Can Extreme Programming be used by a Lone Programmer?

Edward Akpata
PharmaTimes Ltd
75 Sheen Lane, East Sheen, London SW14 8AD, UK
edward@pharmatimes.com

Karel Riha
School of Computing and Information Systems
Kingston University London
Penrhyn Rd, Kingston upon Thames KT1 2EE, UK
k.riha@kingston.ac.uk

Abstract: This paper briefly describes Extreme Programming (XP) and then poses the question: given the well documented benefits of XP for many types of development projects where programmers work in teams has XP anything to contribute to projects where one developer works on his/her own? Examining published literature from this perspective and drawing on the authors' own experience of software development, certain deductions can be made. The paper concludes that without the benefits of pair programming a lone developer cannot practice XP to the fullest extent. However, other techniques can be used in place of pair programming and any of the other practices and principles of XP can be used to improve a lone programmer's development process in the right circumstances.

1   Introduction

Due to the many well-documented drawbacks of traditional "heavyweight" methodologies (both structured and object-oriented) a new group of methodologies have appeared in the last few years [1]. These were formerly known as "lightweight" methodologies but more recently as "agile" methodologies.

The appeal of agile methodologies is that they have less bureaucracy and instead strike a balance between having no defined process (for software development) and having a too prescriptively defined process. They are also less documentation and more code-oriented (the key part of documentation is the source code) and, most importantly, they are more adaptive (e.g. to evolving requirements) than predictive (e.g. not trying to establish the requirements fully before implementation). Several methodologies now march under the agile banner. They all share similar characteristics but also have significant differences. Examples of agile methodology include Extreme Programming, Scrum, Cockburn's Crystal family, Highsmith's adaptive software development, Feature Driven Development and Dynamic Systems Development Method.

The purpose of this paper is to examine the features of Extreme Programming (often referred to as XP, even before Windows XP) and determine how it can benefit a lone programmer (i.e. a programmer who works mainly on his/her own as opposed to a programmer who works as a part of a well-integrated team). We will start by explaining what XP is.

2   What is Extreme Programming (XP)?

XP is the most popular of the various flavours of "agile" software methodologies. It is a set of values, principles and practices for rapidly developing high-quality software that aim to provide the highest value for the customer in the fastest way possible. XP is extreme in the sense that it takes many well-known software development "best practices" to their logical extreme [2].

Communication is the first XP value. For example, XP takes the "best practice" of "good communication with the customer" to an extreme by recommending that the customer works in the same room as the programmers, interacting with the team as necessary. There is also unhindered communication between programmers. Each morning everyone takes part in a short stand up meeting. Studies have shown that the effectiveness of communication decreases as participants become more remote [3]. XP was developed with four core values in mind: **Communication, Simplicity, Feedback** and **Courage**. From these values twelve core practices were derived.

2.1   Twelve Core Practices

The Planning Game: the business and development teams get together to decide on what features of the required system will be of maximum value to the business.

Small Releases: a simple system containing a useful set of features is put into production early and updated frequently in short cycles.

Metaphor: each project has a "system of names" and description which helps to guide the development process and communication between all parties.

Simple Design: the simplest design is always used to build the application as long as it meets the current business requirements. Do not worry about future requirements as requirements change with time anyway. Refactoring (see below) will ensure that the design is of a high standard.

Testing: software developed with XP is validated at all times. Before new features are added tests are written to verify the software. The software is then developed to pass these tests.

Refactoring: this is a technique for improving the design of an existing codebase. Its essence is applying a series of small behaviour-preserving transformations that improve the structure of the code. By doing them in small steps you reduce the risk of introducing errors [4].

Pair Programming: programmers using XP are paired and write all production code using a single machine per pair. This helps the code to be constantly reviewed while being written. Pair Programming has proved to produce high quality code with little or no decrease in productivity [5].

Collective Code Ownership: all the code belongs to every member of the team, no single member of the team owns a piece of code and anyone can make changes to the codebase at any time. This encourages everyone to contribute new ideas to all segments of the project.

Continuous Integration: software systems are built and integrated several times a day; at the very least all changes are integrated into the main codebase at least once a day. Each build is tested using the prepared test cases.

40-Hour Week: programmers in an XP project normally adhere to a 40 hour working week in order to maintain productivity and avoid burn out.

On-site Customer: one or more customers who will use the system being built are allocated to the development team. The customer helps to guide the development and is empowered to prioritise, state requirements and answer any questions the developers may have. This ensures that there is effective communication with the customer and as a result less documentation will be required.

Coding Standards: everyone on an XP project use the same Coding Standards which makes it easy to work in pairs and share ownership of all code. One should not be able to tell who worked on what code in an XP project.

2.2 Other XP Strategies

Incremental change: Big changes can be risky and prone to failure so only small changes are recommended.

Small initial project investment: XP projects are started with a small number of developers and then built up, as more developers are required.

Stand up Meetings: meetings are held physically standing up to keep the meeting brief, at the same time each day. The purpose of this is for members to report problems but no solutions are proposed. The developers then leave the meeting and ponder on the solutions.

Tracking progress: a designated team member is responsible for tracking the progress of other team members.

Minimal documentation: documentation is kept to the barest minimum.

Teach Strategies: to enable staff to learn, e.g. how much testing should be done.

Experiment: experiments are carried out to reduce or eradicate the risk of incorrect technological decisions.

2.3 When should Extreme Programming be used?

Extreme Programming (XP) was created in response to problem domains where requirements change frequently [6]. XP will be more effective than "heavyweight" methodologies in situations where customers are not sure about functions they want or are likely to change their mind every few months. XP is ideal for developing new types of software products and as such is a challenge for the developers. It is also suitable for projects where a system has to be developed within a strict time constraint.

3 XP for a Lone Programmer

Most of the published material on XP focuses on situations where at least two programmers work together in a team. Several of XP practices require teamwork, e.g. having someone else reviewing your code. XP takes that to an extreme by ensuring that all production code is written by two programmers who pair up and constantly review each other's work.

However, a lone programmer using XP is not a completely unheard of idea. There is even a term used to describe it, XP41, i.e. "XP for one". The following sections will explore how a lone developer can use XP to improve his/her process. We shall first consider some possible lone programmer scenarios.

3.1 Lone Programmer Scenarios

The term lone programmer can apply to developers in several situations. We briefly look at some of the most common scenarios.

One-man-shop: a situation where you are the only programmer in your company. The advantage is that you will not have to convince anyone that XP is good; you have a complete control of the project. Your customers may be external or internal, or you may be your own customer.

Remote worker: a similar situation to being a one-man-shop in that you work mainly on your own. But you have colleagues who are making changes to a shared codebase and may be available to brainstorm, talk and have reviews with you. You can implement some XP practices that don't impact on other team members. But if your "remote" colleagues practice XP to the best of their ability then it becomes easier. Remote pairing has proved to be difficult although some, such as Alan Cameron Wills [7], have reported having success with it.

Non-extreme team: you may be working in a company with other team members but XP may be forbidden in your company. In this situation some XP practices such as writing tests before coding could be frowned upon. If caught up in this scenario then Martin Fowler's advice [8] may be very handy "*Change* your organisation or change your *organisation*".

Private Developer: similar to a one-man shop but you are your own customer and have full control, you may be developing software for you own use, with the intention of selling it later. Or you may be developing software for other people to use for free, such as in the Open Source movement. The scope and timescale are smaller. Using The Planning Game is always a good idea if only as a way of documenting ones ideas.

We shall now consider three categories of XP practices, depending on how difficult it will be to make them work for a lone developer: **Will work as they are, Some work required**, **Will not work easily.**

3.2 Practices that will work as they are

Among the three types of practices described in the previous section, the XP "programming practices" are those that are most easily adopted by a lone developer.

3.2.1 Small Releases

Early and often releases can be applied as easily with a lone programmer as with a team or developers.

3.2.2 Refactoring

As a lone programmer one can and should practice refactoring to the hilt except in situations where you don't have permission to change code you don't own.

3.2.3 Testing

Test-driven development is one of XP's main strengths. As a one-man shop or private developer test suits can be easily written and used. If you belong to a non-extreme team then show the rest of the team your results accomplished by test driven development; this will prove that the practice is worth it.

3.2.4 Coding Standards

As a one-man shop or private developer how you choose to code is your coding standard. If you are a remote worker or belong to a non-extreme team then you will either have to follow the team's standards or try to influence the team to adopt your preferred coding standards.

3.2.5   Metaphor

A one-man shop and private developer can use and refine whatever metaphor proves best. Always choose a metaphor that helps you understand the parts of the system you are talking about.

3.3   Practices that can be made to work

There are other practices that may not appear to work unless you are working in an XP team but can be made to work or some benefits can be gained from them depending on the circumstance [9].

3.3.1   The Planning Game

The main activity in The Planning Game is the writing-estimation-prioritisation back and forth negotiation of stories between programmers and customers. A solo programmer with a customer can do this just as well as a team. If the developer is working for himself or knows enough to stand in for the customer, than one only needs to switch roles (role-playing) during planning. A lone developer can achieve other activities like detailed iteration planning (breaking down stories into tasks) as it is a programming activity.

If the customers are available to partake in The Planning Game as in the case of the one-man shop then you are fortunate. In some lone programmer scenarios (e.g. private developer) there probably wont be any customers as such. In such a situation the lone developer has to play both sides of the game, using written requirements to act as the customer and then convert the requirements into stories.

The Planning Game can easily be scaled down for an individual. It is a good practice to write stories on cards, estimate, prioritise, and track when developing alone. Writing and using stories is a good way to break down a big project into smaller chunks that are easier to deal with and manage.

3.3.2   Simple Design

As a lone developer who is a one-man shop, remote worker or private developer then it is easy to maintain a simple design. If you are in a non-extreme team (a team not practising XP) and one which practices big design up front, then you will have to conform in the high-level design but you can still keep detailed and low level design as simple as possible.

An alternate solution would be to use the high level design as a goal to work towards but develop an alternate design that is simpler and better but still delivers all the functionality. If this can be achieved then you may be able to convince others to use the simpler design.

3.3.3   Sustainable Pace

All that is required to do this is to adhere to a 40-hour week and stop working for the day when you need to. This means stop working when no longer productive, are stressed or tired to reduce fatigue and keep you in excellent condition. The trend and culture in the current environment is one in which people or usually overworked; if you are expected to put in overtime by your employers then it will be difficult to maintain a sustainable pace.

3.3.4   Continuous Integration

The aim of continuous integration is to prevent or reduce code from spreading from the main codebase; the more frequently code is integrated into the main codebase the less chances that there will be diversion. There are several scenarios to consider for continuous integration.

If the developer is working alone with no other person making changes to the main codebase (such as in a one-man-shop and private developer scenarios) then there is no problem as the codebase serves as the linear record of one's work. Integration will not cause any conflicts and is trouble free but it is still quite easy to diverge from the main codebase the longer you work without integrating. Therefore continuous integration is still needed. The best way to develop is to work on a task, integrate and then move to the next task. This way divergence is kept to the minimum.

If one is in a non-extreme team and is trying to be extreme, then continuous integration is even more important. You will probably be making changes to other people's code and it is very important that one's changes don't conflict with what other team members are doing. A tool like cruise control [10] is very useful in this situation.

3.3.5   On-site Customer

If you are your own customer (at least initially) then mumbling to ones self without excess is reasonable. If you have other customers such as a one-man shop or someone working in a non-extreme team then communication via email or phone will probably moderate this problem as long as the customers are open to communication that is.

3.4 Practices that will not work easily

XP was developed in a team centric fashion. There are several XP practices that simply require more than one person working together. In order to take full advantage of XP you need to be working in a team of programmers all of whom are using XP. However, there are still some benefits to be gained by adapting the principles of these practices to a lone programmer.

3.4.1 Pair Programming

When working as a lone programmer, you loose the benefits of pair programming but gain an advantage because you don't get interrupted as much, some of the advantages of having a programming partner can be achieved in other ways as will be shown later. As its name implies it is a practice that requires more than one person. This section will discuss the ways of getting the same benefits without having a real partner using other means.

Having someone to share ideas and thoughts with can be achieved in other ways. For example, a friend or a colleague at work who will not mind listening to your woes. In the absence of another human being one will have to make do with a lifeless alternative such as a picture or some other object (e.g. a toy) although one has to be careful not to be viewed as going insane.

Having someone to share your frustrations with can also be achieved without pair programming. You usually get frustrated when you are due for a break. In this situation you should relax, take your mind off the project and do something that will help you unwind. For example, going for a walk, have a quick snack, make a cup of tea or play a game; we would personally recommend "GTA Vice City" as you can vent your anger and frustrations on innocent bystanders.

Having someone to help to keep you on the right track; reminding one to refactor, test etc could be achieved by automating your development environment. If you then carry out certain actions, such as creating a method or writing a function, you will be reminded if you've forgotten to do something (e.g. written the tests). You could also ask a colleague to give you reminders often. Ron Jeffries [11] put it as follows:

"The pair programming partner doesn't have to be a programmer. You don't need to be an expert to say, "Did you run the test?" every five minutes or so. Kids could work well as programming partners."

In the scenario of a remote worker, there are technological solutions such as instant reminders via MSN messenger.

3.4.2 Collective Code Ownership

If you are the only developer in the project then there is no problem as you own all the source code. However, you will not be benefiting from other programmers' input.

In the scenario of a developer in a non-XP team it is a bit more difficult to benefit from this practice as other developers are usually protective of their code. You may have to contend with file locking and class ownership. A partial solution to this would be to seek the permission of the owner of the code to create updateable copies for you. A more radical solution would be to persuade the team to adopt the Collective Code Ownership practice.

3.5 Is XP for One really Extreme Programming?

From what has been written so far we can see that XP for a lone programmer is not really XP but another way of carrying out XP. Ron Jeffries put it as follows [12]:

"Real XP is not the techniques, not the practices. Real XP is the development of software by programmers who are expert in their techniques, keep them honed, and use them with judgement in the light of experience."

What can be said of XP for one is that on current evidence it will probably lead to higher productivity and / or quality compared to the less "agile" approaches.

3.6 A Case Study

We thought it would be useful to provide a real life example of a lone developer using XP rather than just theoretical pontification. This section is based on the presentation given by J.B. Rainsberger [13] to the XP Toronto User Group, describing his experiences while doing "XP for One" at IBM Toronto for five months in 2001 [14]. We have selected points he made we think are relevant to the aim of this paper.

3.6.1 Bad experiences

    a)   Lack of automated acceptance tests led to recurring GUI problems

    b)   No involved customers meant an internal client had to work around my design to implement a key feature

c) Other parts of the system have not-so-easy-to-test design, making it tough to unit test thoroughly; slowed me down

d) No pair programming, which I had been looking forward to

The authors of this paper have had similar bad experiences. c) can be a particular hindrance for those accustomed to the XP "Testing" practice.

3.6.2 Good experiences

a) Worked fewer than 20 hours of overtime in six months, compared to 215 hours between 2000 August 1 and December 31

b) One defect found during integration test; zero defects found by functional test organization to date

c) Comment from colleague: Your code is so easy to read!

d) Reversed general perception of the quality of my work

e) Learning a new way to do things led to renewed interest in the work itself; shifted focus away from the product to the process

Again, there is much here to agree with. One of the common experience of those using XP is how many of the practices can motivate the developers and revive flagging spirits.

3.6.3 What others think...

a) My manager liked my planning technique; she read Planning XP [15] and likes the idea of The Planning Game

b) My closest colleague is sick of hearing about refactoring; when I mention it, he rolls his eyes

c) I have been able to pass on some of my XP coding technique to one colleague; he has been very receptive

d) The top development manager: Yeah. I read about XP. Sounds good, but it doesn't scale

e) Internal clients were impressed with trouble-free integration and a simple API

f) Release managers were annoyed by schema changes late in the game

All interesting observations but the only comment worth making here that d) is a very common misconception – see e.g. [16].

3.6.4 What I think...

a) Easily the best experience I have ever had developing software

b) Sense of completion became addictive; frighteningly pavlovian

c) Making changes any time with confidence is incredibly powerful

d) Satisfied with what I was able to do

e) Disappointed not to be able to pair program

Again – very common views from those involved in XP projects. Lone programmers like ourselves can relate to e) having experienced the joy of pair programming beforehand.

4 Conclusion

It is still a mater of debate whether a lone programmer can practise XP in the normal sense of the term. However, it is highly probable that when a lone programmer tries to practices XP it leads to better productivity and / or higher quality compared to the less "agile" approaches. At the very least using XP will keep the developers' skills honed so that when they join an extreme team they will be ready to go to the extreme.

Of the twelve core XP practices a lone programmer can use the following as they are: Small Releases, Testing, Refactoring, Coding Standards and Metaphor. The following can be made to work with some effort: The Planning Game, Sustainable Pace, Simple Design, Continuous Integration and On-site Customer. The remaining practices need a team of more than one developer but some advantages can be gained from trying to modify them using a bit of imagination: Pair Programming and Collective Code Ownership.

Without the benefits of pair programming a lone programmer cannot practice XP to the fullest extent. But one of the main principles of XP is to take the best principles of other methodologies to the extreme. This principle can

certainly be utilised so a lone developer can be practising Extreme Programming albeit limited to 80–90% of the practices, depending on his/her personal circumstance. To quote Kent Beck [17]:

" we *lighten up* the practices when working alone. For example, we just keep a to-do list instead of going all the ways to stories".

This has been the authors' experience too. Lightening up the XP practices to suit one's personal circumstances is the best way to reap maximum benefits from XP.

5 References

Please note that all the Web pages given below were accessed as recently as January 2004.

1   []   Fowler, Martin (2003, April – last update), "The new methodology", Available: http://www.martinfowler.com/articles/newMethodology.html

2   []   Brewer, John and Design, Jera (2001) "Extreme Programming FAQ", Available: http://www.jera.com/techinfo/xpfaq.html

3   []   Putnam, David and Wright, Graham (2003), "eXtreme Programming – Pushing back the Boundaries", Presentation at the British Computer Society OOPS Seminar, Available: http://www.bcs-oops.org.uk/resources/oops167.ppt

4   []   Fowler, Martin (2003), "Refactoring: Improving the Design of Existing Code", Details available at: http://www.martinfowler.com/books.html/refactoring and http://www.xp2003.org/conference/TutorialsDescr.html#T22

5   []   ManLui, Kim and Chan, Keith C.C. (2003),"When does a pair outperforms two individuals?", in Proc. Fourth International Conference on Extreme Programming and Agile Processes in Software Engineering, May 25-29, 2003, Genova, Italy, Available: http://www.xp2003.org/slides/15.pdf

6   []   Wells, Don (2003), "When Should Extreme Programming be used?" Available: http://www.extremeprogramming.org/when.html

7   []   Wills, Alan Cameron (2003), "Dispersed Agile Software Development and Dispersed Extreme Programming", Available: http://www.fastnloose.com/dad

8   []   Fowler, Martin (2001, May), "Is analysis dead?", Invited talk at XP2001 conference, for details of the conference see: http://www.xp2003.org/conference/program.html

9   []   Astels, Dave (2002, May), "XP for One". Tutorial given at XP2002 conference, summary available at: http://www.xp2003.org/xp2002/tut_desc/Astels.html

10   []   Fowler, Martin and Foemmel, Mathew (2003, probable) "Continuous integration" http://www.martinfowler.com/articles/continuousIntegration.html

11   []   Extreme Programming for one. Portland Pattern Repository Wiki. Online at www.c2.com/cgi/wiki?ExtremeProgrammingForOne

12   []   Jeffries, Ron (2003, November), "Is XP for one really XP?", Email communication at extremeprogramming@yahoogroups.com

13   []   Rainsberger, J.B. (2001), "XP for One at IBM", Available: http://www.diasparsoftware.com/portfolio.html

14   []   Rainsberger, J.B. (2003, November), "XP as an individual programmer?", Email communication with Edward Akpata

15   []   Beck, Kent and Fowler, Martin (2000), "Planning Extreme Programming", Addison-Wesley Pub Co.

16   []   Rumpe, Bernhard and Scholz, Peter (2002), "A manager´s view on large scale XP projects", in proc. XP2002, Available: http://www.xp2003.org/xp2002/atti/Rumpe-Scholz--AmanagersviewonlargescaleXPprojects.pdf

17   [] Beck, Kent (2003, November), "XP for one", Email communication with Edward Akpata