Teaching Software Process Improvement through Extreme Programming practices, study case: UAA

Arturo Barajas Saavedra, Francisco J. Álvarez Rodríguez, Jaime Muñoz Muñoz Arteaga

Universidad Autónoma de Aguascalientes, Avenida Universidad 940, Bosques del Prado, Aguascalientes, Ags., 20100. México

ar.ba.saa@gmail.com, {fjalvar, jmunozar}@correo.uaa.mx

Abstract. This paper describes the practical results of an experiment created in order to analyze the software process improvement by the use of the practices and rules of Extreme Programming methodology. In the experiment a group of 40 students in the Universidad Autónoma de Aguascalientes took part to create a small system, to measure their productivity and to measure how much does this rules and practices are applicable in the real world.

1. Introduction

One of the most significant challenges faced by the software engineering community in recent years has been to continually capitalize on software development and maintenance experiences, whether good or bad. Capitalizing involves application of new technologies and evolution of technologies already widely in use, as well as the definition and adoption of standards. There is an evident need to implement some means by which every software activity provides new and improved insight into continually improving methods for developing and maintaining software; every experience must be used to gain new knowledge. [8]

The Software Process Improvement is the continual and iterative improvement of both the software process and products through the use of project experiences. [8]

In the early 1990’s a man named Kent Beck was thinking about better ways to develop software. What Kent came to realize is that there are four dimensions along which one can improve any software process project. You need to improve communication. You need to seek simplicity. You need to get feedback on how well you are doing. And you need to always proceed with courage. The result was the Extreme Programming (XP) methodology. [1]

Extreme Programming (XP) is actually a deliberate and disciplined approach to software development. The methodology is designed to deliver the software your customer needs when it is needed. XP empowers your developers to confidently respond to changing customer requirements, even late in the life cycle. This methodology also emphasizes team work. Managers, customers, and developers are all part of a team dedicated to delivering quality software. XP implements a simple, yet effective way to enable groupware style development. XP improves a software project in four essential ways; communication, simplicity, feedback, and courage. XP programmers communicate with their customers and fellow programmers. They keep their design simple and clean. They get feedback by testing their software starting on day one. They
deliver the system to the customers as early as possible and implement changes as suggested. With this foundation XP programmers are able to courageously respond to changing requirements and technology. XP can be used in different problems domains, but it was created for domains where: requirements change constantly, maybe every few months, the customer does not have a clear idea about what the system should do, and risk is high due to specific deadlines, new technologies for the development team or, even worse, new technologies in the industry of software development. [1]

XP requires a specific team. The manager, developers, and the customer must be involved in the development of the software process, working together to improve the quality of the product.

XP recognizes that the end goal of a development project is to produce quality, production code that can be executed and maintained. Anything in a project that does not directly support this goal is questioned and discarded if appropriate. [2]

XP is a collection of the “best practices” in software development called rules and practices each of which supports others, and are supported by several others in turn. [1], [2]. The XP practices are classified in four different areas of a software development process: Planning, Designing, Coding, and Testing. The Extreme Programming practices are based on the experience of Kent Beck and Ward Cunningham from a number of projects in which they worked together. Over the years, they observed what practices worked well in software projects and what didn't, and fine-tuned their understanding to four key values and 12 key practices of XP. [1], [7]

![Extreme Programming Project process flow chart](image)

**Figure 1. Extreme Programming Project process flow chart [1]**

2. State of the Art

There are studies that reflect the process improvement of applying the Extreme Programming methodology in several types of projects ([3], [4]).

The first study is one published by the Agile Alliance [3] that compares the performance of a group of 21 students divided into three categories: six programmers in PSP, five pairs of programmers in XP (called by the authors XP2) and 5 programmers in XP but not in pairs, they were alone (called XP1).

In the paper they have compared XP-like pair programming (XP2) with individual programming based on PSP (PSP) and XP practices (XP1).

XP2 appears less efficient than it is reported by J.T. Nosek [5] and L. Williams et al. [6]. Only in one case reduction of average development time for XP2 (to 77% of the time needed by XP1) was similar to that reported by J. T. Nosek (reduction to 71%)
but still it is far from the level of 50% mentioned by L. Williams et al. Moreover, in a case, one of the students misunderstood the assignment and this distorted the results.

PSP seems less efficient than XP1. However, its average code size was considerably smaller than in case of XP1.

Pair programming is more predictable than individual one with regard to development time and program size.

Experimentation and test-oriented thinking reduces development time.

The number of re-submissions indicates that the amount of rework for XP2 is slightly smaller than for XP1 or PSP.

In [4], Michele Sliger, with fifteen years experience in software development, with seven of those years in project management, emphasizes in the advantages of using an Extreme Project Management methodology. She gives us the Four Noble Truths of Agile Development:

Software development can be fun. Fun is the result of team members working in an environment that supports their need for constant real-time communication, which serves to keep everyone focused.

The attainment of these objectives results in software development nirvana: the delivery of a quality, customer-approved product in a shorter time frame.

The path to nirvana is realized by taking the road of right thinking (being realistic and reasonable) and common sense, the basic truths of all agile methodology.

This doesn’t mean that she thinks agile project management is easier than traditional project management, a common misconception. On the contrary, she has found that Extreme Project Management is much more demanding because of the number and level of skills that are required.

3. Problematic

When using a heavyweight methodology; like CMM, CMMi, MoProSoft or SPICE, everything becomes administrative work that was designed to be a historical record rather than a tool to assist in problem solving and project management. This is a huge problem because we stop having personal interactions that does more for the focus of the team than the endless stream of email documents [4].

The adoption of heavyweight methodologies implies the use of extensive and intensive documentation to manage the process with several forms to be filled out by the engineers, programmers, and managers, which means that they have to spend time in these activities instead of spending it on work for the solution of a problem.

Through the use of a lightweight methodology for developing a software product, we can focus on the main and most important activities like planning, designing, coding, and testing according to the pending activities and the current status of the software process, instead of doing it weeks or months before executing the activities.

In order to improve software processes is necessary to reduce their time and costs, and to increase the quality and the scope of their products and of the processes
themselves. According to this, the improvement of a software process involves to make the most of the available time by reducing the time spent in long meetings, in the filled out of several forms, and many others activities that only creates a historical record of the activities executed. To improve software processes means to create essential and very significant documentation that contains the most important aspects of the software processes, to spend time in coding and testing all the software products of the processes, and the most important thing, to communicate effectively and efficiently with the customer and the stakeholders, other programmers and engineers, and the managers of the software project.

4. Methodology

In this section we will explain the activity executed in order to get information and data about the performance of 20 pairs of programmers working according to the rules and practices of the Extreme Programming methodology.

4.1. Programmers selection

The programmers selected for this activity are students in the Universidad Autónoma de Aguascalientes, in the eight semester of Computer Systems Engineering. In this experiment, both of two groups were selected to do this activity.

The two groups were divided into pairs, having 20 of this pairs of programmers; ten per group. The programmers were randomly assigned one to each other to make a pair.

The experiment was scheduled for the first group in April, 11th 2006, and for the second group in May, 17th 2006, with duration of 120 minutes for each group.

4.2. Programming language selection

In this case, the programming language was selected by the pairs according to their skills. The languages selected were: Java, C++, C Builder, Visual Basic 6 and VB .NET.

4.2.1. Used methodology

The used methodology for performing the experiment is described in the next paragraphs.

The experiment was divided into two parts, the first one is the delivery of the user stories, and the second part is the simulation of a change in the requirements.

The first delivery of the user stories were provided to the students and it described the necessity of a computing system capable of calculate different measures (perimeter, area and volume) of six different geometric figures. In the second iteration there was a change in two of the six figures, which means a change in the 33.33% of the original requirements, and these second user stories were delivered almost at the end of the experiment in order to simulate a change in the requirements late in the life cycle.

The products that the pairs must provide or deliver to us were: CRC cards, Iteration planning, Release planning, Source code and Executable code.
The initial time estimation for this experiment was 120 minutes. This estimation included the four areas: Planning, Designing, Coding and Testing, executed in two iterations.

![Diagram of software development process]

**Figure 2.** This figure is an example of a figure caption taking more than one line and justified considering margins mentioned in Section 5.

### 4.3. Experiment objectives

The first objective is to prove that the communication in the work team is increased and it is much more effective. That it isn't about just writing down all the done activities, pending activities and times in many documents that no body will read, it is about talking with your coworkers and discuss the schedule and propose the best solutions for the software process just in that point. [4]

The second objective is to prove that the productivity is increased due to the pair programming, as we can see from the research in [3]. Also to prove that pair programming increases the productivity, not in the way that the pair codes much more lines, but in the way that the few lines coded is the optimal solution for the function, module, or unit test they are working in.

The third objective is to prove that the time is decreased because while one programmer is programming, the other one is checking the program and avoiding deviations from the requirements and objectives.

The forth objective is to prove that the number of LOC, as well size, effort and time is much more predictable with this methodology.

### 5. Experimental Results

As we can see in the figure 3, the mean time spent in the Coding area of the software system is significantly longer than in the other three areas in the first iteration. In the second iteration is almost the same mean time for the four areas, this is because the programmers just had to create two new modules instead of six. Also, in the Testing area a longer time was spent, due to the practices and rules that focus on testing all the modules and codes in order to be released with no bugs.
According to the structure of the rules and practices of XP, this is the expected behavior of the software process, because this methodology emphasizes in the Coding and Testing areas, leaving in a second plane the rigorous documentation and planning.

The figure 4 shows how the time is distributed in the four areas of the software development process. For the first iteration the mean total time was of 105 minutes, and for the second iteration the mean total time was of 53 minutes.

The next figure shows the mean time spent in the construction of each module. As we can see this is a very predictable time due to its small standard deviation.

Some statistical measures for the LOC produced by the pairs are:
6. Conclusions

In accordance with results derived from our experience based on the study of 20 pairs of programmers working with the rules and practices of the Extreme Programming methodology and from the pupils’ experience of working with a lightweight methodology like XP, we proposed the use of this methodology in small and medium size software processes.

All the experiment objectives were achieved, so we based our proposal in them, because this achieved objectives proves that the Extreme Programming methodology improves the software process as we mention in the section 3. Among the achieved improves we can mention that the communication was more effective, the productivity was increased significantly, the time was decreased, and the number of LOC, the effort and the time were much more predictable.

Future works and researches are related to use of XP methodology in large size software processes.

References


Sambasivam, G. “Extreme Programming (XP)”. In Agile Alliance, www.agilealliance.org/articles

Nawrocki, J. and Wojciechowski, A. “Experimental Evaluation of Pair Programming”. In Agile Alliance, www.agilealliance.org/articles


Sharma, P., CEO, Pragati Software “An Introduction to Extreme Programming”, Agile Alliance, www.agilealliance.org/articles


Table 1. Minimum, maximum, mean and standard deviation for LOC and time

<table>
<thead>
<tr>
<th></th>
<th>LOC</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimum</td>
<td>63</td>
<td>1.00</td>
</tr>
<tr>
<td>Mean</td>
<td>160</td>
<td>6.32</td>
</tr>
<tr>
<td>Maximum</td>
<td>752</td>
<td>18.00</td>
</tr>
<tr>
<td>Std Deviation</td>
<td>180</td>
<td>3.13</td>
</tr>
</tbody>
</table>