Teaching And Learning XP

Frank Keenan
Department of Computing
Dundalk Institute of Technology
Dundalk
Co. Louth
Ireland
frank.keenan@dkit.ie

Abstract

This paper reports on the results of a survey conducted on 57 National Diploma Software Development students at Dundalk Institute of Technology. The students had studied programming for two to three years. A development exercise was carried out over 2 months using Extreme Programming (XP) in their Software Engineering subject. The survey investigated student attitude to some of the 12 practices of XP. In addition it also provided feedback on the class attitude to the learning experience.

Keywords

Extreme Programming, XP, Pair programming, Test first, Refactoring

1 INTRODUCTION

It was decided to choose a moderately easy problem to start with and then change the requirements or add new functionality to make it more complex. The project would allow students to investigate Pair Programming, Testing, Simple Design, Refactoring and Standards. Most tasks were conducted in the pair environment.

2 METHODOLOGY

The problem chosen was the income tax calculator from Software Engineering; Theory and Practice [1]. The class was divided randomly into groups of 4 and within that into pairs. Before beginning each group was required to agree a set of standards for implementation and all reports. One pair was then given the problem and requested to produce an algorithm and then implement a solution. The other pair was required to plan test cases for the solution. This ensured that tests were prepared in advance of programming.

Programmers did not test their own code but the tests were prepared by members of the same group. It was felt that the students would learn more from the testing tasks if they did not have the luxury of testing their own code. For the next set of requirements the roles were reversed. This continued for a number of exercises until the last exercise required some refactoring to ensure that the code was more reusable.

3 FEEDBACK AND EVALUATION

In most cases the pairs within the 4-person-group were very competitive. The testing pair seemed to get a great deal of satisfaction on finding errors in the solutions. The reaction was certainly more animated than that of students finding an error in their own code. The programming pair seemed to take more pride in their efforts.

3.1 Pair Programming

Studies of pair programming have shown that programming performance has improved. Results, for example, show a reduction in defect count of 15% [2]. By completing pair programming tasks it was hoped that participants would "get the benefit of learning knowledge from other programmers" [3].

33 students (58%) indicated that they would prefer to be paired with a stronger programmer. 35% would prefer an equally skilled partner.

6 students (10%) considered the experience to have slowed down the development of their programming skills. 4 of these were in pairs where they considered themselves the superior programmer. 40% considered the experience to have improved the development of their programming skills. Of those, 11 (48%) had considered their partner to be equally skilled and 10 (43%) had considered their partner to be stronger.

4 students did not enjoy the experience and were in pairs in which their partner was perceived to be a stronger programmer.

Unstructured Responses

- Programmers of equal ability should be paired.
- Personalities within pair are important.
- Commitment of both partners.
- Communications is important.
- Another opinion is a good idea.
- Another opinion is a bad idea.
- Partner can spot errors at edit.

3.2 Testing

On development projects students typically demote testing as a deadline approaches. XP promotes testing. The percentage of test case failure on the first exercise ranged from 0% to 60%. 95% rated the time spent developing test cases as productive.

32 students (56%) felt that test cases were more complete than they would be in non-XP exercises. Only 12 (21%) considered them less complete. 3 of those stated that they felt test cases would be more complete if they had the opportunity to implement the code first. 51% stated

that they had spent more time developing these test cases than they usually would. 54% felt that they had been more thorough with 31% expressing that they had been at least as thorough. Only 9% felt that they had been less thorough.

Unstructured Responses

- End up with more test cases earlier.
- End up with better test cases.
- Testing was much more thorough, it was as if both pairs were competing against each other.
- Identify more test cases if coded first.

3.3 Coding Standard

Standards had been agreed on in advance. The replies indicated a confidence that the agreed standards were followed. All students responded that they had applied their standards. As a result of agreeing the standard 60% felt that the code was easier to understand with 35% expressing that it made no difference.

Unstructured Responses

- Useful for team projects
- Easy to agree but difficult to apply
- Takes too long – get on and write the code
- Unnecessary in small projects
- Makes programs a lot easier to understand
- Necessary in any group – should be enforced more rigorously.
- Standards may be overkill.

3.4 Simple Design and Refactoring

Most of the initial solutions were simple and did solve the early problems. As new requirements were added and existing requirements changed the solution grew and became more difficult to modify. A more simple design was required and the design was modified.

- Difficult to decide when to refactor.
- Easier to refactor if customer was actually there providing feedback.
- Not easily done but produces a better program.
- Too much analysis at the start would bring us back to the diagrams. Better to refactor.

3.5 Collective Ownership

In the exercises conducted each team had 4 members. This allowed any pair of programmers to improve the code if the opportunity arose. Responses showed that 37 students (65%) felt that the code belonged to both members of the pair, with 16% stating that it belonged to themselves and 19% feeling that it belonged to their partner.

4 CONCLUSION

This was a first attempt at teaching and learning XP. Student attitude to the process and the exercises completed during practical classes was mostly positive. The pair learning experience seemed to work in that most participants were motivated and involved in the exercises. When a programming pair were at a PC there was little of the usual distraction such as net surfing. Communication was increased. In addition to programming skills other factors such as personality, commitment and communication skills need to be taken into consideration when forming pairs. Through test-first programming the importance of Testing was promoted. An awareness of the importance of standards, design and refactoring was created. Further work with XP is planned.

REFERENCES

[1] Pfleeger, S.L., Software Engineering; Theory and Practice. Prentice Hall (2001).

[2] Williams L. A., "The Collaborative Software Process PhD Dissertation", Department of Computer Science, Salt Lake City, UT, University of Utah, 2000

[3] Yongqing Ye, Wolff W., "A Great Challenge: XP in a typical dot.com.", 2nd International Conference on Extreme Programming and Flexible Processes in Software Engineering, 2001