A Study of the Acceptance of XP Practices

Vinay Ramachandran, Anuja Shukla, Laurie Williams

Department of Computer Science,
North Carolina State University,
Raliegh, NC - 27695
{vramach,ashukla,lawill3}@unity.ncsu.edu

Abstract. Many of the twelve carefully defined practices of Extreme Programming are tightly coupled. The various practices have checks and balances on each other and are, in many ways, dependant on each other. Therefore, neglecting essential practices can have sub-optimal or negative consequences to team results. The practices that are actually utilized in team development can often be attributed to the perceptions of the value and difficulty of the practices by the developers on the team. Through two surveys answered by 27 developers, we have assessed the utilization of and the sentiments towards the XP practices. In general, these developers we surveyed valued and utilized most of the practices.

1 Motivation

The twelve XP practices support each other. The weaknesses of one are covered by the strengths of others[1]. In general, the dependencies between the practices imply risk when the team does not utilize all the practices. However in [2], Ron Jeffries shares his wisdom and experienced judgment in assessing which of the twelve practices are essential for teams to be successful through utilizing an XP or XP-like methodology.

At times, XP teams explicitly or implicitly create a customized variant of the methodology for one of two reasons. First, the team consciously decides they do not want to utilize some of practices. Alternately, the team or certain team members might not actually do these practices, whether they are technically supposed to or not. Second, experienced XP teams can learn to reduce the coupling between practices by utilizing other practices to customize their own variant XP methodology. Jeffries states, “As teams become experienced with XP, they develop the wisdom to move beyond the basics, modifying practices, adding them, replacing them”[2].

To assess the utilization of and the sentiments towards the XP practices, we ran two surveys of practicing Extreme Programmers. Twenty-seven developers answered the survey. We share the results of this survey in this paper.

2 Keywords

XP, extreme programming, agile process, XP practices

3 About the Surveys

Two surveys were conducted in order to gather information regarding XP projects. The survey questionnaire was based on the survey conducted by Robert Gittins[3] at School of Informatics University of Wales Bangor. The questions in each of the two surveys were very similar, but not identical. The first survey, which is referred to as Survey-I throughout this paper, was administered to ten developers at a single project group of a telecommunications software development organization. The second, referred to
as Survey-II, was administered to individuals belonging to four different organizations. Since the two questionnaires were not identical, some information obtained by Survey-I could not be obtained from Survey-II and vice versa. When neither survey is specified, the question was asked of both surveys. For more details about the survey, please refer to the appendix.

4 Views on XP Practices

4.1 The Planning Game

The planning games are a very important aspect of XP. Ron Jeffries mentions “The planning game is part of the essential cycle of the XP team’s process: it defines what will be done, it provides key feedback between programmers and customers.” [2] The survey respondents in general felt that the planning games had worked well for them. Planning games necessitates a great deal of interaction between the programmers and customers so that they are in sync. Since the interaction between the customer and the programmer is so important, it would be interesting to know if they comment freely. In Survey-I, 90% of the participants agreed that they commented freely whenever they felt they could contribute in the planning games.

XP uses story cards to obtain customer requirements. Ninety-two percent of them believed that story cards were effective in capturing customer requirements. Once the stories are written, estimates have to be made by the development team. XP teams are supposed to empower the developers to create estimates themselves, rather than to have developers inherit estimates developed by managers or coaches. Eighty-nine percent felt that the story estimates and 56% felt that the task estimates were made collectively by the team. We did not have data to explain why only 56% of the participants agreed that task estimates were made collectively. A probable reason for this could be that the developers responsible for implementing the tasks made the task estimates. On the whole, 92% of them felt that estimates made about ‘story’ and ‘task’ time allocations were effective.

The planning game includes tracking the velocity of the project; an assessment of how much observable progress has been made by the team. Monitoring velocity can give early warning of problems with testing or refactoring [2]. The majority (63%) agreed that tracking project velocity is important. But, a significant 41% of the participants stated that they had no experience in using project velocity. We did not have data in order to explain why a significant percentage of participants did not have experience in using project velocity. A probable reason could be that the higher management and not the developers did the setting and tracking of project velocity.

4.2 Pair Programming

Pair programming is a style of programming in which two programmers work side-by-side at one computer, continuously collaborating on the same design, algorithm, code or test [4]. Research has shown that pair programmers produce higher quality code in about half the time as solo programmers[5]. However, there is still considerable resistance to transitioning from solo to pair programming. From the manager perspective, it appears that two persons are doing the work of one. Programmers initially resist the transition, likely because they are long conditioned to working alone[6]. They are skeptical of the value of collaboration in programming, not expecting to benefit from or to enjoy the experience [4]. Our data indicates that programmers ultimately overcome this initial resistance. The programmers who took Survey-I initially classified pair programming as a most concerning XP practice before they made the transition. Ultimately, 93% felt that enjoy work more than when programming alone. Ninety three percent also felt that they were more confident in their solution than while working alone. Also, 67% of the participants felt that pair programming was more efficient than independent programming and only 19% felt that pair programming is no more efficient than an experience programmer working alone.
Another notable aspect was that almost none of the participants preferred to work independently. This alludes to preference of pair programming over independent programming. However, more than 52% agreed that pair programmers should work alone some of the time.

The participants felt that following factors were critical for pair programming:
- 96% of them felt that the partner must be committed to pair programming.
- 96% of them concurred that both partners must share the screen and the keyboard.
- All of them felt that management support is critical for pair programming.

### 4.3 Small Releases

XP specifies that software be built in two- to four-week small releases. Small releases allow for the fastest possible return on investment and giving customers greater vision into the project [1]. Among the survey participants, this was one of the most highly used practices of XP. In Survey-I, the majority of the participants (88%), felt that small release cycles were helpful. On a scale of 0 to 10 (where 0 indicates the minimum and 10 indicates the maximum), the Survey-I respondents rated its overall usage as almost a perfect 10 (on average).

### 4.4 Continuous Integration

Ron Jeffries states, “To really accomplish small releases, you need the ability to build the system reliably and frequently. To support collective ownership, you need to avoid code conflicts. The more frequently you integrate, the fewer conflicts you’ll have.”[2]. It seemed that continuous integration was a frequently used practice. In Survey-I, the participants rated its usage as 9 (on average) on a scale of 0 to 10 and the majority (75%) felt that continuous integration was helpful for them.

### 4.5 Simple Design

The Extreme programming design philosophy emphasizes that the features should be built using the simplest practical design. This practice, combined with Design Improvement (Refactoring), lets XP teams deliver business value from the beginning, rapidly, and safely [2]. Simple Design was a highly popular practice of XP. Seventy eight percent of respondents believed that simple design is the key to getting the job done quickly. Getting the job done quickly is not the only priority; it must also be done right. However 33% of them also believed that simple design is not necessarily the right answer. Perhaps these survey participants felt that complex design was inevitable under some circumstances.

Often programmers try to speculate future requirements and design accordingly. Kent Beck states, “If you believe that future is uncertain, and you believe that you can cheaply change your mind, then putting in functionality on speculation is crazy.”[1]. The survey participants felt the same too. Only 7% felt that putting functionality based on speculation can be justified and only 15% felt that adding functionality early in the project saves time later. Also, 44% of respondents believed that adding functionality early in the project slows the project down.

### 4.6 Testing

In XP, tests are divided into acceptance (or customer) tests and unit (or programmer) tests. Programmers continually write unit tests before writing code, which must run flawlessly for development to continue. Customers work with the programming team to write tests demonstrating that user stories are finished [1].
Jeffries states, “Running without acceptance tests is running out of control” [2]. Fifty eight percent of respondents considered functional testing in general to be adequately covered, meaning that the functional test cases written by the customer properly exercised the program. But, it seems that acceptance test cases are seldom written early. In Survey-I, only 30% reported that they were often written at story creation time and in Survey-II, 88% said that functional test cases were only available in the middle or later part of the story development.

Unit testing seemed to be performed thoroughly in the project teams that were surveyed. Ninety two percent perceived that all classes in the system were tested during unit testing; and 92% of them considered that unit testing in general was properly exercising the code. Unlike acceptance testing, a high majority (96%) felt that these automated unit test cases were often created written early.

4.7 Refactoring

XP does not advocate a “Big Design Up Front” (BDUF) [7]. The design of the code evolves as it is written utilizing the test-first unit test practice. The resulting implemented design may not be desirable. Therefore, the Simple Design practice is dependant up on the refactoring practice. Refactoring is the process of improving the design of existing code [8]. Refactoring is an essential aspect of XP. If a project is to deliver business value from the very beginning, the design must be kept clean. This requires continuous design improvement implemented by refactoring [2]. But there is a concern as to how much refactoring is actually done. Developers may get overly focused on delivering functionality and neglect the important refactoring. Sixty two percent considered refactoring to be adequately performed. However refactoring seems to be sporadic. Fifty eight percent said that they performed refactoring when they had the time. Also, Survey-I revealed that, 50% of the participants felt that the refactoring aspect of XP could be further improved in their project.

The refactoring practice is in turn highly dependant upon the set of automated unit test cases. When code is refactored, it must still pass all the unit test cases in the code base. If test cases fail after refactoring, either the code or the tests themselves must be fixed. All (100%) of the Survey-II respondents reinforced that automated test cases were often run during refactoring; 94% of them agreed that duplicate code was removed during refactoring.

4.8 Collective Ownership

On an Extreme Programming project, any programming pair who sees an opportunity to add value to any portion of the code can do so any time. Everyone shares the responsibility for the quality of all the code Ron Jeffries also contends that this collective code ownership improves quality [2]. Seventy percent of the participants agreed with Jeffries statement. Moreover, 85% of the participants believed that collective ownership encourages the entire team to work more closely.

But it was interesting to find out to know whether the participants were comfortable with someone changing their code. In both the surveys, none of them objected others changing their code.

4.9 40-Hr Week

This practice reflects the need to keep the team healthy. The practice means to work hard, and to rest when you need to [2]. This is also a popular practice of XP and is used frequently. In Survey-I, the participants rated its usage as 9 (on average) on a scale of 0 to 10 and the majority (86%) felt that the 40 hr week practice was helpful to them. In Survey-II, the majority of 65% stated that they were comfortable with the pace of their team.
4.10 On-site Customer

XP requires the presence of an on-site customer to clarify requirements. Kent Beck states, “A real customer must sit with the team, available to answer questions, resolve disputes, and set small scale priorities.” [1] Consequently, 81% of the participants in Survey-II agreed that onsite customer was working positively for them. Also, in Survey-II, 67% stated that they had access to the customer all the time; the rest had access to the customer sometimes for answering their queries, which served the purpose of having an on-site customer.

In Survey-II, 94% of the participants had their customer located at least in the same floor and the most common means of communication with the customer was via informal meetings. Among the other ways of communication were email and telephone, which were moderately used.

4.11 Coding Standards

XP teams all write code following the same style guidelines. This makes pairing easier and supports collective ownership [2]. This is also another commonly used XP practice. There seems to be many ways by which coding standards are created and followed. In Survey-II, 53% stated that the entire team collectively decided the coding standards that would be adopted for the project and 29% stated that they followed the already existent company/department/team coding standards. In Survey-II, 12% even stated that they chose their own coding conventions. The coding standards document was generally small. In Survey-II, 89% of the participants had coding standards document that was 5 pages or less.

4.12 Metaphor

With the Metaphor practice, the system design/architecture is expressed using a simple, common system of names, a simple easily-explained pattern of design. This enables you to say “the program is like an assembly line” or “the program is like bees bringing pollen back to the hive.” [2]. Metaphor seemed to be the least used XP practice among the people surveyed. Only about 37% of them believed that metaphors are helpful in creating a mental picture of the system. The majority of the participants, 80%, claimed that they knew the concept of a metaphor but had never used it and only 1% of the participants had used metaphors and felt that it was important. In Survey-II, 86% of the participants stated that their projects did not have any metaphor. As one of them put it, “We don't need it to help understanding the system.”

5 Overall Impression of XP

The participants felt that all of the following aspects were aided by XP and these are reflected by the positive averages on a scale of −5 (Much worse) to 5 (Fully achieved)

- Deliver software in time (4)
- Develop software with a high quality i.e. with less bugs (4)
- Let developers have fun with their work (3)
- Allow changes without incurring big costs (3)

Finally, 92% of the participants in Survey-II stated that they achieved 75% or more of their initial objectives when their project completed. This information could not be obtained from Survey-I as the project had not yet terminated.

Overall, almost all of the survey participants seemed happy with XP. Unanimously, respondents stated that they would use XP again and almost all of them expressed that they would advocate using XP in the future. One survey participant wrote,
I still don't know how management will plan releases of a product which is developed using XP. On the other hand, XP gives management a very high visibility of the progress of the work, which is something that they normally don't get in a traditional development paradigm. In the old days, management won't know that a project is late until after integration testing has started (or tried but failed to start), which may be many months from when coding first started.

6 Conclusion

All the practices were popular and well accepted with the exception of the metaphor. Most of them had not used metaphors in their projects and it seemed many projects did not have metaphors. Among the most popular practices was planning games, pair programming and short release cycles. Most of the participants were happy with XP and said that they would advocate XP in future projects.

Acknowledgements

The research outlined in this paper was funded by the Center for Advanced Computing and Communication (CACC), NC State University. The authors gratefully acknowledge the employees of the various organizations who participated in the two surveys. We would also like to thank James Grenning of ObjectMentor for helping us find survey participants.

About the Author

The authors Vinay Ramachandran and Anuja Shukla are computer science graduate students of North Carolina State University. They are currently involved in research pertaining to agile methodologies.

Appendix: Details about the Survey

Survey-I was administered to a group of ten participants working on the same project of a particular organization. Survey-II was administered to a group of seventeen participants spread across four different organizations. The two surveys had multiple-choice questions with set of answers to choose from. Generally, the answers choices provided to the user conformed to Likert scaling in order to aid the collection of the statistics. The answer choices were continuums mostly like “Strongly Disagree”, “Disagree”, “Agree” and “Strongly Agree” or “Very Ineffective”, “Ineffective”, “Effective” and “Very Effective”, which can be scaled linearly from 1 to 4. There were no neutral choices like “Neither Agree nor Disagree”. Hence, the choices were such that they forced the respondent to express a definite opinion, either positive or negative. There were also questions for which Likert scaling was not used since the answer choices were not continuums for e.g. for a question “I perform Refactoring …”, the answer choices were “If I have the time”, “When I am asked to”, “Always”, “Other”. A few optional questions also required the respondent to type in entire sentences for e.g. the participant had to enter some comments about testing.

The analysis of the data involved finding the statistical information such as mean and mode of the answers chosen by the participants for a particular question. In some cases the final values were rounded off to the nearest integer since their decimal values were not very significant. And since the two questionnaires were not identical, some information obtained by Survey-I could not be obtained from Survey-II and vice versa.
A majority 74% of the participants were involved in pair programming for about 3 months. Sixty seven percent, of Survey-I participants had attended planning games for 3 months or more. Equivalently, 76% of participants in Survey II had attended 7 or more iteration planning meetings.

References