Why is it hard to learn eXtreme Programming?

Philosophical and psychological aspects of learning a new methodology

Joseph Pelrine
Daedalos Consulting
Zürich, Switzerland
jpelrine@daedalos.com

Regina Uhtes
Dept. of Philosophy
University of Dortmund, Germany
regina.uhtes@gmx.de

Christian Noack
Daedalos Consulting
Witten, Germany
cnoack@daedalos.com

Abstract

"The practices that make up XP can be learned by anyone who has convinced someone else to pay them to program. That isn't the hard part. The hard part is putting all the pieces together, and then keeping them in balance" (see [1], p. 151). Although the new methodology of eXtreme Programming involves the use of many simple, intuitive practices, it includes other practices and techniques that go against the learners instincts and habits. eXtreme Programming must be learned, and the successful application of the methodology is largely dependent upon the ability of the developer to keep all elements and practices in balance throughout the whole lifecycle of the software. We maintain that the use of a coach is necessary to successfully learn XP, and to guarantee the balance in the development process.

eXtreme Programming (XP) is a new, lightweight methodology that has polarized the software development industry. It is built on highly pragmatic principles, resulting from the observation of numerous software projects that have succeeded (and many more that have failed). Many of the techniques and methods of XP are highly intuitive and, one would think, easy to learn. Nevertheless, XP is a radical approach to software development, and the discipline required to properly learn and apply the techniques and methods is itself extreme. We maintain that the use of a coach is necessary to successfully learn the methods of XP, and to guarantee the success of the development process.

Before we define the terms learning, method and coach, we would like to point out a basic difficulty in our endeavour: the definition of terms is a complicated and long-disputed problem in philosophy.

Nevertheless, it is essential (especially for interdisciplinary studies such as in this case philosophy, psychology and computer science), to agree on the definition of basic terms, in order to provide a common base for argumentation and discussion. We do not concentrate solely on an exposition of the broader concepts implied by a term, but look especially at the terms usage in the context of eXtreme Programming.

Method

A method is generally understood to be a systematic, standardized and planned procedure used to accomplish a task. This definition goes back to the translation of the Greek word *methodos*, which means "the path to" (hodos = path, meta = towards). What is meant is a consciously chosen way to solve a specific problem. This definition implies that a method is not simply any path, but rather a goal-oriented path.

The method as path seems to be an obvious and self-evident definition. A path has a beginning, a number of sections, with milestones along the way, it can be straight or curvy, but it leads to a goal. A method does not just describe a solution path, but rather implies a certain quality to its form. In our case, the formation of the path, or the so-called methodical path results from the principles and strategies of eXtreme Programming. More importantly, though, it results from the interaction between coach and the student - from the dialectically interpreted exchange occurring between persons in the leading role and assimilating role. This explanation touches upon yet another important aspect, which we will discuss later: the role of the coach in the XP learning process. First let us look at the aspect of learning per se.

Learning

Learning pervades our lives. Learning may be defined as a relatively permanent change in behavior that results from practice: a process that leads to changes in behavior and experiencing, or to new ways to behave and experience (see [3], p. 11). The term learning does not apply to ways of behaving and experiencing which are based on long-term biological processes such as growth and maturation, or which are caused by short-term organic processes such as fatigue or the influence of drugs. Learning results in

- in the cognitive realm: changes regarding knowledge, verbal and other skills, understanding, and the cognitive prerequisites for these
- in the behavioral realm: changes in various abilities, habits and actions
- in the emotional realm: changes in feelings, prejudices and values

The actual process of learning is not directly observable, but rather is deduced from the experiential observation of permanent changes in behavior. Irrespective of the manifold theories of learning, which will not be analyzed in this context, we propose the following basic principle:

Learning means to leave old ways and methods behind, to put aside old ways of thinking and to go beyond perceived limits! As G. B. Shaw (Major Barbara, Act III) says: "you have learnt something. That always feels at first as if you had lost something". The aforementioned coach

has the responsibility to help and support us in this process of change, to encourage us to venture beyond our perceived boundaries, and to assist us in changing old habits. (See below)

Which role do mistakes play in our learning process? Mistakes are essential for learning! The person who is ashamed of making mistakes is ashamed of learning. The biggest mistake is to constantly fear making mistakes. No one is immune to mistakes. The trick is never to make the same mistake twice. Unfortunately, one often runs into the opinion that a mistake at work is a "sin". (See [2], p. 8 f.) The requirement of throwing work away (when doing a Spike Solution, for example), is quite often not possible in a corporate context - how can one throw away work that the company has paid for? This behavior leads to developers being pushed into the defensive, where they have no courage to try out alternatives, to be creative - solely for fear of doing something wrong. In addition, the insistence on doing things in the "usual way" greatly increases the difficulty of finding a better solution.

Difficulties of eXtreme Programming

Having now defined the terms "method" and "learning" in the extended context of eXtreme Programming, let us know take a look at a few problems inherent in the XP approach to software development

As stated above, eXtreme Programming is a radical approach to software development. This approach, however good or relevant to the greater task at hand, leads to certain counter-intuitive techniques and methods that a developer must introduce into his development practice and process, exercising them until they become second-nature. Let us look at a simple example.

When the implementation alternatives (and their related costs) for a task are not clear, XP developers will implement a Spike Solution - a simple implementation addressing the problems in all layers of the software. One of the requirements of the Spike Solution is to unconditionally throw away the code developed for the solution.

Throwing away code is a big thing to request from a developer. We have mentioned above the difficulties in the learning process in cases where the corporate culture stigmatizes making mistakes, or (in our case) "building one to throw away". This is not the only problem, and not the biggest problem. A normal developer becomes attached to his code, to his product, becomes possessive, and extremely reluctant to let it go. The idea behind this requirement, though, necessitates unconditionally throwing away all Spike Solution code. Why?

Any one who has developed software long enough has experienced having worked hours solving a problem, only to have his computer crash, taking all the code with it. After grumbling and trying - mostly unsuccessfully - to salvage what remains of the code, the developer gives up and goes home, promising himself to put proper backup procedures in place the next day.

What normally occurs, though is that the developer arrives at work the next day having had a bright idea during the night, and rewrites the previous days solution in a fraction of the time, with significantly better quality. What has happened is that the subconscious has used the elapsed time as an incubation and ges-

tation period to feed the subconscious creative process, and has brought an optimized solution up to the level of the conscious mind.

The explicit use of the subconscious minds creative process is exemplary for the radical process that XP has become. It has taken something considered bad - i.e. losing code, and found the positive side to it. Unless one has had much experience in nurturing the creative process, though, one will be extremely reluctant to let the solution he has in the hand go, in order to get a yet better solution.

There are numerous other examples of techniques or methods in XP that, although useful, are very difficult for a developer to learn and accept doing on his own. Because of this, we maintain that XP can only be successfully learned by having an experienced coach available. Numerous critics support this hypothesis.

"On the other hand, I think that programmers who aren't as good as Beck and his colleagues will view this book as a license to be reckless. 'Analyze requirements? Bah - we'll just start coding.' 'Design the class hierarchy to be flexible? Bah - we'll just through some code down, then refactor it when we need to.' This might be productive for someone who has Becks insight into software architecture, and who has internalized good practices to the point where they are instinctual. Most of the programmers I know, however, would quickly find themselves in a morass of inconsistent class interfaces, passed-through parameters, and switch statements that don't quite cover all possible cases. I realize that XP practices such as prophylactic testing are intended to prevent this, but these have to be backed up by experienced judgement to be effective, and it is just such judgement that less reflective programmers lack." (see [5])

Coach

Nan-in, a Japanese master during the Meiji era (1868-1912), received a university professor who came to inquire about Zen.

Nan-in served tea. He poured his visitors cup full, and then kept on pouring. The professor watched the overflow until he no longer could restrain himself.
'It is overfull. No more will go in! '
'Like this cup', Nan-in said, 'you are full of your own opinions and speculations. How can I show you Zen unless you first empty your cup?'(from [6])

How does one find a good coach? What exactly is good in a good coach? What does this mean exactly? How can one tell a good coach from a bad one? What is an eXtreme Programming coach anyway? According to Kent Beck a coach is: "A role on the team for someone who watches the process as a whole and call's the team's attention to impending problems or opportunities for improvement" ([1], p. 177). His job "is to get everybody else to make good decisions" ([1], p. 73)

Which skills and capabilities does this difficult job require? Paraphrasing Beck, the following traits are deemed to be essential:

[1], p. 73-74: he is a good communicator, not easily panicked, is technically skilled, and confident. The role is very difficult, he needs be available as a development partner, to see long-term refactoring goals, to help programmers with individual technical skills, and to explain the process to upper-level managers.

[1], p. 145: The coach is responsible for the process as a whole, needs to remain calm, when everyone else is panicking, to know the ideals behind XP and their relation to the current situation, and works

best when he works indirectly.

A number of psychological factors need to be added to this. A major realization that a coach will make is that the learning process has a unique “gait”, which the coach must move in rhythm with if the process is to be successful. It is the realization that learning is the acquisition of new knowledge based on previous knowledge. This means that the developer brings a long and complex experience in learning and programming into the interaction called coaching. A result of this is that the developer will primarily judge the practicality of the proffered theories and methods for himself. Also, the developer has already attained a position and the associated status. The coach must consider all this. The coach must especially take into consideration the acquired knowledge and comprehensive skills of the developer. For ages, it has been a pedagogical requirement to pick up the student at the point on the learning curve where he currently stands. The tricky question is: where does the learner currently stand? This means that the practical realization of this requirement may be extremely difficult.

Special skills and capabilities (won through experience) are required in order to live up to the requirements of a good coach. On the other hand, it is the coachs job to start a process of leaving the well-beaten path!

The relationship in which the coach and the developer stand is crucial for the course and result of the learning process. The development of the relationship depends in part in the tactfulness and pedagogical skill of the coach, but mainly on the attitude and previous knowledge of the learner. Certain characteristics are required of the learner - not only for XP, but especially there. Especially in Pair Programming “there must be a great deal of mutual respect, tolerance, understanding [...]” (see [7]). Certainly not every developer is “blessed with the kind of strong communication and social skills” (see [4]). It is part of the learning process to build up these social skills as much as it is to teach the theoretical and practical aspects of a method.

The coachs responsibility is to accompany the whole XP-Process, to teach, to support learning, to supervise the process of learning, and to ensure the balanced introduction of the XP methodology and practices.

References

[1] Beck, Kent: eXtreme PeXtreme Programming explained. Addison Wesley Longman 1999

[2] DeMarco, Tom & Timothy Lister : Peopleware : Productive Projects and Teams, Dorset House 1999

[3] Hilgard, R. & Bower, G. H.: Theories of Learning. Prentice Hall 1981

[4] Rosenberg, Doug & Kendall Scott: XP: Cutting Through the Hype, in ’Objective-View’ issue 3

[5] Wilson, Gregory R.: How matters More Than What, review of eXtreme Programming explained, in Dr. Dobbs Journal #310, March 2000

[6] Zen Flesh, Zen Bones, compiled by Paul Reps, Penguin Books 1957

[7] Charles Ashbacher's review of 'eXtreme Programming Explained', at www.amazon.com

About the authors

Joseph Pelrine is an expert Smalltalk programmer with over 11 years extensive OT experience and is one of Europe's leading eXtreme Programming (XP) coaches. A former assistant of XP originator Kent Beck, he has led a number of successful XP projects. Mr. Pelrine is a former columnist for the Smalltalk Report and noted international speaker, and is currently a senior consultant with Daedalos Consulting in Switzerland. He is co-author of the forthcoming book, Mastering ENVY/Developer, due for publication from Cambridge University Press in fall 2000.

Regina Uhtes is currently studying philosophy, psychology, and German at the University of Dortmund, where she is a research assistant at the department of philosophy. Her research is focused on the field of ethics, evaluation of the consequences of technology, and organization diagnosis, and she is supported by the VW-Foundation as part of the research project "Teaching of Psychology in Europe".

Christian Noack is an experienced Smalltalk programmer who has been working professionally as software developer for over 5 years. He has been a consultant for Daedalos Consulting since 1998. In addition to his professional work, he is currently studying organizational psychology.