eXtreme Teaching

Roy Andersson and Lars Bendix, Department of Computer Science, LTH
{Roy.Andersson,Lars.Bendix}@cs.lth.se

***Abstract*— It sounds extreme to talk about getting higher quality at lower costs – and yet that is exactly what happens in eXtreme Programming. If it can be done for producing software products, we think it might be possible to carry over (all or most of) their techniques to teaching and get better teaching products (learning) at lower costs – simply by initiating eXtreme Teaching.**

I. Introduction

It is an inherent value in most teachers to strive for high quality in their teaching. On the other hand it seems like it is an equally inherent value in governments to cut down on the funding for teaching. Furthermore, with more students enrolling at university, they tend to have a more varied background. Faced with this situation something extreme has to be done.

In eXtreme Programming (XP) [1] they claim to produce software of high quality, on time, with less resources and fitting the customer's specific needs. This in contrast to more traditional development methods that deliver buggy software over budget, too late and often missing the customer's reel needs. We have been inspired by the techniques used in eXtreme Programming and have brought some of them into a teaching setting, thus initiating what could be coined eXtreme Teaching (XT).

In this paper, we lay out some of our own experience from trying out eXtreme Teaching and prepare the ground for a round table discussion.

II. eXtreme Programming

XP is a software development method that was "invented" by Kent Beck almost ten years ago [1]. It was created as a response to the many problems he saw in contemporary ways of developing software. Actually there is very little new in XP – Beck took a number of already known "best practices" and then really applied them to his daily work. His credo was (is) that if testing is known to be good, then let us do it all the time – if code review is known to be a good practice, then let us do it all the time.

In total he came up with a dozen different practices: Simple Design, Refactoring, Coding Standards, Metaphor, Test-First, Customer On-Site, Pair Programming, Collective Ownership, Continuous Integration, Planning Game, Frequent Releases, and 40-Hour Week. In the book where he first describes XP, he states that you have to adopt all the practices to be doing XP – if you miss one or more practices you are not doing XP. In his new book [2] he modifies that statement in the light of the past years' experience. He adds more practices to XP, but divides them into two categories. Primary practices that are safe and offer immediate improvement in the areas addressed, and corollary practices that should not be attempted without doing and mastering all the primary practices. In this paper, we will use the initial practices and their terminology, as this is what we knew and used when we worked with our ideas of eXtreme Teaching. However, there is very little difference as no practice has been revoked.

In addition to the practices, Beck provides a framework in which to carry out these practices. Two things are at the centre of XP: the customer and planning.

To be able to create a product that fits the customer's real needs (which changes) the team must be in close contact with the customer all the time. To allow good communication, the customer has to be in the same room as the team. This way it is easy for the team to communicate its progress through showing the effect of additions to the program to the customer and to get immediate feedback. Likewise, it is easy for the developers to ask questions to the customer when in doubt and for the customer to provide immediate feedback to these questions.

When it comes to planning, Beck claims that four factors are in play: time, scope, resources, and quality. It is essential that the developers have control over at least one of these – otherwise the customer will push for unrealistic estimates. So if time and resources are fixed (as is often the case in teaching), developers and customer will have to negotiate scope and quality. When the customer pushes for more scope, the developers should respond by saying that this would compromise quality and that they will not sacrifice quality. Plans can never be precise if they are long-term, so in XP they plan for small iterations of a couple of weeks' duration.

On an XP project, the customer writes a number of "user stories" to express what he wants the system to do. These stories are estimated by the developers after which the customer prioritizes the stories he wants to go into the next iteration. The developers then start implementing the stories and in parallel with that the customer writes acceptance tests and gets ideas for more user stories. At the end of the iteration, the team releases a new version of the system. In addition they also look at how much work has been done and adjust the estimate for how much work they can do in the next

iteration accordingly and a new iteration can begin. It is common on XP projects to have a first iteration where a small group of experts create the initial important architecture of the system after which more people are brought in to add manpower to the following iterations.

During the years, the XP ideas have been generalized into what is called agile software development methods, as expressed in the Agile Manifesto [3]:

> We are uncovering better ways of developing
> software by doing it and helping others do it.
> Through this work we have come to value:
>
> *Individuals and interactions* over processes and tools
> *Working software* over comprehensive documentation
> *Customer collaboration* over contract negotiation
> *Responding to change* over following a plan
>
> That is, while there is value in the items on the right,
> we value the items on the left more.

III. Transferring XP to XT

Just as XP is about software development "best practices", this round table is about teaching "best practices". The goal of XP is to achieve efficiency and quality in software production; that of XT is to achieve efficiency and quality in the production of learning.

XP works particularly well when you have to deal with vague or rapidly changing "requirements" – which is often the case in teaching. Especially when we have to deal with the fact that different students can have quite different ways of learning. As teachers, we know very little about our students' optimal way of learning – except from the fact that it is highly individual. So if we want to cater for all students we need to be extremely flexible and agile in our teaching.

Our initial interest in trying to carry XP ideas into teaching came from the fact that our department has been using XP as software development method in a course/project in software engineering [7] for the past four years. One of the authors is involved in this and noticed that the highly iterative nature of XP had many beneficial side-effects on the students' learning process. Each week during the project period the students try out XP in practice during a one-day programming lab. During the subsequent two-hour session they also have to reflect on what went wrong and what went well – and how to improve for the next iteration. This way of "teaching" has much in common with Kolb's learning cycle [4], where learning is achieved through active experience, reflection, conceptualisation and planning.

This gave us the idea that it might also be possible and beneficial to "translate" the individual practices of XP into corresponding teaching practices. As it turned out, some practices were easy to translate; some were difficult and others impossible. In the process of working out the "translations", we realised that we really needed to go through the underlying values and philosophy of XP, as seen from figure 1. So this is also the road we will take in the round table discussions.

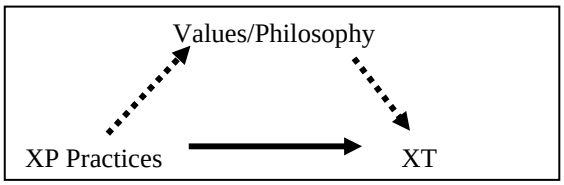

Figure 1. Going from XP to XT.

So what are the values behind XP? Fortunately, in the second edition of his book on XP [2], Kent Beck explicitly deals with these values. He says, that they are universal whereas the practices are situated and specific for software development – and probably even specific for some software development situations. The five values of XP are: Communication, Simplicity, Feedback, Courage, and Respect.

These values can be cast into a number of principles to guide your behaviour: Humanity, Economics, Mutual benefit, Self-similarity, Improvement, Diversity, Reflection, Flow, Opportunity, Redundancy, Failure, Quality, Baby steps, and Accepted responsibility. When we try to formulate XT practices, it will be helpful to keep in mind also these principles.

IV. Some XT-Practices

It is not the goal of this round table to discuss how to translate subjects such at customer, product and developer from XP to XT. We will take as given that the product is "student learning", and that the customer is in part the student (for receiving the product), and in part the government (for deciding the curriculum and economic sponsoring). The part of the development team is taken by teachers, assistants, and technical staff at the department.

From these definitions we have chosen to define and describe three XT-practices in this paper – Pair Teaching, Collective Ownership, and 40-Hour Week:

1) Pair Teaching

In the development phase of a course it is naturally to have someone to share ideas and to discuss different approaches with. This is as important in the actual teaching phase and in the maintenance phase between courses. Besides that people working solo are more likely to make mistakes, pair teaching is also preferable from a scholarly approach to teaching [5,6]. To be open with your teaching is a corner stone in the term scholarly, and it is as important to be able to develop your teaching as it is completely naturally to all researchers to expose their academic research to peer review to make progress.

From our own experience we prefer pair teaching at all levels of a course. All teachers, including the teaching assistants (TAs), on the teaching team should work in pairs

whenever it is possible; during course planning, lecture planning, exam construction, exam marking, exercises (labs), and even lectures. We think it is preferable to really work face to face as often it is possible, but working alternate on something also fits in the term pair teaching. Besides better quality, pair teaching also leads to a bonus back up flexibility – most things can be carried out solo even if planned to be carried out in pairs; if someone gets ill, gets double booked, has to attend a conference, meeting, etc.

2) Collective Ownership

Anybody on the teaching team who sees an opportunity to add value to any part of the course is required to do so at any time. This sounds obvious but to be able to obtain it requires all on the teaching team are taking responsibility for the whole of a course and it can only be done if all are both given the responsibility of their teaching and that they shoulder this responsibility. Of course, not all on the teaching team needs to know everything about all teaching activities on a course equally well, though everyone has to know something about all activities.

Two contrasts to collective ownership are no ownership and individual ownership. In the first model no one takes the responsibility for the whole. All involved people do their part as it best suits them without noticing whether it fits with other activities or not. The disadvantage of this model is quite obvious. In the other model all changes has to be approved by the official course owner. In this model the whole is stable, but it doesn't evolve as quickly as it could. Two results from strict ownership is that a course easily diverges from the teaching team's understanding since people are reluctant to interrupt the course owner, and people not given any responsibility takes less responsibility.

From our point of view and experience collective ownership is preferable to both no ownership and individual ownership. Collective ownership is also preferable from a scholarly approach to teaching.

3) 40-Hour Week

Overtime is a symptom of a planning problem. If a working week is precisely 40 hours is not terribly important. But no one can do several consecutive 60 hours weeks and still be fresh and creative and careful at the same time. The 40-hour practice allows overtime once in a while, but the practice states a second week of overtime is never allowed. Especially not for the same reason! The 40-hour practice is not only about minimizing mistakes and ensuring creativity. It is also about respect, respect of other people's time. Especially is it the department's responsibility to show respect for the employees' time by a reasonable planning. Then on the next level it is the teachers' responsibility to show respect for the students' time by a reasonable planning!

V. The Round Table Session

At the round table session we will open up for the participants to share their experience and opinions on both the XT-practices described above and on other XT-practices not described in this paper.

One practice in particular that could be very exciting to try to cast in a teaching setting is that of "Test-First". It sort of turns the normal way of teaching upside-down. The course starts with presenting an (old) exam – and then goes on to teach the students enough to make them pass the exam. The course ends when all students have passed the exam. Now that is extreme ;-)

References

[1] Beck K.: *Extreme Programming Explained – Embrace Change*, Addison-Wesley, 1999.

[2] Beck, K.: *Extreme Programming Explained – Embrace Change*, second edition, Addison-Wesley, 2005.

[3] *The agile manifesto*, http://agilemanifesto.org, 2001.

[4] Kolb, D. A.: *Experiental Learning: Experience as the Source of Learning and Development*, Prentice-Hall, 1984.

[5] Boyer, E., L., *Scholarship Reconsidered. Priorities of the Professoriate*, The Carnegie foundation, 1990.

[6] Antman, L., Booth, S., Hammar Andersson, P. and Olsson, T. *Excellent Teaching Practice – ett forskningsprojekt kring LTHs pedagogiska akademi*, 2:a Pedagogiska inspirationskonferensen, Proceedings, pp 14-16, Lund Institute of Technology, 2004.

[7] Hedin, G., Bendix, L. and Magnusson, B.: *Introducing Software Engineering by means of Extreme Programming*, in proceedings of the 25th International Conference on Software Engineering, Portland, Oregon, May 2003.