Filleting XP for Educational Purposes[*]

Friedrich Steimann[1], Jens Gößner[2], Thomas Mück[2]

[1]Universität Hannover, Institut für Informationssysteme, Appelstraße 4, D-30167 Hannover
steimann@acm.org
[2]Universität Hannover, Learning Lab Lower Saxony, Expo Plaza 1, D-30539 Hannover
{goessner, mueck}@learninglab.de

Abstract. Rather than teaching XP as a software development method, we have found that some of XP's core practices are actually viable learning scenarios. By combining these practices into a set of regulations, we have organized a well-received 200 h software practical regularly conducted during the 4th semester of an applied informatics curriculum.

1 Introduction

While learning how to program can be fun, teaching how to program is notoriously difficult. This is particularly so because programming is best learnt by doing, not by listening (or watching), thereby reducing the role of the docent to the one who assigns the right exercises (with the tutorial assistance being provided by others). Learning by doing, however, is not learning on one's own: it requires rapid feedback and gentle guidance.

Whereas the syntax of a programming language can always be internalized through trial and error (with the immediate feedback being given by the syntax editor or compiler), checking the semantic correctness of a program requires a much deeper understanding. This deeper understanding will usually be that of a peer; however, if the peer has sufficiently clear expectations of how the solution should perform, these expectations can also be cast into a test suite, making their accessibility independent of that of the peer.

Learning in small groups can be highly effective. Students can learn from each other by positive and negative example, they can join efforts to attack difficult problems, and they can learn by teaching what they have not fully understood for themselves. In addition, students are likely to have more time, to be more patient, and to have a better understanding of each other's problems than their teachers. All this makes programming in pairs seem a favourable setting for learning how to program.

It appears that test driven software development and pair programming can actually help with learning how to program. Because they are key practices of XP, what was more obvious than testing other XP practices for their pedagogical usefulness? In the following, we take a first look at the suitability of XP practices for teaching, and discuss some problems together with how they can be solved. A systematic investigation of the general aptness for XP in education however is yet to be undertaken.

[*] in: M Marchesi, G Succi (eds) *XP 2003 – Proceedings of the 4th International Conference* (Springer 2003) 414–417.

2 A practical scenario for learning how to write software

In order to teach programming at a university in an industry-like setting, we make our students form small "companies" (teams of six) and have each company develop a 150 person day software project. Progress and equal distribution of work are controlled by dividing the project into blocks and each block into a number of individual tasks assigned to the individual team members. At the end of each block, the solutions to the individual tasks must be integrated and turned in as one functioning program. Teams have to meet once a week for two hours, to assign tasks and perform walk-throughs (code reviews). Because programming abilities vary widely, we encourage participants to code in pairs (solving the double number of tasks per pair).

Pair programming fosters collaboration and mutual assistance. Successful collaboration depends on many factors, personal sympathy (or antipathy) being one of them. While students find it natural to collaborate with their mates, most feel uncomfortable with co-operating with strangers, most likely because they fear exposure of their deficits. In practice, however, good (i.e., productive) collaboration even with colleagues one dislikes is indispensable. Therefore, one of the key abilities to be learnt by a programmer is a social one: being able to co-operate. We teach this ability by requiring the pairs of a company to rotate after each block.

User stories present adequately sized tasks. In order to be able to guide students and track their progress, tasks should be cut down into chunks of small, manageable size with clearly defined outcome. On the other hand, individual tasks must be big enough so that some progress can be experienced. As it turned out, we had designed our tasks so that it took each pair an average of 25 h (or approximately three working days) per task. Thus it appears that a typical user story has about the right size for a single task. From hindsight, it would have been a good idea to present tasks as user stories, giving them a more realistic flavour. Hints on the solution of each task could then be offered as separate help (displayed from within a specially adapted development environment; see below).

Test first enables a constructivist approach. XP promotes the test-first approach as one of its core practices, requiring that tests are implemented before production code is entered. Having the tests in advance allows one to experiment with possible solutions, find out how they failed, and try alternatives. If students are freed from writing the tests themselves, the so-modified test-first approach presents a constructivist learning environment, the obvious downside being that the tests must be written by someone else, namely the instructors. While following this approach students do not learn how to write tests, they learn to appreciate the existence of tests. A setting for learning how to write tests is presented in the next section.

Continuous integration facilitates frequent submission of solutions. One problem with long-term exercises is that students tend to get lost. If the deadline is far ahead, there is only little pressure to proceed with one's work. Heterogeneous groups embracing members with different work attitudes are likely to fall apart: if some members are more determined than others, they will rather take over the work of their colleagues than wait while the deadline slowly approaches. We counteract this cause of disintegration by introducing blocks, entailing short delivery cycles.

At the end of each block, students have to turn in their solutions as a whole, i.e., they must submit the current state of the project as developed by their group by uploading their project files to the server on which they must compile and pass all tests. Although a daily build is not mandatory, the continuous integration promoted by XP is likely to prove a practice helping to meet delivery deadlines without worry.

3 Practical problems and their solutions

Problems with pairing. We found that although students were enthusiastic about it initially, actual pair programming times were much shorter than anticipated: only one third of all tasks were actually solved in pairs. Even though students appear to be open to pair programming, its actual acceptance is disappointingly low; as one student put it, "pair programming only makes sense […] with better opportunities to meet and work together at the face". It seems that meeting is a severe obstacle to (co-located) pair programming.

To facilitate pair programming a number of students (not all, since we conduct a controlled experiment [5]) will be equipped with notebook computers with Internet access enabled via WLAN (802.11b), Ethernet cable and modem. MS NetMeeting is used as the basis for application sharing and voice communication. WLAN access points are distributed over parts of the campus and computer science buildings. Cooperation in peer-to-peer mode is also possible and the connection of choice if participants reside in close proximity, even side-by-side. It remains to be seen, however, if the ubiquitous possibility for pair programming will actually improve the acceptance of this mode of teamwork.

How to test the tests. Naturally, with the tests being provided students learn to appreciate the existence of tests, but they do not learn how to write them. Therefore, our curriculum of exercises must contain tasks dedicated to the writing of tests.

Following the test-driven approach to practicing software development, the tests (as a product) must be tested. Naturally, tests are tested by the application they test: if the application contains errors, the tests must find them, and if the application is error free, the tests must approve this fact. Thus, the first and most obvious approach to testing the tests is to also write the application they are testing, to introduce errors (intentionally or accidentally), and to correct both tests and application until they conform to the specification.

If writing the test is the task, however, then testing it afterwards (even if by writing the application) suffers from the same (psychological) problems as known from conventional testing: the tester is blind towards his/her own errors. Therefore, we provide with each test-writing task a set of solutions, one with no defects, the others with errors injected. A test suite is considered correct only if it finds all errors and lets the correct solution pass.

4 Discussion

Distributed pair programming. Distributed or dispersed XP (DXP) is a relatively new facet of computer-supported collaborative work (CSCW) that is increasingly being used across XP-style software developing companies. In DXP, programmers collaborate using voice communication and application sharing software, typically MS NetMeeting. First experiences with this setting in an educational context have been reported in [1]; our own trials are promising enough to let distributed pair programming appear a viable alternative to co-location.

Automatic verification of exercises. The idea of automatic verification of programming exercises has previously been put forward by Praktomat [3] and WebAssign [4], two publicly available frameworks for the conduction and evaluation of exercises. However, these systems are not integrated in a practical of our size.

5 Conclusion

Perhaps, with most of the alleged advantages of XP yet unproven, it is still too early to teach XP as a state-of-the-art programming method [2]. But if didactically valuable learning scenarios happen to coincide (or at least blend smoothly) with XP practices, then this should be sufficient justification to practise these practices in teaching. As with XP as a whole, their proliferation will depend of the personal experiences the students make, and on how successful they are.

Acknowledgements

This work has been supported in part by the BMBF Förderprogramm Neue Medien in der Bildung, Ausschreibung "Notebook-University", Förderkennzeichen 08NM222A, and by the e-Learning Academic Network Niedersachsen (ELAN).

References

1. P Baheti, L Williams, E Gehringer, D Stotts, J McC. Smith *Distributed Pair Programming: Empirical Studies and Supporting Environments* Technical Report TR02-010, Department of Computer Science, University of North Carolina at Chapel Hill (USA 2002).
2. P Becker-Pechau, H Breitling, M Lippert, A Schmolitzky "An Extreme Week for First-Year Programmers" in: *XP 2003 – Proc. of the 4th International Conference* (Springer 2003).
3. www.infosun.fmi.uni-passau.de/st/praktomat/
4. http://niobe.fernuni-hagen.de/WebAssign/
5. F Steimann, J Gößner, U Thaden "Proposing Mobile Pair Programming" *OOPSLA 2002 Workshop on Pair Programming Explored / Distributed Extreme Programming* (Seattle, USA 2002).