Quality Improvement in XP Development

Avram Eskenazi
Plamen Balkanski

Abstract: *Extreme programming (XP) has been introduced a few years ago as a suitable programming method for high-speed and quality software development. XP and earlier attempts to discover a relation between XP and software process improvement methodologies are reviewed in an attempt to initiate a methodology for XP process improvement. The conclusion is that XP supports many good practices but there is still place for refinements. An XP process improvement methodology is to be defined and used to achieve better quality of code and design. Another possible strategy we propose is to add dedicated quality assurance measures, which seems to be sufficiently effective but at the same time is much more simple .*

Key Words
Extreme Programming, XP, Software Process Improvement, Capability Maturity Model, CMM, lightweight methodologies, quality assurance.

1. INTRODUCTION

Extreme Programming (XP) has been introduced a few years ago as a suitable programming method for high-speed and quality software development, Kent Beck, Ron Jeffries and Ward Cunningham being considered as its authors. It was described as a "lightweight" methodology that is appropriate for small to medium sized teams and environments accompanied by rapidly changing requirements. Although XP has been presented as a disciplined process, defined by twelve basic practices, some have use it in opposition to software process improvement models as the Capability Maturity Model (CMM), BOOTSTRAP or Personal Software Process (PSP). Such an approach is controversial since XP addresses many accepted software engineering practices that can be considered for adoption in every software development company where appropriate.

This paper presents a short introduction to XP, an overview of the previous attempts to reveal some relation between XP and software process improvement methods and critiques both in an attempt to initiate a methodology for XP process improvement or to find out a more restricted, but sufficient way to assure quality. XP is a new methodology and was proven to be successful when used by small to medium sized teams to develop small to medium sized applications. Most of the practices are defined as a result of their successful performance in real projects. The conclusion is that XP could get improved to achieve better quality in design, coding standards and reusability. This can be achieved by applying a software process improvement model designed for XP. However, our main idea is that more simple measures would also lead to better quality of the end product.

2. INTRODUCING EXTREME PROGRAMMING

XP is a discipline of software development based on simplicity, communications, feedback, and courage. [1] XP requires the whole team to work together observing a set of simple practices and receiving enough feedback from the Customer to enable the team to tune the process and apply the practices to meet the unique circumstances.

Although some have used XP as opposition to accurate models for software process improvement, it is a disciplined process. Kent Beck says that none of the ideas in XP are new; all the techniques have been already proven and what is new is the emphasis on

practicing all these ideas continuously [2]. The principles advocated by XP can be summarized by the following twelve basic practices.

Team Members

The XP team must include a member to represent the client (Customer), who provides requirements, sets priorities and controls the project. The team may include testers, who help the Customer to define the acceptance tests and analysts who help the customer to define the requirements. There are programmers, a coach to relieve the process and possibly a manager to synchronize activities, supply resources and handle communications. The roles are not strictly defined, instead everyone on an XP team contributes in a way that they can.

Planning game

XP planning aims to predict what will be accomplished by the due date and what must be done next. Release planning is where the customer presents the desired future and timing of releases based on estimates provided by programmers. Programmers implement only the functionality demanded by the customer stories in this iteration. Iteration planning is where the customer presents desired future for the next release and based on the amount of work accomplished in the previous iteration, the team decides what will be undertaken in the current iteration.

Acceptance Tests

Both programmers and Customer write functional tests for the stories in iteration. The team builds automated acceptance tests to prove that a feature is implemented correctly. Once the test runs, the team must keep it running correctly thereafter.

Small releases

At the end of each iteration the team releases running and tested software, providing business value ordered by the Customer. Release to the end users, if any, is also highly recommended. New releases are made often anywhere from daily to monthly.

Simple design

At any moment the program must be capable to pass all the tests for the current functionality, must not contain duplicate code, and must have the smallest possible amount of classes and methods. The XP team keeps the design exactly suitable for the current functionality of the system and the development goes on through testing and design improvements all the time.

Pair programming

All the code in XP team is written by two people working together at the same machine. Thus the entire source code is reviewed all the time and the result is better design, testing and code. Pair programming also serves to communicate knowledge throughout the team.

Test-First Development

Since XP relies on quick feedback it requires quick results from testing. Programmers write acceptance tests before coding the feature that these tests address. Top XP teams practice "test-first development" working in very short cycle of adding a test and making it work. [3]

Refactoring (Design Improvement)

XP is trying to deliver business value in all iterations. Therefore the design of the program is developed through continuous improvements of the existing design. Refactoring is removing duplication, increasing integration and lowering mixture of the code.

Continuous integration

XP teams keep the system fully integrated all the time. New code is integrated with the current system after no more than a few hours. When integrating all tests must pass or the changes are discarded. This practice evades infrequent integration that usually leads to serious problems and delays.

Collective ownership

In XP teams every programmer can improve any code anywhere in the system and at any time if they have the opportunity. In order to work this practice all the programmers must follow a common coding standard. Thus all code gets the benefit of many programmers, which increases code quality and decreases defects.

Metaphor

XP teams must develop a common vision of the functionality of the program. This outline of the system is defined by a metaphor – a simple suggestive description of how the program works. The team must use a common system of names to ensure that everyone understands how the system works.

40-hour weeks

XP teams may work overtime when it is effective but usually no one can work repeated weeks of overtime. The normal work must be in a way to maximize the team productivity.

Ron Jeffries says that if we are doing less than XP suggests our project is in danger of going two slowly or producing too low quality; if we are doing more than what XP suggests, there is fair chance that we are not as efficient as we could be. [4]

3. PRIOR ATTEMPTS TO IDENTIFY A REFERENCE BETWEEN XP AND EXISTING METHODS FOR SOFTWARE PROCESS IMPROVEMENT

As a defined process XP follows defined rules that have to result in a rapid and quality software development. This is not the first time someone tries to discover a methodology to improve XP processes. Before doing any suggestions lets have a look at the previous attempts.

3.1. Ron Jeffries, Extreme Programming and the Capability Maturity Model [5]

This paper appears to be the first topic trying to find a relation between XP and the existing SEI CMM for process improvement.

In this article Ron Jeffries advocates the statement that XP has some characteristics in common with the higher SEI levels, up to and including level 5 and is in some ways a vertical slice through the CMM. [5]

The author is passing through all CMM levels and quotes model Key Process Areas (KPA) that he considers as proposed by XP practices too. Then he tries to interpret XP rules through the perception of CMM and explain how an XP team manages to achieve the specific requirements. However there is no assertion about what place can take an XP team into the levels of CMM or which KPA can be used to improve an XP development process.

Following his goal in this paper, to answer the question: "Is XP a SEI level 1 process?", Ron Jeffries does not try to identify any specific and well defined relation between XP and CMM. Instead he only tries to prove that XP addresses many of the CMM principles of any of the 2-5 levels.

3.2. Mark C. Paulk, Extreme Programming from a CMM Perspective [7]

As is well known, Mark Paulk is one of the creators of the SEI CMM. His article summarizes both XP and the CMM and critiques XP from a CMM viewpoint.

In the paper Mark Paulk broadly presents CMM by identifying the goals of the 18 KPA that formally describe the model. He also describes XP as a lightweight methodology summarized by twelve practices. Since most of these practices are commonsense and appropriate in any disciplined process he is trying to show what are the extreme ones.

The author objects to using XP for process improvement as it hardly touches management and organizational issues. He also states that CMM and XP can be considered complementary as CMM defined what to do and XP is a set of basic "how-to" practices. In spite of that Mark Paulk thinks that XP practices may be compatible with the intent of a KPA even if they do not completely address it. Then he goes through the CMM levels 2 to 5 trying to find relations between XP practices and CMM KPA.

In conclusion Mark Paulk recommends most of XP practices to be considered thoughtfully for any environment. He names XP a methodology that is effective within its context of small, co-located teams but it is questionable whether XP should be used for high reliability systems.

The author tries to find out common practices between XP and CMM but he considers the two entities complementary. He objects to using XP for process improvement and does not try to find a way to apply such methods on XP. This paper is not a study on how to improve XP processes but examines the relations of the basic XP practices with a model for software process improvement.

3.3. Jerzy Nawrocki, Bartosz Walter and Adam Wojciechowski, Toward Maturity Model for Extreme Programming [6]

The authors of this paper are proposing a simple four-level maturity model for XP similar to the CMM. They consider that we need such a model to be able to distinguish different levels of advancements in XP practices.

Rules and practices of XP are described as in [8]. They are split into four areas: Planning, Designing, Coding, Testing and Facilities which is a new area added by the authors as well as some new practices in Planning, Coding and Testing. This scheme is created as in [8] by splitting the twelve practices defined in [1] into small rules grouped by development activity.

In the proposed maturity model for XP (XPMM) the defined rules are assigned to the levels 2-4. KPA are taken from the CMM and above rules are used to discern whether the KPA is used in a project.

The article presents an XP maturity model, which should help to distinguish "real" XP Projects and pseudo-XP projects [6]. The proposed model assumes that we may do XP at different levels. For example On-site customer and Acceptance tests are present at the highest level 4 called mature. As these are basic XP practices according to [1], [2] and [3] the proposed XPMM model sounds contradictive. It also does not conform to the words of Ron Jeffries that if we do less or more than XP suggests we probably are not efficient enough. And finally following its goal the model does not care about what if we are "doing more than XP suggests" [4].

The model is not what we have expected from a maturity model but despite some inaccuracies, XPMM seems to be the first attempt to apply a maturity model for XP.

4. POSSIBILITIES FOR QUALITY IMPROVEMENT IN XP DEVELOPMENT

In no doubt XP supports many good practices that are an important part of almost every software engineering method. It can be easily found even a relation between XP and software process improvement models as CMM. But XP is not a methodology for process improvement. Primary it is a new methodology that still can be considered as "in development". XP defines a set of rules that if followed can lead you to developing quality products quickly by skipping most of the design and documentation, working more efficiently by using pair programming and having a client as a member of the team.

Many papers have recently appeared, describing successful implementation of XP in a variety of projects. XP efficiency is proven when the method is applied by small to medium sized teams to develop small to medium sized applications. However it is doubtful if XP, as published, should be used for life critical and high reliability systems. The lack of design documentation and the de-emphasis on architecture can be considered as risky decisions. Rejecting code reusability causes the team to reproduce already written code in many projects. Some of the practices may be objectively hard to follow in 100 percent.

We may consider safe to skip the design documentation when the project is small and often web/scripting based. However if a reconstruction of the program is requested, without having design documentation the team possibly will lose most of the time trying to understand architectural, technical and functional concepts of the project even if the code is well documented. This also refers to eventual attempts to re-use all or some portion of the code in other projects. Concerning the work on a huge project without having the above mentioned documentation will probably cause the work being excessively inefficient or even to terminate.

XP relies on refactoring – a technique to improve the quality of the code and to extend program functionality and design. XP suggests that every project and every situation are different, so we have to find a specific approach to each program development. This may easily cause the team to reproduce already written code, which cannot be considered efficient in the sense of XP.

And finally, many of the XP practices are very hard to be entirely followed. For example XP requires having as a member of the team a client representative which is capable of taking important decisions about the project. However, in most of the cases, a person having such authority is so important to the client that he will probably be unavailable for the project.

Obviously XP is an efficient method for software development. However some of the recommended practices are controversial within the context of life critical and highly reliable systems. Other practices can be considered as hard to be completely followed. The lack of design documentation and unimportance of the architecture can be considered as risky decisions. XP development can be improved to become more reliable and suitable for wide range of projects. This should be accomplished by applying a methodology for XP process improvement that will propose required activities to achieve better quality of code and design.

But we might ask a more general question. What would be the final effect of applying together with XP a CMM subset or even a dedicated CMM like ("sui generis") methodology, as is proposed in [6] ? If successful, we would have guaranteed a permanent improvement of the processes in the development organization and, as an

additional effect - a predictability of its results. An XP team aims at quickly producing quality software. Following the XP philosophy it seems more reasonable to concentrate on only one CMM KPA - the Software Quality Assurance. Certainly, any common XP - CMM principle/practice should be observed, but this anyway happens automatically (as demonstrated in [5], [6] and [7].

5. CONCLUSION

Although XP has been introduced in the last few years most of its practices should be thoughtfully considered for any environment. XP was described as a lightweight methodology but it performs efficiently and soon became very popular. It is appropriate for small to medium sized teams and has the advantage to be adjustable for different environments.

Since XP is a defined process for software development it is understandable that there are already existing attempts to reveal a relation between it and process improvement methods. As we showed in our analysis, each of the attempts following its specific goal does not try to identify a specific method for XP process improvement.

XP can be improved to become suitable for different setting and so the practices can be changed appropriately to satisfy the requirements. There is still place to improve some of the practices and their adjustment to specific environment and this should be accomplished by applying a methodology for XP process improvement that will propose required activities to achieve better quality of code and design.

The other possibility is to concentrate only on the quality assurance. Such an approach has the drawback not to ensure the predictability of the organization's processes, but requires much less efforts and still brings a major result - an end software product of quality.

REFERENCES

[1] Beck K., Jeffries R., Anderson A., Hendrickson C., Jeffries Ronald E. Extreme Programming Installed, Addison Wesley Pub.Co, 2000.

[2] Beck K., Embracing change with Extreme Programming, Computer IEEE, October 1999, p.70-77

[3] Jeffries R., What is Extreme Programming, XP Magazine, 08.11.2001

[4] Jeffries R., Are we doing XP?, XP Magazine, 27.11.2000

[5] Jeffries R., Extreme Programming and the Capability Maturity Model, XP Magazine, 01.01.2000

[6] Nawrocki J, Walter B., Wojciechowski A., Toward Maturity Model for Extreme Programming, EUROMICRO Conf.,Sept.2001, Warsaw, IEEE Computer Press, Los Alamitos, p.233-239

[7] Paulk M.C., Extreme Programming from a CMM Perspective, XP Universe, 23.06.2001

[8] Wells D. J., The Rules and Practices of Extreme Programming, http://www.extremeprogramming.org , 02.2001

ABOUT THE AUTHORS

Avram Eskenazi, Phone +359 2 9792873, e-mail: eskenazi@math.bas.bg
Plamen Balkanski, Phone: +359 2 9715518, e-mail: plamen@ikhaya-bg.com