Teaching Framework for Software Development Methods

Orit Hazzan
Department of Education in Technology and Science
Technion – Israel Institute of Technology
Haifa 32000, ISRAEL
oritha@techunix.technion.ac.il

Yael Dubinsky
Department of Computer Science
Technion – Israel Institute of Technology
Haifa 32000, ISRAEL
yael@cs.technion.ac.il

ABSTRACT
In this paper we suggest a framework for teaching software development methods (SDMs). Specifically, based on our accumulative research and in-practice experience of teaching SDMs, a set of principles, that guides our teaching of SDMs in different settings and teaching experiences, has been formulated. The teaching framework consists of 14 principles that their actual implementation is varied and adjusted in different teaching environments. This paper outlines the principles and addresses their contribution to learners' understanding of the said software development method.

Categories and Subject Descriptors
D.2.0 [**Software Engineering**]: General;
K.3.2 [**Computer and Information Science Education**]: Computer science education.

General Terms
Management, Human Factors.

Keywords
Software Engineering, Software Engineering Education, Software Development Methods, Constructivism, Active Learning, Learning, Teaching.

1. INTRODUCTION
In this paper we suggest a framework for teaching software development methods (SDMs). Specifically, based on our accumulative experience in teaching SDMs we formulated a set of principles that guides our teaching of SDMs in all our teaching experiences; the implementation of these principles, however, is varied in different teaching environments.

2. TEACHING FRAMEWORK
Table 1 outlines the teaching principles which are formulated as pedagogical guidelines. Naturally, when we teach different aspects of SDMs, different principles are highlighted according to the topic's main characteristics. At the same time, however, the principles presented in Table 1 provide a comprehensive picture of our teaching framework.

ICSE'06, May 20-28, 2006, Shanghai, China.
ACM 1-59593-085-X/06/0005.

Table 1: Teaching Framework by Principles

Principle 1: Inspire the SDM's nature
Principle 2: Let the learners experience the SDM principles as much as possible
Principle 3: Explain while doing
Principle 4: Elicit reflection on experience
Principle 5: Elicit communication
Principle 6: Establish diverse teams
Principle 7: Assign roles to team members
Principle 8: Manage time
Principle 9: Be aware of abstraction levels
Principle 10: Inspire a process of an on-going and gradual improvement
Principle 11: Use metaphors or "other world's concepts"
Principle 12: Consider the awareness needed for the implementation of the SDM practices
Principle 13: Listen to participants' feelings towards the presented SDM
Principle 14: Emphasize the SDM in the context of the software world

In what follows we elaborate on the principles. Due to space limitations, all principles are described in brief.

Principle 1: Inspire the SDM's nature
This is a meta-principle that integrates several of the principles described in the continuation of this paper and, at the same time, is supported by them. It suggests inspiring the SDM, over lecturing about it.

The application of this principle is expressed by active learning, on which the next principle elaborates. In addition, it is reflected in the teaching environment. More specifically, active-learning based lessons should take place in a site that enables the actual performance of the SDM. For example, it is preferable to learn Extreme Programming (XP) [1, 2] in sites that include a large table for the planning game, computers arrangement for pair programming, and flipcharts or a whiteboard to elicit communication processes.

Principle 2: Let the learners experience the SDM as much as possible

This principle is derived directly from the previous one. In fact, these two principles stem from the importance attributed to learners' experimental basis, which is essential in learning processes of complex concepts. This assertion stands in line with the constructivist perspective of learning, the origins of which are rooted in Jean Piaget's studies. According to the constructivist approach, learners construct new knowledge by rearranging and refining their existing knowledge [cf. 3, 16]. In this process, mental structures are developed in steps, each step elaborating on the preceding ones.

We suggest that an SDM is a complex concept. Therefore, its gradual learning process should be based on learners' experience. One way to support and enhance such gradual mental learning process is to adopt an active-learning teaching approach, according to which learners are *active* to the extent which enables a reflective process (see also Principle 4 in the continuation of the paper).

Principle 3: Explain while doing

This principle relates to the just presented "Let the learners experience the SDM as much as possible" principle and to the constructivist approach mentioned there. It implies that when an activity or a practice is introduced to learners at the first time, the educator should *not* explain it for too long, but rather the educator should invite the learners to start applying it as soon as the basic relevant knowledge has been introduced. While the learners are performing the said activity, the educator should add the details gradually, refine the explanation, and reflect on learners' activities.

A good example to illustrate this principle is the planning game, one of the XP practices mentioned above [6]. The planning game consists of many relatively simple stages and details. However, if learners hear all these details before they start experiencing the planning game, the global picture, as well as the logic of each stage, the order of the stages, and their interconnections, might not be clear. We suggest that if learners hear first just the main ideas of the planning game and then start playing it, when the details are gradually introduced during this process, the different stages may be situated within the wider context and clarify the structure and logic of the entire process.

Principle 4: Elicit reflection on experience

The importance of introducing reflective processes into software development processes has been already discussed [7, 10, 11], mainly based on Schön's *Reflective Practitioner* perspective [14, 15].

According to this principle, learners should be encouraged to reflect on their learning process of the said SDM. We note that reflection processes should not be limited to technical issues, but rather should address also feelings, working habits and social interactions related to the software development processes. In addition, learners can be asked to recall and reflect on situations taken from their past experience in software development and be presented with questions that request them to analyze and compare different situations related to software development processes.

Principle 5: Elicit communication

Communication is a central theme in software development processes. Indeed, sometimes, successes and failures of software projects are attributed to *people* communication issues. Accordingly, in all learning situations we aim at fostering learner-learner as well as learner-teacher communication. When communication is one of the main ingredients of the learning environment, the idea of knowledge sharing becomes natural. Then in turn, knowledge sharing reflects back on communication.

Principle 6: Establish diverse teams

Diversity can be expressed in different ways, such as nationalities, gender, minorities, cultures, life styles and world views. The importance attributed to diversity is based on management theories that assert the added value of diversity (cf. the American Institute for Managing Diversity *http://aimd.org/*). Therefore, and not surprisingly, diversity is perceived as a powerful management practice (see for example, Toyota's 21st Century Diversity Strategy[1] and David Thomas's paper in Harvard Business Review[2]). In the same spirit, with respect to software development, diversity is encouraged in agile and lean software development processes. Further, since more and more companies become global, diversity becomes an integral characteristic of software development teams; therefore diversity can not be neglected when SDMs are taught.

This principle advocates the idea that teams should be diverse. Naturally, the more diverse a team is, the more diverse perspectives are elicited. We suggest that this diverse view points may improve software development processes as well as learning processes related to SDMs.

Principle 7: Assign roles to team members

In the different settings in which we teach SDMs, each team member has an individual role, chosen by the team member from a given list, in addition to the personal development tasks for which each team member is responsible. Based on our teaching experience, we have identified twelve roles (for example, coach, unit tester, acceptance tester, code reviewer, etc), each of them is related to a specific aspect of software development processes [4]. The role scheme does not imply that each role holder carries out all the activities related to the domain for which he or she is responsible; rather, each role holder makes sure that the activities related to the domain for which she or he is responsible, will be accomplished satisfactory by all team members.

The assignment of roles helps splitting, among all the team members, the responsibility for the project progress. The rationale for this act stems from the fact that one person (or a small number of developers) can not take care of the entire richness and complexity involved in software development processes. When the responsibility is split among all team members, each aspect of the entire process is treated by one team member, and, at the same time, each team member feels a personal responsibility for that specific aspect. Indeed, both the project itself and the team members are advantaged from this kind of organization. Further, the need to perform one's role successfully actually enforces one to be involved in and to become familiar with all parts of the

[1] Toyota's 21st Century Diversity Strategy: http://www.toyota.com/about/diversity/21stcenturyplan.pdf

[2] David Thomas, "Diversity as Strategy", *Harvard Business Review*, September 2004, pp. 98-108. http://www.gpworldwide.com/quick/sep2004/art2.asp

developed application. Consequently, knowledge sharing and involvement is increased among team members.

Principle 8: Manage time

This principle relates to time management and is expressed in our teaching of SDMs in different ways, two of which are presented here:

First, we emphasize sustainable pace and reinforce the participants' feelings about the potential contribution of sustainable pace to software development processes.

Second, time estimation for development tasks (and the assessment of this time estimation) is never skipped. Furthermore, learners are asked to estimate (and then to evaluate) the way their time is shared between the accomplishment of their personal development tasks and the performance of their personal role (Principle 7).

Principle 9: Be aware of abstraction levels

In [9] we suggest that in the process of software development, developers are required to think on different abstraction levels and to move between abstraction levels.

This principle suggests that instructors or workshop facilitators who teach an SDM should be aware at what abstraction level each stage of each activity is performed. Based on this awareness, they then should decide whether to stay at this abstraction level, or, alternatively, maybe there is a need to guide the participants to think in terms of a different level of abstraction. It is further suggested to explicitly highlight the movement between abstraction levels and to discuss with the learners the advantages, as well as disadvantages, we may gain from such moves.

Principle 10: Inspire a process of an on-going and gradual improvement

It is clear that software development is a gradual process, conducted in stages, each one improves its preceding ones. In many cases this improvement takes place in parallel to improvement in the developers' understanding of the developed application.

Clearly, this principle is derived from the constructivist approach presented in Principle 2. Accordingly, the learning environment should specifically inspire that feeling of a gradual learning process and elicit reflection processes when appropriate (cf. Principle 4).

We briefly describe an illustrative scenario how this principle may be applied in practice. When learners are stuck, the moderator can stop the discussion, reflects on what happened, and suggests moving on, explaining that it might make more sense to readdress the issue, that currently blocks the development progress, in a later stage when learners' background and knowledge will enable to solve the said problem.

Principle 11: Use metaphors or "other world's concepts"

Metaphors are used naturally in our daily life, as well as in educational environments. Generally speaking, metaphors are used in order to understand and experience one specific thing using the terms of another thing [12, 13].

Metaphors can be useful even without specifically mention the concept of metaphor. For example, the facilitator may say: "Can you suggest another concept-world that may help us understand this unclear issue". Our experience indicates that learners do not face any problem with suggesting a varied collection of concept-worlds, each world highlights a different aspect of the said issue and together they support the comprehension of the discussed topic.

Principle 12: Consider the awareness needed for the implementation of the SDM practices

As has been mentioned previously, an SDM is a complex idea. Based on the constructivist approach mentioned above, it should be introduced in stages. Accordingly, several questions should 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? This principle suggests a way by which a teacher/workshop facilitator can select what ideas to teach and in what order.

We suggest analyzing and mapping the SDM ideas in a way that reflects the complexity involved in the teaching and learning of each idea. One way to do this is illustrated in the following mapping [8]. We believe that such a mapping can be suitable for the mapping of the main ideas of any SDM, and can thus assist instructors and facilitators in organizing their teaching.

Specifically, the main ideas of the SDM are mapped along two dimensions: ‘Aspect’ and ‘Cognitive Awareness’. Specifically, on the Aspect dimension, the SDM ideas are divided into either a code/technical aspect or a human/social aspect. The Cognitive Awareness 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.

Clearly, this mapping is not the only mapping possible. Our experience, however, tells us that when an SDM is taught, such a mapping may help the teacher 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.

Principle 13: Listen to participants' feelings towards the presented SDM

The adoption of a new SDM requires a conceptual change with respect to what a software development process is. In addition, it is well known that cultural changes related to the introduction of a new SDM raises emotions that, in some cases, are in fact objections.

In practice, when learners express emotional statements against some aspect of the SDM, we propose to take advantage of this opportunity and to encourage participants to describe the subject of the said statement as it is manifested in their current software development environment. As it turns out, in many cases these descriptions elicit problems in the current SDM used. Then we explain how the learned SDM attempts to overcome the problematic issues just raised. For example, when a statement is made against the test-driven development approach, it is a good opportunity to ask the person making this statement to describe the testing process that he or she currently uses. In some cases this in itself is sufficient: The question highlights the test-driven development approach towards the discussed issue, and consequently the facial expression of the person expressing the objection changes immediately.

In general, in all teaching situations we propose to try sympathizing with and legitimizing learners' feelings towards the new SDM, and be patient with them until learners start becoming aware of the benefits that can be gained from the SDM. As it happens, in many cases learners' objections disappeared in part after a short part of the teaching period. One plausible explanation is that they begin to realize that the SDM might actually support their work.

Principle 14: Emphasize the SDM in the context of the software world

This principle closes the circle that has been open with the first principle – Inspire the SDM's nature. We believe that part of this inspiration is related to the connections made between the taught SDM and the world of software engineering. Since the world of software engineering has witnessed relatively many cases in which new terms have emerged and have shortly turned out to be no more than buzzwords, 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 to the learners specific problems faced by the software industry, illustrating how the taught SDM may help overcome them. Learners will then, hopefully, feel that, on the one hand, they are being introduced to an SDM that is not detached from the software industry 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 the future.

For example, in the case of teaching the agile approach, the need for agility in software development may be first explained and some problems with plan-driven SDMs may be outlined. Such a broad perspective enables learners to understand the place of the agile approach in the software industry world and to observe that SDMs is a field that is still undergoing development.

3. SUMMARY

The collection of principles presented in this paper aims at establishing a teaching framework within which we teach SDMs. In [5] we present a broader theoretical framework for teaching SDMs.

ACKNOWLEDGMENTS

Our thanks to Technion V.P.R. Fund – B. and G. Greenberg Research Fund (Ottawa) for supporting this research.

4. REFERENCES

[1] Beck, K. (2000). *Extreme Programming Explained: Embrace Change*, Addison-Wesley.

[2] Beck, K. (with Andres, C., 2005, second edition). *Extreme Programming Explained: Embrace Change*. Addison-Wesley.

[3] Davis, R. B., Maher, C. A. and Noddings, N. (1990, eds.). Constructivist views on the teaching and learning of mathematics, *Journal for Research in Mathematics Education*, Monograph Number 4, The National Council of Teachers of Mathematics, Inc.

[4] Dubinsky, Y. and Hazzan, O. (2004). Roles in Agile Software Development Teams, *Fifth International Conference on Extreme Programming and Agile Processes in Software Engineering*, Garmisch-Partenkirchen, Germany, pp. 157-165.

[5] Dubinsky, Y. and Hazzan, O. (2005). A Framework for teaching software development methods, *Computer Science Education* **15**(4), pp. 275-296.

[6] Fowler, M. and Beck, K. (2002). *Planning Extreme Programming*, Boston, MA.

[7] Hazzan, O. (2002). The reflective practitioner perspective in software engineering education, *The Journal of Systems and Software* **63**(3), pp. 161-171.

[8] Hazzan, O. and Dubinsky, Y. (2003A). Teaching a Software Development Methodology: The Case of Extreme Programming, The proceedings of the *16th International Conference on Software Engineering Education and Training*, Madrid, Spain, pp. 176-184.

[9] Hazzan, O. and Dubinsky, Y. (2003B). Bridging cognitive and social chasms in software development using Extreme Programming, *Proceedings of the Fourth International Conference on eXtreme Programming and Agile Processes in Software Engineering*, Genova, Italy, pp. 47-53.

[10] Hazzan, O. and Tomayko, J. (2003). The reflective practitioner perspective in eXtreme Programming, *Proceedings of the XP Agile Universe 2003*, New Orleans, Louisiana, USA, pp. 51-61.

[11] Kerth, N. (2001). *Project Retrospective*, Dorest House Publishisng.

[12] Lakoff, G., and M. Johnson, *Metaphors We Live By*, The University of Chicago Press, 1980.

[13] Lawler, J.M. (1999). Metaphors we compute by, In *Figures of Thought: For College Writers* by D.J. Hickey, Mayfield Publishing.

[14] Schön, D. A. (1983). *The Reflective Practitioner*, BasicBooks,

[15] Schön, D. A. (1987). *Educating the Reflective Practitioner: Towards a New Design for Teaching and Learning in The Profession*, San Francisco: Jossey-Bass.

[16] Smith, J. P., diSessa, A. A. and Roschelle, J. (1993). Misconceptions reconceived: A constructivist analysis of knowledge in transition, *The Journal of the Learning Sciences* **3**, pp. 115-163.