Coaching Coaches

Görel Hedin, Lars Bendix, Boris Magnusson

Department of Computer Science, Lund Institute of Technology, Box 118,
SE-221 00 Lund, Sweden
{gorel|bendix|boris}@cs.lth.se

Abstract. We have developed a tandem of undergraduate courses for teaching XP and coaching of XP teams. This paper focuses on the coaching course and the coaching practices we have developed. The tandem of courses enables us to give a challenging and interesting course for the coaches, and, at the same time, allows us to afford on-site coaches for the younger students, providing them with a high quality environment for learning XP. We also describe our experiences from the first instance of the courses and how we have tackled the bootstrapping problem.

1 Introduction

At Lund Institute of Technology, we have introduced extreme programming (XP) into the curriculum of our 4.5 year computer science and engineering program. The students are introduced to full XP in their second year, using XP as a vehicle for teaching basic principles of software engineering [5]. In the practicum part of this introductory course, the students run a scaled down extreme programming project during a 7 week study period. The project is run as 6 weekly iterations, each comprising one full day of production programming (a “long lab”), a 2-hour planning meeting, and 6 hours of individual experiments (“spike time”). The students take this course in parallel with other courses. Last year (2002), 107 students followed this course, and were divided into 12 teams of 8-10 students during the practicum part. The project scenario is that the 8-10 students are “newly hired staff” that ramp up the initial team consisting of a pair of senior developers (the coaches).

In addition to the basic 12 XP practices [1], we found two practices to be fundamental to teaching XP successfully. One is TeamInOneRoom, i.e., the whole team is present in the same room during the “long labs”. The other is CoachOnSite, i.e., each team has a coach that is present during the “long labs” and actively coaches the team. See [5].

How should one go about coaching such teams, and who should be the coaches? We needed to develop coaching practices and train people to the coaching role.

One alternative could be to use faculty as coaches. They are, however, very expensive and, furthermore, their role is easily perceived as “teaching”, i.e., examining the team rather than a leadership role that is part of the team. Another alternative would be to let one of the students on the team take on the role of a coach, but we never considered this alternative since these students are, in general, too inexperienced. We settled for a third alternative, namely to run a second course, a coaching course, for senior undergraduate students, and let them coach teams in the introductory course. In addition to allowing us to afford on-site coaches, this provides us with an opportunity to give an advanced software engineering course where the students can make use of their team for practicing leadership, practicing software architecture issues, and learning about agile methods in more depth. To train the coaches, the coaches are themselves coached by faculty.

The rest of this paper describes the structure of the coaching course, the coaching practices we have developed, and the bootstrapping process of running the first instances of both courses. Finally, we draw some conclusions and reflect on how our practices might apply in industry.

2 The coaching course

The coaching course is a one-term pass/no-pass course corresponding to around 1/3 time studies (the students take other courses in parallel). It consists of three parts: a theory part followed by a practicum part in parallel with an in-depth study.

The theory part consists of seven 2-hour lectures and discussion seminars. There are also some short lab sessions for trying out techniques and tools. Topics include design and architecture covering test-driven development [2], design patterns [6], and software architecture based on FirstIteration [8], patterns generating architectures [3], and multiple views [7]. We also review the basic XP practices, and go into more depth on specific topics, in particular testing and planning. There is also some theoretical material on coaching as such, although there is not much literature available yet on this subject, and the actual coaching is taught mainly through the practicum part. The theory part ends with a one day lab session where the coaches develop the first iteration of the product in pairs. This first iteration is then used as the starting point for the teams in the introductory course.

The practicum part is the actual coaching where the coaches take care of teams of students from the introductory course. The coaches lead the planning meetings and they coach actively during the one-day long lab iterations. They also have the responsibility of tracking the evolving architecture of the product and to document it in the end. The coaches meet once a week together with a faculty supervisor in order to share experiences from their coaching, to get advice, and to reflect on their own role.

During the practicum part, the coaches also perform an in-depth study of a specific topic of their own choice. Each study is documented as a small report written in article style. Examples of selected topics include pair programming experiences, refactoring-aware versioning, and surveys of agile methods. Many of the coaches take advantage of their team in the study for conducting small experiments and surveys.

3 Coaching practices

We have run one instance of both the introductory course and the coaching course during the study year of 2001/2002, and are currently (March 2003) in the midst of the second instance. During the first instance of the course we have developed a number of coaching practices that are discussed briefly below.

Coach

The coach should know the basic XP practices and be able to explain and motivate them to the team. Being a slightly more senior student, the coach also sometimes has technical knowledge that is of help to the team. However, the coach does not know everything, so the main role of the coach is to help the team to; learn the XP practices, develop the practices, learn to communicate, and to help them learn from each other. In essence, the goal of the coach is to help the team to help themselves. This is done by being an ActiveTrackingCoach and by TriggeringCommunication.

ActiveTrackingCoach

The coach is active in the sense that he/she actively follows the activities of the team, and in practice works as a tracker. The coach keeps track of what each programming pair does, how their work progresses, and how the code evolves. This allows the coach to act proactively, already when problems start to arise, rather than after the fact. If the coach spots a problem, it can often be solved by TriggeringCommunication between team members.

TriggerCommunication

In a team learning XP, communication within the team does often not arise spontaneously, but needs to be learnt. Initially, the team members do not know each other well and are hesitant to communicate. Also when they know each other well, they may be hesitant to ask for help because they are not yet a functioning team. A typical sign is that a pair does not ask for help when they have gotten stuck on some technical problem. Sometimes, the coach can help to solve the problem, but more often, the coach can explain that it is a practice in this case to ask other programmers on the team for help, or to switch pairs. Another common case when communication is needed is when two pairs are working on the same code without being aware of this. If a problem is of relevance to all of the team, the coach can take a TimeOut.

TimeOut

For any kind of problem that occurs in the team, if the coach cannot solve it easily and directly, a good way of tackling the problem is to take a TimeOut, i.e., to bring the team together for a brief stand-up meeting, and let the team solve the problem together. This works for technical problems, where someone on the team might have skills in a certain area, or where the team can brainstorm possible solutions or ideas for experimental spikes for solving the problem. A TimeOut is also useful for process problems where the team needs to agree on what practices to follow. Basic XP practices are given by the course material, but the team needs to reason about these practices in order to understand and accept them. There are also lots of little subpractices that need to be developed by the team itself. Often such subpractices emerge spontaneously and can be turned into a more permanent practice at a TimeOut. As an example, several teams developed the practice of switching pair programming partners after lunch. This practice had the advantage of getting the team members to know each other in a short time, and turned out to be logistically easier in the scaled down project than the normal XP practice to switch partner after the completion of a task or story.

PairCoaching

Usually, the students coach in pairs. While this might be inefficient in a commercial setting, it has many benefits in an educational environment. The coaches appreciate having someone to share the coaching with. It also allows a greater number of students to take the coaching course, a greater flexibility in the exact number of coaches needed, and finally it gives a redundancy that is good to have in case of temporary illness, since the coaches are crucial to the team.

CoachIsCoached

Each iteration, between the long lab and the next planning meeting, all coaches meet for 2 hours of reflection and advice. During this meeting, the coaches share their experiences from the previous long lab, and many good ideas emerge among the participants on how to handle various kinds of problems. These meetings are coached by faculty who also give advice on what to focus on during the next iteration. For example, the first iteration might focus on just getting started; the second one on team building, e.g., by switching pairs often; the third one on making sure that a swift release process is developed; the fourth one on checking extra carefully that testing is done in an appropriate way; etc. What to focus on need not be fixed in advance, but can be a reaction to what kind of problems that arose in the previous iteration.

CoachIsArchitect

The students taking the introductory course have previously studied programming, algorithms and data structures, and object-oriented modeling and design, but have previously only had assignments of small programs of a few hundred lines of code. They are thus quite inexperienced programmers. The product to be developed by the team is, in general, their first larger program, and there is a need for someone on the team to take a more senior role concerning the software architecture. We solve this by letting the coach take on the role as an architect as well.

During the project, the coaches have the responsibility of taking the initiative to architectural discussions during the planning meetings, they keep track of the evolving architecture, and document it at the end of the project. This allows the coach to practice software architecture in a realistic, albeit small, project.

The architect role here is very different from the traditional one of prescribing an architecture before implementation. Here, the architect role is rather to take the responsibility of keeping track of the architecture, verbalizing it and maybe visualizing it to allow it to be communicated among team members and, if needed, to take the initiative to architectural refactorings.

4 Bootstrapping the courses

The description of the course above refers to the second instance where the coaches have already taken the introductory course and are familiar with the basics of XP. During the first instance, we did, however, not have coaches that were experienced with XP. In order to bootstrap the courses we had to train a first batch of coaches. To this end, the persons following the first coaching course instance was a combination of faculty staff, graduate students, and a few handpicked senior undergraduate students. This allowed us to train staff at the department as well as try out the course on a few student coaches.

This course instance used the same format as described above, with a theory part, a practicum part, and an in-depth study. However, the contents of the theory part were different, focusing on introducing the basic XP practices.

The first instance of the coaching course also served to bootstrap the introductory course. The theory part of the coaching course was then given prior to the theory part of the introductory course, and much of the lectures and short lab material (e.g., on version control and unit testing) could carry over from the first coaching course to the introductory course.

In the second instance of the coaching course, the situation is more like we planned. The course is a part of the regular curriculum and essentially all participants are undergraduate students, although the course is open to graduate students and interested faculty as well. The coaching students already know the basics of XP and we now take the opportunity to bring in more advanced material into the theory part. In particular, we found it beneficial to add a strong focus on architecture.

In the first instance of the coaching course the coaches developed the initial architecture in the first iteration, but many did not track the evolving architecture very closely, and later became somewhat frustrated that they did not know enough about the actual implementation. Their experience was that this made it more difficult to coach. This was one reason for developing the CoachIsArchitect practice and we expect that the active work with the architecture will allow the coach to stay in touch with the code although he/she does not do any production programming, thereby making the coaching itself more interesting and of higher quality.

The second instance of the introductory course is essentially the same as the first one. There are, however, some minor changes, such as adding a new short labsession on refactoring tools, using the Eclipse Java development tools [4]. Since this is new material to our new coaches too, we have included that in the current coaching course as well. This way, we can continually make use of the coaching course to try out new teaching material in a small group before running it in the large introductory course. We expect such smaller changes to be useful also in future instances of the courses, as the field and practice of XP evolves.

5 Conclusions

The idea to have older students studying coaching by actually coaching younger students, is in our experience a fruitful solution and, to our knowledge, has not been tried before. This tandem of courses enables us to give a challenging and interesting course for the coaches, and, at the same time, allows us to afford on-site coaches for the younger students, providing them with a high quality environment for learning XP.

In this educational setting, there is a need for a streamlined process for learning how to coach, and development of coaching practices. We have described the main practices that we have developed to this end: Coach, ActiveTrackingCoach, TriggerCommunication, TimeOut, CoachIsCoached, PairCoaching, and CoachIsArchitect.

Although we have developed these practices for student coaches, we believe that many of them are useful also in an industrial setting: Coach, ActiveTrackingCoach, TriggerCommunication, and TimeOut, are all general coaching practices that have to do with learning communication, and about learning and evolving the process. The essence of the CoachIsCoached practice is the reflection and sharing of experience concerning the actual coaching. This should be useful also in industrial settings where people are learning to coach. The PairCoaching practice is mainly motivated by the educational setting, but might be useful when introducing XP in industry as well.

The CoachIsArchitect is also mainly motivated by the educational setting where the developers are inexperienced programmers and where the combination of architecture with coaching provides more depth and challenge to the coaching course. However, we believe that also in many cases in industry, it is natural for project leaders to combine these two roles of coach and architect. In particular, the tracking architect role (rather than the traditional prescribing architect) that we envision should be of use for project leaders that wish to stay in close touch with their agile project.

The coaching role still has the association of self-learned guru in the XP world. Our experience shows that it is quite possible to teach coaching in a systematic way, even to fairly inexperienced programmers. The coaching practices we have developed to this end will certainly evolve as we get more experience, and should be thought of as protopractices that need to be further elaborated and evaluated. Nevertheless, we have tried them out with success and believe that other educators will be able to use the same techniques. The coached teams were very satisfied with their student coaches, and the coaches themselves found the course very interesting and stimulating.

Acknowledgements
We are in great depth to all the faculty, graduate students, and undergraduate students that followed the first instance of the coaching course, in particular for all ideas and feedback that surfaced at the coaching meetings.

References

1. K. Beck: *Embracing Change with eXtreme Programming*, IEEE Computer, 32(10): 70-77 (1999).
2. K. Beck: *Test-Driven Development*, Addison Wesley, 2002.
3. K. Beck and R. Johnson: *Patterns Generate Architectures*, in proceedings of ECOOP 1994: 139-149. LNCS 821. Springer Verlag.
4. *The Eclipse Java Development Tools subproject*, http://www.eclipse.org
5. G. Hedin, L. Bendix, B. Magnusson: *Introducing Software Engineering by means of Extreme Programming*, in proceedings of the International Conference on Software Engineering, Portland Oregon, May 5-8, 2003.
6. E. Gamma, R. Helm, R. Johnson and J. Vlissides: *Design Patterns: Abstraction and Reuse of Object-Oriented Design*, in proceedings of ECOOP 1993: 406-431. LNCS 707. Springer Verlag.
7. P. Kruchten: *The 4+1 View Model of Architecture*, IEEE Software 12(6): 42-50 (1995).
8. W. C. Wake: *Extreme Programming Explored*, Addison Wesley, 2001.