knowing that they will then be required to implement them. Thus, it was made clear to the students that the rules of the planning game were changed slightly for the sake of the course project. Namely, if as a result of the planning game it would become evident that the project requirements became too broad to be implemented within the time frame of a single semester, a subset of the requirements will be chosen and the project’s scope will be

defined accordingly. The supervisors undertook to keep the scope of the project realistic relative to the given time frame.

- In order to help students stay focused on the project and on track, the supervisor took the role of the XP project coach (one of the roles in XP). In order to enable the students to experience this role at least partially, it was decided to appoint a coach-apprentice to act as the actual coach in between the weekly sessions. This approach also ensures that the students would feel that they are the responsible entity for the project progress.

IV. Projection of several SDM's notions into the project development environment: This principle suggests that the development environment, in which students are supposed to work, should reflect the SDM being taught. For example, if an SDM emphasizes teamwork, the learning environment should support teamwork. As in the case of the evaluation policy (see Principle VI), this principle supports students' feelings that they receive a clear and coherent message.

The case of XP: Two XP practices, for which this principle was implemented in the case described here, are presented below:

- **Pair programming:** In order to promote pair programming, the number of computers in each studio is set to be equal to half the number of students in each studio. Thus, students naturally begin programming in pairs. Still, more effort should be invested to ensure pair-exchanging.
- **Continuous integration:** In university courses, in which students do not perform all of the work together, at the same time, a situation can occur by which a student enters the studio in between weekly sessions, integrates his or her code, causes irreversible damage, and leaves the studio (or in the worst case – drops the course altogether). One plausible solution, which might solve this problem, is to enable integration only during the weekly sessions, which are attended by the supervisor. This solution could, however, slow down students' progress in between weekly sessions and may diminish the importance of this XP practice. Thus, it was decided to formulate, together with the students, a set of rules according to which all students would work and which would be posted near the integration machine. In other words, instead of doing away with continuous integration, a solution was found in order to emphasize the importance of this practice.

V. The SDM in the context of the world of software development: The world of software has witnessed relatively many cases in which new terms have emerged and have shortly turned out to be no more than buzzwords. Thus, when teaching a new SDM, it would be preferable to connect the SDM to the world of software development in general, and to other SDMs in particular. This can be done, for example, by presenting students specific problems faced by the software industry, illustrating how the taught SDM may help overcome them. Students will then, hopefully, feel that, on the one hand, they are being introduced to an SDM that is not detached from the software world and is not just a passing fashion, and on the other hand, that the SDM in question has emerged as a timely answer to the needs of the software industry and that it will be useful to them in their future work as software developers. In this way, the message conveyed to the students is that what they are learning is closely related to what they have learnt so far and to what they will be doing after they graduate.

The case of XP: In the case of teaching XP, the need for agility in software development was first explained and some problems with traditional SDMs were outlined. When students raised questions regarding the compatibility of XP with what they have previously learnt, the connection was made between XP and topics that were familiar to the students from other courses, such as UML and CMM. Such a broad perspective enables students to see XP's place in the software industry and to see that SDMs is a field that is still undergoing development.

The connection between the taught SDM and the world of software development can also be made in ad hoc situations. For example, one of the students had expressed his objection to the concept of simplicity (one of the XP values), saying that “It runs against what we have already been taught – that is, to pre-design, to define all the data structures in advance, to get a global view of the software being developed, before getting into the details”. The supervisor’s reaction was that the global view is not at all ignored or overlooked. On the contrary, the global perspective of the software is inspired by the refactoring practice and it is used for redesigning the software in a continuous process. However, it was emphasized that there are cases, at certain stages of software development, when a mental global picture cannot be conceived, and then simplicity and software development in small releases can be useful tools.

VI. Reflection of the SDM orientation in student evaluation: It is accepted that when a university courses instructor wishes that his or her students follow specific principles that he or she deems important, these principles must somehow be incorporated into the evaluation policy of the course. This is particularly true when an SDM is used in a project-based course. It is reasonable to assume that students naturally devote more effort to what is valued (and graded), and eventually improve their understanding of the SDM.

The case of XP: This principle was applied in the course described here, by assigning a course grade that was composed of a personal component (35%) and a team component (65%), which was identical for all members of the team. We believe that such an evaluation scheme conveys the message that both teamwork and individual contribution count. It is assumed that, on the one hand, students will be encouraged to contribute to the teamwork, and on the other hand, this method affords those wishing to excel, the opportunity to improve their grade through the personal component of the grade.

Table 4 presents the evaluation model formulated by the supervision team and used in the four studios. As can be observed, most of the XP practices appear explicitly in the evaluation model, with special emphasis on the first-test and pair-programming practices. The first-test practice gets its importance from the supervision team’s belief in educating students for production of high quality software products. The student-supervisor pair-programming session emerged as a solution to the “free riders” phenomenon, which was evident in previous semesters (when the supervisors did not meet the students on a weekly basis and had no way of knowing just how the project was actually progressing).

<u>Team Component (65%)</u>	<u>Individual Component (35%)</u>
15% - Setting up the computerized environment of the studio 25% - Project documentation 60% - For each of the two project releases: * Implementation of the customer stories * Complete releases on time (according to students' estimations in the planning game)	45% - Attendance the 8 Studio meetings Written reflections at the end of each meeting Pair-programming session with supervisor Homework assignments about XP practices Presentation of a first-test experience 55% - Performance of a personal role (release presenter, iteration presenter, in charge of continuous integration, customer, tracker, coach assistant, in charge of testing)

Table 4: Grading Policy in the Project-Based XP Course

VII. Students’ objections: If Principle I is applied and if the student-lecturer lecture hall interaction is replaced by intensive student-supervisor communication in a studio or in another development environment, it is reasonable to assume that the change in the teaching approach will affect both students and supervisors. Specifically, since we are dealing with educational institutions in which both students and teachers have learning/teaching belief systems, it is natural that feelings of

discomfort will arise when teaching conditions change. This principle addresses the question of how students' reactions should be addressed when such feelings emerge. In Principle IX we address supervisors' concerns. Based on our experience, we propose that instructors and supervisors should try sympathizing with students' feelings towards the new learning environment and be patient with them until students start becoming aware of the benefits that can be gained from the new course setting in general, and from the introduced SDM in particular.

The case of XP: As it happens, in the course described here, student objections disappeared in part after only one week. Thus, it seems that the sympathizing attitude mentioned above was indeed called for. This shift in students' attitudes can be explained by their newly emerged awareness to the advantages of the new approach. First, they were told that if they proceed according to the decisions they made in the planning game, they would complete their projects by the end of the semester. This situation stands in contrast to other project-based courses, in which some students do not complete their project by the end of the semester, dragging it even months later. Second, students began to realize that their team might actually support their work, and not just hinder it, as they originally suspected it would.

In order to illustrate students' concerns with respect to the new course format, we present two issues that were raised by students during the first session:

- *“The main component of the grade is the team component, and I do not know who my team members are. How can I trust them? This is not a real working place”.* It seems that this concern stems from bad experiences students had in other courses, in which they were required to develop projects in teams. It seems that, in some cases, students had had to cope with “free riders”, fellow students who did not contribute to the progress of their projects. We believe that XP is taught in such a way that avoids such a situation: First, all team members work together in the studio and no student would like to be perceived as a “parasite”. Second, each student has a very specific task assignment, which is determined during the planning game.
- *“Who is the team leader? Who controls what students do?”.* The instructor explained that in the previous format of the course as well, there was no control over students' work. Furthermore, it was emphasized that the weekly meetings and the coach-apprentice assistant increase the control over the project progress.

VIII. Student reflections and progress diaries: In order to learn about the feelings of students, their working habits and their social interactions with respect to the project development during the course, students should be offered tools and means for self-expression. The ability to express one's impressions gives students the feeling that their thoughts and feelings are of interest to someone (in our case, the course instructors and supervisors). In addition, if students wish to complain, this is a good way to encourage them to do so. In this spirit, students should be encouraged to express not only positive ideas, but also negative feelings and suggestions for improvement.

Furthermore, we believe that such means of expression enhance the students' reflective skills; skills whose importance in software development processes is acknowledged and which are encouraged by the studio approach. As it is well known in the software industry, a reflective person, who learns both from the successes and failures of previous software projects, is more likely to improve his or her own performance in the field.

The case of XP: The following two means of expression were offered to students in the present XP course, in order to express their reflections and feelings and to describe project progress: A team diary, which is part of the ‘project documentation’ evaluated as part of the team grade (cf. Principle VI); and personal web-reflections, which are posted on the web at the end of each session and which are to be completed within three hours after the end of each session. The web-reflections were short (requiring no more than 15 minutes) and presented questions that address students' impressions from the session they just attended. As it turns out, these web-reflections provided the supervisors with quite good feedback. For example, the first web-reflection that was posted following the first session,

just after the first planning game had been played, revealed the following observation. On the one hand, students admitted that working in a big team (10-12 students) is a new experience for them and some of them were looking forward to working in such teams. On the other hand, this working style was also their main concern: They were worried about how the teamwork would be coordinated and who would lead the team.

The reflections were posted on a web-based course management tool that also served for the posting of learning material and administrative messages by the supervisors, and for student e-communication in the various forums.

IX. Supervisors' hesitations: While Principle VII addresses the students' state of mind at the onset of the course, this principle focuses on the supervisors who are to guide students in the new software development environment. This principle is suited mainly to situations in which a *team* of supervisors guides the students, and, similar to the students, experiences the development environment for the first time. In such cases, not only must the supervisors familiarize themselves with the SDM itself, but also they have to adopt a new way of teaching and a new supervision conception. For example, supervisors must be physically present, alongside the students, during the actual software development stages. This is not something previously experienced in the traditional supervision style, in which meetings are held with the students in between student programming sessions. In addition, the new evaluation model requires the supervisors to evaluate items that were not previously evaluated. Indeed, such a change requires a transformation in the supervisors' conception of teaching and of learning processes.

The principle suggests repeated emphasizing of the novelty of the introduced SDM and legitimizing supervisors' feelings of insecurity. Furthermore, it is suggested that the supervisors state before the students that this experience is new for them as well, and thus relate such a situation to real life software development circumstances, in which uncertainty in general, and not-knowing-everything in particular, are frequent components (Cf. Principle V). In practice, the supervisors' role in the new development environment and other supervisor-students-interaction related issues should be discussed during the supervision team sessions held prior to the course. In order to overcome the supervisors' hesitations (and sometimes resistance), general decisions about the course should be made jointly by all supervision team members, leaving each supervisor the freedom to adjust the particular studio he or she teaches to his or her teaching beliefs.

The case of XP: Since the supervision team of the course described in this paper was not familiar with XP prior to this course (neither in practice nor in theory), it was necessary to teach them XP philosophy as well as XP practices. This was accomplished during the 4 training sessions described in the Introduction. The sessions were based on short presentations followed by some activity (such as pair drawing, planning game, reflection). In the course of the training process, the supervisors began feeling comfortable with XP. For example, after the first planning game, one of them said: "This way, I believe, less tension will be experienced by the students towards the end of the semester". Comparing her previous supervision experiences with her current knowledge about XP, this supervisor realized that the planning game will enable students to complete their projects on time, and without the stress that, in the traditional course format, characterizes the end-of-the-semester period.

Most of the decisions, regarding the structure of the course (4 studios), the course schedule (2 releases, each consisting of two iterations), and the role of the supervisors in the studio, were made jointly by all supervisor team members. As suggested above, in addition to these general decisions, each supervisor made specific decisions with respect to the studio he or she was intended to teach. For example, each supervisor decided on the software project to be developed by the students in his or her studio. In the 2002 Summer semester (the first semester in which XP has been introduced in the course), it was decided to offer students projects that had already been developed in previous semesters and to adjust the scale of such projects to team sizes of 10-12 students. This way the supervisors could focus on the XP practices without being distracted by technical issues. In the 2003

Winter semester, it is expected that the supervisors will feel more comfortable with XP, and new topics for projects will be introduced.

X. Inspiration of the SDM: This is a meta-principle that integrates several of the principles described above. It suggests inspiring the SDM, rather than talking about it. In fact, this principle is supported by all of the previous principles, such as the experimental basis, the projection of some SDM principles into the project development environment, and the reflection of the SDM orientation in the students' evaluation model.

The case of XP: We hope that this principle is mirrored in the way we have described the course on which our paper is based. We have tried to convey the message that, throughout the entire course, XP is inspired on all levels: Individual, pair, team, supervisor, and customer.

4. Conclusions

This paper discusses the teaching of SDMs. The importance of teaching SDMs in the manner introduced here, stems from our belief that students should be introduced to SDM in a learning environment that simulates as much as possible environments they are likely to encounter in their future careers.

The main ideas are demonstrated based on our experience in teaching XP in a project-based course. Although this paper focuses on the teaching of SDMs in general, we would like to present some concluding remarks with respect to our XP teaching experience. First, as was anticipated, students' projects were completed on time. We believe that this fact, by itself, will dramatically (and perhaps entirely) reduce students' resistance in future semesters. Second, and maybe more important, we observed a strong commitment to the project development on the part of both students and supervisors. Student commitment can easily be explained by the evaluation model (Cf. Principle VI) and the planning game: The evaluation model guided students to personal and team commitment; the planning game assigned each student with specific tasks to be accomplished in a specific estimated time. It is interesting to note the commitment to the entire project observed on the part of the supervisors: The supervisors cared about the implementation of the XP practices, and were concerned about the on-time completion of the projects.

In our future work, we intend to propose a more detailed description of the teaching of SDMs within a single-semester time frame. In addition, we have begun an examination of the adoption of the principles presented in this paper in the teaching of other topics, such as programming paradigms. For example, the suitability of the ten principles will be examined in an introductory object-oriented course scheduled to be offered in the 2003 Winter semester to 25 graduate students, for whom this will be their first programming experience.

References

- Beck, K. (2000). *Extreme Programming Explained: Embrace Change*. Addison-Wesley.
- Denning, P. J. (1992). Educating a new engineer, *Communications of the ACM* **35** (12), pp. 82-97.
- Fowler, M. (2002). The New Methodology, published in martinowler.com, <http://www.martinowler.com/articles/newMethodology.html>
- Hazzan, O. (In press, 2002). The reflective practitioner perspective in software engineering education, *The Journal of Systems and Software*.
- Kuhn, S. (1998). The software design studio: An exploration, *IEEE software* **15**(2), pp.65-71. <http://www.computer.org/software/so1998/s2065abs.htm>.
- Meyer, B. (May, 2001). Software engineering in the academy, *Computer*, pp. 28-35.
- Schwaber, K. and Beedle, M. (2001). *Agile Software Development with Scrum*. Prentice Hall.
- Sebern, M.J. (2002). The Software Development Laboratory: Incorporating Industrial Practice in an Academic Environment, *Conference on Software Engineering Education and Training (CSEE&T)*, pp. 118-127.

- Tomayko, J. E. (1996). Carnegie-Mellon's Software development Studio: A five-year retrospective, *SEI Conference on Software Engineering Education*.
<http://www.contrib.andrew.cmu.edu/usr/emile/studio/coach.htm>.
- Wilson D. (2001). Teaching XP: A Case Study, *XP Universe Conference*.