Comparing Extreme Programming to Traditional Development for Student Projects: A Case Study

John Noll and Darren C. Atkinson

Department of Computer Engineering
Santa Clara University
Santa Clara, CA 95053-0566 USA
{jnoll,atkinson}@cse.scu.edu

Abstract. We conducted an experiment attempting to compare XP with a traditional approach to student software engineering projects. We tasked two groups of student teams with producing a simple room reservation system, one group following a traditional development process, the other using XP. We observed little actual difference between the products delivered by either process. However, in our opinion this is due to certain deficiencies in the way XP was realized in this instance, rather than fundamental flaws in the process itself.

1 Introduction

Our experience with Software Engineering project courses over the past three years showed that student project teams following traditional development processes do not consistently produce good products or documentation. Extreme Programming, with its iterative development cycle, emphasis on delivering value, and lean documentation requirements, seemed like a viable alternative. Because Extreme Programming is driven by delivering value to the customer, we hypothesized that this would help students focus on the product, rather than the individual deliverables, and therefore result in better products.

In the fall quarter of 2002, we conducted an experiment to try to confirm this hypothesis.

2 Method

The undergraduate Software Engineering course at Santa Clara University is a traditional ten week survey course, involving twice weekly lectures, a weekly lab session, and a term-long development project. We divided the class into four teams of six to eight developers; students were assigned randomly to teams. Each team was tasked with developing a web-based room reservation system that could be used to schedule classrooms in several buildings for courses and seminars. All teams were aware that they were part of an experiment to compare traditional methods with XP.

Two teams followed a "traditional" development process, based on Boehm's Anchoring Milestones [1]. These teams had three deliverables: a design document, due at

the end of the third week; an initial operational product, due at the fifth week; and a final product, due the tenth week.

The other two teams followed Extreme Programming, with two releases comprising four one-week iterations. Production code was written using pair programming during the weekly two and one-half hour lab session; we required unit tests to be written before the lab session. The Teaching Assistant served as the "customer" for all four teams, while one of us (Noll) served as the XP coach. We conducted a brief (half hour) overview of XP for the XP teams, covering iterations and releases, test-first design, and the relationship between the customer and developers. We then proceeded directly into the first Planning Game.

3 Results

Neither of the traditional teams could install their first release on any of the target platforms; one team could not even demonstrate a working product on their development platform, and could not offer an explanation why. Both XP teams delivered first release products with minor bugs in their installation scripts; once these were corrected, both products functioned but exhibited significant failures.

All four teams managed to deliver an installable product for the second release. The XP teams delivered relatively robust products with minimal features (completing three of eight stories) during acceptance testing, although the products suffered from sloppy and difficult to use user interfaces. The traditional teams delivered more features (equivalent to all eight stories), but much less robust code.

Overall, none of the delivered products, from either XP or traditional teams, could be considered suitable for the intended application.

4 Observations

In their study of an industrial XP team, Farell, Narang, Kapitan, and Webber observed that a split between customer and developers occurred, resulting in an attitude among developers that they had to "protect their interests" from customers; they also observed that not having detailed acceptance tests early in the development process resulted in developers making assumptions about features, and a "trial and error" cycle between customer and developers trying to achieve closure on stories [2].

Both of these observations are entirely consistent with our experience. When faced with missing or ambiguous requirements, our students seemed prone to making assumptions that suit their desires, rather than seeking clarification from the customer. This was despite the fact that the customer was always available during lab sessions. According to one student, "No one wants to make changes, and we challenge you [the customer] when you request changes because we need to make sure they are worth to you the effort they will require from us."

Students also had difficulty with the concept of collective ownership: they were very reluctant to fix problems in code written by others. This seemed to be a social rather than technical problem: most were familiar enough with the entire code base to have

the expertise to fix a problem, but seemed to feel that this would violate some principle of responsibility ("you broke it, you fix it").

The XP teams were reluctant to apply continuous integration. We frequently observed pairs "hoarding" finished code rather than integrating and testing it. This often resulted in a mass integration at the end of the lab period, producing confusion when inevitable problems with the build arose.

This may have been due to a lack of confidence that the features completed were actually finished; although they passed the unit tests, developers still seemed to think problems remained. They did not seem to make the connection between integration testing and discovering these problems.

5 Conclusions

Despite the inconclusive results of our experiment, we still believe that XP has potential as a pedagogical software process for software engineering project courses. We feel that the failure of our XP teams to produce significantly better products was due to two deficiencies in our application of the process: an insufficiently strong customer, and inadequate introduction to XP's values and practices. Consequently, we would adopt the following changes in future applications of XP in the classroom:

1. Provide a strong customer by having the professor play this role. This will ensure there is no misunderstanding of who is responsible for defining requirements, and that the customer has sufficient authority to insert himself into the development process when necessary.
2. Devote more class time to discussing the differences between XP and traditional software processes, in an attempt to increase student understanding of the reasons for various XP (and traditional) practices. Without adequate explanation of the reasons for XP's practices, students seemed inclined to dismiss XP as more difficult, restrictive, and therefore lacking merit. As Lappo [3] observed, it is difficult to appreciate the benefits of XP without first experiencing the pitfalls of other development processes.

References

1. Boehm, B.W.: Anchoring the software process. IEEE Software **13** (1996) 73–82
2. Farell, C., Narang, R., Kapitan, S., Webber, H.: Towards an effective onsite customer practice. In Succi, G., Marchesi, M., eds.: Proceedings of the Third International Conference on Extreme Programming and Agile Processes in Software Engineering, Alghero, Sardinia, Italy (2002) 52–55
3. Lappo, P.: No pain, no XP – Observations on teaching and mentoring extreme programming to university students. In Succi, G., Marchesi, M., eds.: Proceedings of the Third International Conference on Extreme Programming and Agile Processes in Software Engineering, Alghero, Sardinia, Italy (2002) 35–38