Nordic Journal of Computing 9(2002), 261–265.

SUMMARY OF THE SUBWORKSHOP ON EXTREME PROGRAMMING

LARS BENDIX
GÖREL HEDIN
Department of Computer Science, Lund Institute of Technology
Box 118, S-221 00 Lund, Sweden
{Lars.Bendix|Gorel.Hedin}@cs.lth.se

1. Introduction

This report summarises the results of a subworkshop on eXtreme Programming (XP) held as part of NWPER'2002 in Copenhagen, Denmark, August 2002. NWPER usually includes a number of "interactive subworkshops" such as this one, where the format is based on focused discussions rather than on the presentation of papers.

The aim of this subworkshop was to gather people from academia and industry to discuss and share experiences from using extreme programming at university and in companies. Before NWPER'2002, the authors had been involved in a software engineering course and project at the university of Lund where XP was used. We were curious to know whether our particular experiences were also valid in general. Did other universities have similar experiences? Were the results specific to student projects – or would they apply to industry projects too?

Results from the XP projects in Lund made up the starting point for the subworkshop and three sub–themes were defined to structure discussions:

- XP and testing,
- XP, refactoring and configuration management,
- XP in university education.

The subworkshop was organised as a three hour session of interactive discussions chaired by the organisers and each theme was started by one or two short presentations to kick off discussions.

2. XP and testing

For the short presentations, Anders Nilsson from Lund University gave an overview of his work on acceptance tests. He had made a prototype tool – JaTAck [Nilsson 2002] – that is capable of automating the execution of specified acceptance test, much the same way that JUnit [Beck and Gamma 2002] can automate the execution of unit tests.

Then Per Madsen from Aalborg University presented his work relating traditional testing and XP testing. Unit test is quite similar to traditional black box testing and bad design of a program can make both types of tests very hard to do.

His experience from introducing XP testing to students and industry was that people expected full automation and confused automatic testing with automated execution of specified tests – which is what XP test tools provide. Furthermore, the general attitude was that the XP approach to testing looked great – but not for the stuff that we are doing.

In XP everything revolves around the tests. There are two types of tests: unit tests that are written by the developers and acceptance test that are written by the customer. Both types of tests are written *before* the code is written and therefore also act as a way to specify the intended behaviour of the program. For the customer to ensure that the application satisfies his requirements; for the developer to ensure that his detailed implementation works.

The more detailed unit tests should guarantee that there is no functionality without a test to ensure that it works properly. It is the philosophy in XP that *all* unit tests should be executed very frequently – even several times every hour. Obviously unit tests written before any code is written will fail. However, the XP reason for writing code is then to look at a failed unit test, figure out why it failed and what code needs to be written to make it pass – and then run *all* unit tests again. This is repeated until no unit test fails, at which point the code works as specified.

This way of working also gives security – and courage to change the code. If the implementation of a functionality is changed, we run all unit tests and when no unit test fails, the changed implementation works correctly. If a functionality is added, we add some unit tests, run *all* unit tests and when no unit test fails the new functionality works *and* all the existing functionality continues to work. If a bug is found, we add some unit tests to catch the bug, run *all* unit tests and when no unit test fails, the bug has been removed.

Experience from the projects in Lund showed, that students did not always write tests first and in some projects wrote only few unit tests. We believed the reasons to be that the students lacked training in writing unit tests the right way and that they could not see the long–term benefits. However, during the discussion it turned out that it was not only students that were reluctant to use and see the benefits of unit tests and especially test first. In a Danish company, unit tests were hardly used at all and almost never were tests written before the code. Instead *acceptance* tests were used as the tests to program against when writing code. Some of the arguments were that in real life it is difficult to write tests before code, because you do not know what you are really looking for. Also, sometimes code and/or functionality is thrown away together with all unit tests. Such problems could be caused by not having a clear distinction between doing spikes – where XP does *not* mandate testing and test–first – and consequently writing the production code.

3. XP, refactoring and configuration management

Torbjörn Ekman from Lund University had studied how students carry out refactoring of their code on XP projects. Often students spent far too much time patching the code before finally deciding to refactor it. That might in part be caused by the fact that students did not get all user stories from the start, but rather from iteration

to iteration. This, together with the relatively short life–span of the project, could have given the students the hope that every little patch was the last one.

Most groups, though, did carry out refactoring to some degree. Small refactorings were not troublesome, but on a couple of occasions big refactorings caused problems as they took too long time and therefore blocked the progress of the rest of the group. Refactorings were always decided – and designed – by the group, while the actual implementation was always done by a pair. A lesson to learn could be that big refactorings should be done by the group all the way through to implementation.

XP says that refactoring does not add any new functionality. It improves the design and the code by making it simpler. From the discussions it emerged that most people saw refactoring as a painful process, but something that has to be done. XP gives the advice that refactoring be done in many small steps, however, no–one had actually done that. People agreed that better tools for moving code – other than simple cut–and–paste – are needed. Such tools should not be line–based but rather syntax–based.

Ulf Asklund and Lars Bendix from Lund University had studied how CVS is used by students for simple configuration management (CM) on their XP projects. XP does not say anything specific about CM, but the practices of small releases, collective code ownership, continuous integration and to some degree refactoring can in fact be supported by a CM tool. In this respect, CM is seen as a way to co–ordinate the collaboration in a team of programmers [Asklund and Bendix 2002], rather than the more traditional view of CM with lots of paperwork and red tape.

The students found CVS to be indispensable for the success of the group's efforts. However, it turned out that very few had ever retrieved an older version of a file. Branches were hardly ever used and then only for the release process. In general, tags were not used and in some groups not even for releases. There was only sporadic usage of "edit" to flag to others that a file was being worked on. In fact, the only functionality of CVS that was extensively used was "update" and "commit" to automatically synchronise a pair's workspace with the repository. For that task the students considered the merge support extremely helpful, both the automatic merges and the flagging of merge conflicts. So in reality students considered CVS a merge tool more than anything else.

People from the audience used CVS as well and liked it too for the support for co–ordinating people or pairs working in parallel. One company, however, had experienced the limitations of CVS. For more advanced support, like variants and several parallel branches, CVS is not sufficient.

4. XP in university education

Lars Bendix and Görel Hedin presented results from the XP–course/project in Lund. Students following the course and project were on their second year and had previously taken courses on algorithms and data structures, programming, and analysis and design. They were given a seven week introductory course with one lecture a week. Lectures featured an overview of XP, testing and pair programming, configuration management, design and architecture, and planning and estimation.

The last lecture was divided up into a one hour quiz test to ensure that students had read sufficiently to start the project and one hour to introduce the coming project.

During these seven weeks there were also three compulsory two hour labs on extreme hour, testing and pair programming, and configuration management. The main literature for the course – and project – was a book [Jeffries *et al.* 2001] and a paper [Beck 1999] on XP. The book is hands–on and easy to read in a positive sense, which means that students can – and did – re–read a chapter on an XP practice in 5–10 minutes.

For the project, the approximately 110 students were divided up into 12 groups of 8 or 10 students. Each group was allocated one whole day in a computer lab with access to 6 computers. Students were assigned to groups randomly and had appointed a coach to follow the group. Coaches were recruited from senior people, PhD students and senior students at the department. All coaches had followed a special coaching course.

The project lasted for seven weeks and presence was compulsory. There were six iterations and a final evaluation. Each iteration started on a Tuesday with a two hour meeting of coaches, customer and project manager. On Wednesday each group had a two hour planning meeting. Until the following Monday each student was supposed to spend six hour on spikes, either alone or in pairs. Monday was the working day where groups met at 8 AM and worked together until 5 PM.

During the six iterations they made three releases of the product. The first was a trial of the release process, the last two were releases to a peer group. Releases included not just the executable code, but also source code, documentation and user manual. Based on the final release, the peer group had to do a 15 minutes presentation where they evaluated the quality of the released product including the user manual. Furthermore, they should evaluate the ease of picking up the other group's product – source code and documentation – and continue developing it.

We found that the coach's role is important. He has to be pro–active to anticipate problems and call for time–outs whenever necessary. Stand–up meetings were a time–effective way of disseminating information and whiteboards were good use in co–ordinating tasks. We discovered that the intended six weekly hours of spike time was not always used. This could be explained by the somewhat rigid and artificial division between programming time (Mondays only) and spike time imposed by the schedule.

A university in Finland had tried using XP on student projects too, but with very poor results. Their set–up had, however, some characteristics that were different from the project at Lund University. The coach was only there to consult on the initiative of the students. There were no fixed hours of working, which meant that students had problems in deciding when to pair up for work. This meant that they ended up working as individual pairs and not as a group. In Lund they had fixed hours and a room to stay in.

Other people with experience from using XP in teaching pointed out that cultural differences could be a source of big problems. It was not just a matter of cultural differences between university and industry – even companies have different cultures and not all are equally well suited for XP. In homogeneous cultures and cultures with little competition between people there could be few problems

in practising the tight collaboration and sharing inherent in XP. However, in a culture as varied and competitive as the American, there could be real problems with people from one ethnic group refusing to work with people from another ethnic group. In such conditions much care would have to be taken in assigning students to groups.

References

ASKLUND, U. AND BENDIX, L. 2002. A Study of Configuration Management in Open Source Software. *IEE Proceedings – Software 149,* 1, 40–46.

BECK, K. 1999. Embracing Change with eXtreme Programming. *IEEE Computer 32*, 10, 70–77.

BECK, K. AND GAMMA, E. 2002. *JUnit Cookbook.* Available at `http://junit.sourceforge.net/doc/cookbook/cookbook.htm`.

JEFFRIES, R., ANDERSON, A., AND HENDRICKSON, C. 2001. *Extreme Programming Installed.* Addison–Wesley.

NILSSON, A. 2002. JaTAck – Java Acceptance Testing Tool Ackronyme. In *Proceedings of the NWPER'2002 Workshop*, Copenhagen, Denmark.