

Human Aspects of Software Engineering: The Case of Extreme Programming

Orit Hazzan¹ and Jim Tomayko²

¹Department of Education in Technology and Science, Technion - IIT, Haifa 32000, Israel
oritha@tx.technion.ac.il

²School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, U.S.A.
jet@cs.cmu.edu

ABSTRACT

As with to other agile methods, which value "Individuals and interactions over processes and tools" (<http://agilemanifesto.org/>), Extreme Programming (XP) cares about the interaction among the people involved in software development processes. The four XP values and its twelve practices inspire this feeling as well. Accordingly, and not surprisingly, in a course that we teach about human aspects of software engineering and in a book that we are writing about the topic (Hazzan and Tomayko, 2004, in preparation), we find it illuminating to highlight human aspects of software engineering incorporated in XP. This article gathers these illuminations into one place.

1. INTRODUCTION

If you ask a group of software engineers what software engineering is, you would probably come up with more than one definition, each definition emphasizing different aspects of the discipline. This phenomenon is reflected also in the definitions of software engineering described in the professional literature. Indeed, software engineering is a multifaceted discipline. The rationale for the course described in this article stems from the fact that though recently the human aspects of software engineering get more and more attention, in our opinion they do not get yet the attention they deserve.

The beginning of the awareness of the human aspects of software engineering appeared in Brooks's book *The Mythical Man Month* (Brooks, 1975, revised in 1995). In the preface to the 20th Anniversary Edition, Brooks writes that he is surprised that *The Mythical Man-Month* is popular even after 20 years. Such a statement indicates how difficult is to apply lessons that had been learnt with respect to software development to future software development projects. This difficulty may be explained by the multifaceted nature of the discipline and the uniqueness of software development processes. The course described in this article addresses this complexity as well.

As has been mentioned above, the importance of the human aspects of software engineering becomes acknowledged recently. For example, many failures of software systems can be explained by human factors. Taking into the consideration the complexity of the topic, the course described in this article focuses on social and cognitive aspects of software engineering, and addresses topics such as teamwork, customer – software-engineer interaction, and learning processes in software development.

Since the agile approach in general and Extreme Programming (XP) in particular are about humankind, we find it appropriate to illustrate human aspects of software engineering by the agile approach in general and XP in particular. The next two sections describe the rationale for the course and how some of its main messages are illustrated by XP.

2. HUMAN ASPECTS OF SOFTWARE ENGINEERING

As it is well known, the more the software world is developed, the more it is accepted by the software engineering community that the *people* involved in software development

processes deserve more attention, not the processes themselves or technology. This fact encapsulates the rationale for the introducing of the agile approach to software engineering and may explain the relatively rapid acceptance of the agile approach by software developers. In this spirit, the course presented in the article attempts to highlight the world of software engineering from the perspective of the main actors that are involved in software development processes: the individual, the team, the customer, and the organization. Needless to say, the code and technology are main actors in this process as well. Indeed, they are also discussed in the course. However, when code and technology are addressed, the discussion is conducted from the human perspective.

The course aims at increasing students' awareness of the various facets of the human aspects of software engineering. The idea is neither to cover all the available material about the human aspects of software engineering nor to supply a full, comprehensive, and exhaustive list of references about the topic. Rather, the course aims to illustrate the richness and complexity of the human aspects of software engineering and to increase the learners' awareness of problems, dilemmas, questions, and conflicts that may be raised with respect to the human aspect of software engineering during the course of software development.

3. EXTREME PROGRAMMING ILLUMINATES HUMAN ASPECTS OF SOFTWARE ENGINEERING

In this section we describe how XP serves as an illuminating example for human related topics that are addressed in the course. Specifically, in what follows we review the course lesson by lesson, and illustrate how we highlight the human aspects of software engineering by using XP.

Lesson 1. The Nature of Software Engineering: This introductory lesson sets the stage for the entire course. Both the agile approach and the heavyweight approach to software development are introduced. Yet, the terms (agile and heavyweight) are not yet introduced at this stage. Alternatively, instead of describing these two approaches, we tell a story of two software developers: One works in a software house where development processes are guided by the heavyweight approach; the second developer works in a software house that its software production is guided by the agile approach. Students are asked to compare the two environments and to formulate their preferred environment for software development. These two approaches are introduced in the second lesson of the course.

Lesson 2. Software Engineering Methods: In this lesson different software development methods are described. Naturally, one of them is XP. In general, it is explained that agile methods are more tied to the human aspects of software development. A special attention is given to the meaning of the adjective *extreme*.

Each of the four XP values and each of the dozen XP practices is examined from the human perspective. For example, it is discussed with the students how the value of communication is expressed by the practices pair programming, metaphor, simple design, on-site customer, the planning game and coding standard. It is also explained how the “40-hour week” (or, sustainable pace, or no extended overtime) is based on the assumption that tired programmers make more mistakes. It is highlighted that communication subsumes many of the XP practices. Indeed, this is not surprising since communication is very much a human characteristic of software engineering.

In addition, roles in XP teams are explained and their contribution to the software development process is outlined. Sensitive human issues in software development, such as load balancing among individuals, are highlighted as well.

Lesson 3. Working in Teams: After several central roles in software teams are described, their implementation by different methods (such as the Personal Software Process - PSP) and different team structures (such as hierarchical teams) are explained. One of the discussed approaches is the agile paradigm. It is explained that a team using agile processes probably functions better as a democratic team, since there are no layers in between the manager and the engineers, and communication, valued by many of the agile processes, is enhanced.

With respect to XP, it is illustrated how the XP practice of collective ownership ensures the democratic approach. Particularly, with respect to XP teams, the human related topic of small ego that is necessary for any democratic is emphasized. In addition, it is discussed how the XP practices may increase trust among team members.

Lesson 4. Software as a Product: This lesson examines the human aspect of software engineering from the perspective of the customers - the people that use software products. It focuses on how users and developers deal with defining software requirements in a way that fulfill customers' needs. Indeed, the process of defining customer's requirements is viewed in this lesson as a process in which both the customer and the developers participate.

When the topic is illustrated by XP, it is explained that XP advocates that if a software team wishes to produce the software that the customer needs, the customer should be able to give the developers on going feedback. For this purpose the customer should be on-site. In addition, XP outlines a very specific process (the Planning Game) for defining the customer requirements (here called 'customer stories').

Lesson 5. Code of Ethics of Software Engineering: In this lesson the focus is placed on the notion of ethics in general and the Code of Ethics of Software Engineering* in particular. The discussion focuses on the main ideas and the essence of the Code of Ethics of Software Engineering and students analyze scenarios taken from the software engineering world. The idea is to guide the students in the formulation of personal ethical behavior. This is a personal process that each practitioner should go through individually. In this sense, this lesson is different from the other lessons of the course since it raises more questions and philosophical dilemmas than it provides answers.

XP is discussed in this lesson by focusing on the practice of test-driven-development. Since the source of many of the ethical issues in the world of software engineering stems from the fact that software systems are not tested properly, it is found illuminating to describe how XP bypasses these ethical dilemmas by including the test-driven-development practice among its core practices. Students are asked to learn about the XP way of testing and to explain how this approach ensures that tests will not be skipped or diminished. It is also discussed how this approach towards testing may help developers avoid many of the ethical dilemmas, such as those that deal with how to tell customers that due to time pressure some tests are skipped.

Lesson 6. International Perspective on Software Engineering: In this lesson the field of software engineering is examined from an international perspective. The focus is placed on the community of software engineers and what is examined is expanded beyond the team and the organization frameworks. Specifically, in this lesson the influence of certain events on the global high-tech economy and the nature of software engineering in several places on the globe are explored. In addition, the topic of gender and minorities in the hi-tech industry is discussed.

Since the agile approach in general and XP in particular represent a new paradigm in the evolution of software development methods, we find it appropriate to discuss in this lesson the fitness of XP to different cultures and to women's management and communication style.

Lesson 7. Program Comprehension, Code Inspections, and Refactoring: One of the main messages of the course in general and of this lesson in particular is that the development of software is based on an iterative process through which the code structure and functionality are improved gradually. It is discussed that different software development methods deal differently with these gradual improvements. XP serves in this lesson as a suitable example since it includes refactoring (Fowler, 2000) as one of its twelve core practices.

To connect this topic to the previous lesson (International Perspective on Software Engineering), it is discussed with the students that in a way similar to other human beings' habits, one would refactor or not pending on the culture one lives in and one's attitude towards refactoring. Accordingly, for an XP team, refactoring is one of the method's practices, it is accepted naturally, it is part of the development routine and it stops feeling

* The URL of the code of ethics is <http://www.acm.org/constitution/code.html>.

like an overhead activity. Furthermore, refactoring is tightly connected to the other XP practices, such as unit testing and continuous integration.

In addition, it is illustrated how several IDEs (Integrated Development Environments) offer Refactoring menus which include actions such as Extract, Rename, etc. Two such Java IDEs are IntelliJ IDEA (<http://www.intellij.com/idea/>) and Eclipse (<http://www.eclipse.org/platform>). This inclusion of refactoring in IDEs is a clear sign that refactoring has become part of the profession of software engineering. At the same time it is highlighted that though refactoring has so many advantages, there are cases where refactoring should not be carried out. One of these cases is when the code is a mess and it would be better to start its development from the beginning. In fact, even this act can be viewed as some form of refactoring.

Lesson 8. Learning Processes in Software Engineering: This lesson is largely based on Hazzan and Tomayko (2003). Specifically, together with the students we discuss how a reflective mode of thinking may be interwoven in software development processes in general and in XP environments in particular.

The reflective practice perspective was introduced by Schön, mainly in the context of certain kinds of professional fields such as Art and Architecture (Schön, 1983, 1987). Generally speaking, the reflective practice perspective guides professional practitioners (such as architects, managers, musicians and others) towards examining and rethinking their professional creations during and after the accomplishment of the process of creation. The working assumption is that such a reflection improves both proficiency and performance within such professions. Analysis of the field of software engineering, the kind of work that software engineers usually accomplish, and the XP practices, support the adoption of the reflective practice perspective to software engineering in general and to XP development environments in particular. Specifically, it is suggested that a reflective mode of thinking may improve the application of some of the XP practices. In this lesson this possible contribution is examined.

Specifically, we aim to construct, together with the students, ladders of reflection. Ladders of reflection are described by Schön in the following way:

We can [...] introduce another dimension of analysis [for the chain of reciprocal actions and reflections that make up the dialogue of student and coach in the architecture studio]. We can begin with a straightforward map of interventions and responses, a vertical dimension according to which higher levels of activity are "meta" to those below. To move "up", in this sense, is to move from an activity to reflection on that activity; to move "down" is to move from reflection to an action that enacts reflection. The levels of action and reflection on action can be seen as the rungs of a ladder. Climbing up the ladder, one makes what has happened at the rung below an object of reflection. (Schön, 1987, p.114)

The ladder of reflection described in this quote refers mainly to student-tutor dialogue in the *architecture* studio. Hazzan (2002) expands the ladder of reflection presented by Schön to a student-coach dialogue in a *software* studio and with respect to an individual work. The idea in both cases is to illustrate how one may increase the level of abstraction of one's thinking when reflection is interwoven in software development processes.

In this lesson we construct ladders of reflection with respect to different software development situations in general and with respect to XP-based cases, such as a pair programming session, a planning game session and a refactoring process, in particular. These cases illustrate how a ladder of reflection may promote one's comprehension of the relevant development process and may lead to insights that eventually may save time and money. Figure 1 presents one of the tasks presented in this lesson.

In what follows two situations are describes. For each of them:

- Identify potential topics for reflection.
- Construct a ladder of reflection.
- Describe what lessons you have learnt from this experience.

Case 1 – Testing in eXtreme Programming: In eXtreme Programming (Beck, 2000) programmers write tests before they write the code. This approach is called test-driven-development (TDD). Write a class Student according to the TDD approach.

Case 2 – New developers join your team: You are a team leader. You are asked to accept to your team two software engineers who have developed software that can be integrated into the project that your team works on. You ask your team and the two new developers to meet in the meeting room to discuss how to merge their code successfully.

Figure 1. A Task in the Human Aspects of Software Engineering Course: Construction of Ladders of Reflection

Lesson 9. Different Perspectives on Software Engineering: In this lesson we aim to illustrate how the profession of software engineering is shaped and that different approaches may influence the way it is eventually organized and established. Among other dual perspectives towards software development we introduce the agile paradigm vs. the heavyweight approach. Naturally, XP is mentioned among the other agile methods. It is explained that the agile approach towards software development has emerged in the last decade as an answer to the unique and typical problems that characterize software development processes.

In this lesson it is also illustrated how different approaches address failure and success of software projects. Among other approaches we cite Kent Beck who refers to the conceptual change towards software project success and failure that both developers and customers should adopt when they decide to use XP as their project development method (Beck, 2002). Indeed, the question of what a successful software project means invites many debates. It is emphasized that one agreement has been reached, though, among the entire community of software practitioners. They mostly agree that software projects should meet customer's needs.

Lesson 10. Abstraction and Other Heuristics of Software Development: This lesson examines different kinds of activities that are carried out during software development processes that are, in fact, heuristics (or ways of thinking) that one employs when one performs other activities. One of these ideas is abstraction.

With respect to XP, the students are asked to review the different XP practices and identify those practices that guide software developers to think in terms of different levels of abstraction when appropriate. It is suggested that developers' familiarity with the big picture of the developed application may improve their performance in the development of their specific tasks. One way to achieve this familiarity with the entire picture of the developed application is by the Planning Game. Though each developer is responsible for specific tasks, they all participate in the Planning Game. Their participation in the Planning Game enables them to become familiar with the entire picture of the developed application. In later development stages, when they have to make decisions related to different parts of the application, this kind knowledge may be useful. The main message of this part of the lesson is that the ability to think in terms of different levels of abstraction throughout the development process may contribute to and support software development processes.

Lesson 11. Characteristics of Software and the Human Aspects of Software Engineering: This lesson examines software characteristics from the developers' perspective and illustrates that even software characteristics that seem to be connected only to the software itself, cannot be isolated and detached from the software developers.

Specifically, the focus is placed on communication issues related to programming style. Figure 2 presents sample tasks discussed in this lesson which are connected to XP.

Example of a preparation task:

In Lesson 2 (about Software Development Methods) we discussed three software development methods: Spiral Model, Unified Process, and eXtreme Programming. Analyze each of these methods according to the software characteristics that each of them guides its production.

Example of a summary question:

Simple design (one of the eXtreme Programming practices) is a software characteristic.

- What does it require from the software developers to produce code according to this practice?
- Discuss connections between simple design and refactoring.
- How do these two practices support the eXtreme Programming values of communication and simplicity?

Figure 2. Tasks about Characteristics of Software, the Human Aspect of Software Engineering and XP

Lesson 12. The History of Software Engineering: In this lesson the history of software engineering from its early days in 1968 is outlined. One phase in this history is the entry of the agile methods in general and of XP in particular to the software engineering world. The uniqueness of XP – the specification of practices that implement values – is explained.

Lesson 13. Software Project Estimation and Tracking: It is well known that many software projects are late and over budget. This lesson explores the effects of overtime on programmers. In addition, it presents several methods of estimating and tracking time on task, so that the students will have additional tools to avoid being part of another late project. One of these tools is The Planning Game. Its main steps are described and the potential contributions of its different stages to software project tracking are examined and discussed with the students. When there is enough time, the Planning Game is played in detail for one iteration of a specific software tool according to the students' choice.

Lesson 14. Software as a Business: This lesson is about software as a business. It consists of two main parts: a brief account of how software became profitable, and more recent stories of making money with software. With respect to XP, its explicit attention to returning value to the customer with each release is highlighted.

4. CONCLUSION

In this paper we outline how in a course about human aspects of software engineering we use XP as an example of a software development method that emphasizes human aspects of software engineering. Specifically, we explain how XP is connected to each topic discussed in the course.

At the end of the semester the students discuss and analyze case studies related to the human aspect of software engineering. These activities aim to increase students' awareness, sensitivity and analysis skills when they participate in software development environments. The analysis of the case studies is based on theories that have been learned in the course. Not only students are presented with case studies. Rather, they are also asked to present case-studies from the professional literature or from their own experience in software development. The appearance of XP in these cases is optional. When XP does appear, its human aspects are highlighted.

5. REFERENCES

- Beck, K.(2000). *Extreme Programming Explained: Embrace Change*. Addison-Wesley.
- Beck, K. (2002). Extreme programming: an interview with Kent Beck, The Cutter Edge, <http://www.cutter.com/research/2002/edge020903.htm>, 3 September 2002.
- Brooks, P. F. (1975). *The Mythical Man Month: Essays on Software Engineering*, Addison-Wesley.
- Fowler, M. (2000) (with contributions by Kent Beck, John Brant, William Opdyke, Don Roberts). *Refactoring: Improving the Design of Existing Code*, Addison-Wesley.
- Hazzan, O. (2002). The reflective practitioner perspective in software engineering education, *The Journal of Systems and Software* **63**(3), pp. 161-171.
- 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.
- Hazzan, O. and Tomayko, J. (2004, in preparation). *Human Aspects of Software Engineering*, Charles River Media.
- Schön, D. A. (1983). *The Reflective Practitioner*, BasicBooks,
- Schön, D. A. (1987). *Educating the Reflective Practitioner: Towards a New Design for Teaching and Learning in the Profession*, San Francisco: Jossey-Bass.