Information Systems Methodologies

Assessment 4

An Essay on Extreme Programming

F21IF

Boris Mocialov
Assem Madikenova
Max Baird

Heriot – Watt University, Edinburgh

October 2014

Date: 05.11.14
1 Table of Contents

2 Introduction ............................................................................................................................... 2
3 Context ........................................................................................................................................ 2
4 Philosophy ................................................................................................................................ 4
  4.1 Core Principles ...................................................................................................................... 4
  4.2 Core Practices ...................................................................................................................... 5
5 Possible applications and limitations ......................................................................................... 7
  5.1 Possible applications ........................................................................................................... 7
  5.2 Limitations ........................................................................................................................... 9
6 Conclusion ................................................................................................................................ 10
7 References ................................................................................................................................ 11
8 Bibliography ............................................................................................................................ 12
2 INTRODUCTION

The purpose of this report is to explore, discuss and evaluate XP (eXtreme Programming) software development methodology. First, reader will be given an overview of the more general agile methodologies paradigm and underlying core values that are inherited by the XP in the context section. Second, philosophical overview will introduce structure of XP and its underlying practices that are derived from methodologies’ values and principles. For convenience, figures in this report encapsulate most important aspects of the methodology to give another perspective to the reader. Third, in possible applications and limitations section, critical evaluation of a hypothetical organization, that employs XP as its business structure, will be given. This section is synthesized from analysis of multiple factual surveys and reviews of practical implementations of the methodology in various organizations. Impact XP has on organizations is then divided into two dichotomous sets, where each set is evaluated separately based on its effects.

Evaluation will refer to frameworks that evaluate methodologies, so the reader is advised to have some prior knowledge on them, since this report will not give any such information.

At the end, conclusion will state possible and challenging applications of XP, while justifying such decisions based on preceding analysis given in possible applications and limitations section. While reading this report, it should become clear that XP tries to protect itself from possible criticism against its use by enforcing practices and distributing responsibilities.

3 CONTEXT

Agile methods are defined as increasing software development methods with short increments. These methods divide the system logically by several parts and provide for its’ customer completed piece of system step by step until the end of the project. Customer’s involvement is demanded to derive swift response on changing requirements. Such practice allows shortening the amount of documents, replacing them with informal and face to face interactions (Sommerville, 2011, p. 58)

(Greg Pearman, 2006, p. 4) Point out that agile methods of software development appeared with growth of Internet technologies. As a result business and economy also migrate to the web environment which requires rapid deployment of software development. Changes in the economy have required
quick response of developers in a short time period. These demands led some developers to search alternatives to waterfall methods that need more time for implementation and cannot cope with rapid changes of business environment.

Both of aforementioned sources emphasized that agile methods emerged in the 90s from the dissatisfaction of software development members who were working with waterfall methods before. Waterfall methods were not allowing them to get necessary deliverables that software development team considered significant. They faced a number of drawbacks coming from heavy accent on process and documentation. Afterwards those former users of waterfall methodologies decided to establish Agile Alliance, new approach of problem solving in software development which gives team members more flexibility on their work and lets them avoid enormous documents which are sometimes even useless. In 2001 in Utah, USA seventeen members of the alliance negotiated four core values, that they felt all agile projects should have in common. They called the collection of these values the Agile Manifesto (Highsmith, 2001). These core values are the basis for agile methods:

<table>
<thead>
<tr>
<th>Individuals and interactions</th>
<th>over</th>
<th>Comprehensive documentation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Working software</td>
<td>over</td>
<td>Contract negotiation</td>
</tr>
<tr>
<td>Customer collaboration</td>
<td>over</td>
<td>Responding to change</td>
</tr>
<tr>
<td>Responding to change</td>
<td>over</td>
<td>Following a plan</td>
</tr>
</tbody>
</table>

The founders of agile manifesto concluded these values with following words: “That is, while there is value in the items on the right, we value the items on the left more” (2001).

Flexibility and convenience of agile methods encourage a number of methodologies to use its features. One of such methodologies is Extreme Programming (XP for short), which appeared nearly at the same time as agile methods, which inherited afterwards the key concepts of Agile Manifesto.

XP is an agile software development methodology which defines software development practices and principles on a high level, using four key agile values on a daily basis. The essential feature of XP is the readiness to meet natural changes that occur during the software development. These factors makes XP highly efficient and effective tool of software developing (Scott Millett, 2011, pp. 29-51).
4 PHILOSOPHY

XP is a radical approach to software development and this is reflected in its practices. This approach to software development requires a high degree of developer discipline along with continuous customer involvement for the duration of the project. Extreme Programming takes traditional developmental practices to ‘extreme’ levels; test-driven development, design as needed (any design not pertinent to current release cycle is deferred) and pair programming which is the methodology’s form of a formal review. (Sommerville, 2011). The following sections briefly discuss XP’s philosophy, and provides an overview of its phases.

The philosophy surrounding XP is grounded in four key values and a set of fifteen principles (Greg Pearman, 2006, p. 5). For brevity, we will only list the key values which are: communication, simplicity, feedback and courage. These values establish a foundation on which all XP principles and practices are built so that the philosophy of XP may be realized - that is, the establishment of a better development process through embracing the change which naturally occurs when developing software. (Pro .Net 2.0 Extreme Programming, p.5)

4.1 CORE PRINCIPLES

XP’s values lay the foundation for a set of five core principles that implement the aforementioned values; rapid feedback, simplicity, incremental changes, embrace change and to deliver high quality work. The first of these principles is XP’s accommodation for rapid feedback; XP’s short, incremental delivery stints (usually two weeks) ensures that intended users receive prompt feedback at the completion of each story. Developers will likewise receive prompt feedback from the onsite customer who is a part of the team. Feedback is also derived from XP’s practice of test-driven development, which provides instantaneous results of the correctness of a particular implementation. Rapid feedback leads us to simplicity; this is another key principle that successful teams need to exhibit and entails completing each task in as simple a manner as possible. This simplicity in approach encompasses code, documentation, processes and the form of communication used. By keeping things simple the team is able to focus on just what is needed to satisfy the requirements for the current iteration.

Simplicity is supported by XP’s principle of making incremental changes, the methodology advocates against making large changes for the simple reason that a small bug can be difficult to track in a large body of code. Incremental changes allow for the quick identification of bugs since integration of changes
are only allowed on the premise of successful testing. Incremental development facilitates change embracement and effective change management. If customer requirements change between iterations, the team needs to adjust and adopt the new requirements for the next iteration. Finally, the end result of following the principles of XP should be a high quality product that fits the user’s requirements exactly and is stable.

4.2 Core Practices
The implementation of the values and core principles are executed through essential practices advocated by XP: onsite customer, paired programming, collective code ownership, refactoring and testing. This section will provide a brief overview of these core practices.

The onsite customer minimalizes the need for extensive, upfront requirements gathering and analysis which is subject to change throughout the development process. However, the customer does not always know exactly what they desire at the beginning of a project which therefore makes any requirements specified at that point unstable. XP therefore requires a representative of the customer to be ever present throughout the project. The onsite customer’s role is to provide constant feedback in such a way that the project remains finely tuned to current requirements.

Paired programming entails the physical sitting together of two programmers at one machine to work on a programming task. The benefit of this is an informal code review which is often skipped if scheduled subsequent to programming. Also, an extra pair of eyes aids in keeping code simple while providing ideas and support where necessary. An extension of paired programming provides for collective code ownership; what this means is that teams do not work in segregation, rather, the pairs of developers work on all areas of the system to prevent islands of expertise (Sommerville, 2011). This results in all developers bearing a responsibility for all the code.

The fundamental characteristic of XP are short iterations, to keep code maintainable and simple refactoring needs to be performed as soon as code improvements are found (Sommerville, 2011). In this way the external behavior of the code is maintained with an improved internal structure. Testing is the final key aspect, high quality code is sustained through test driven development and acceptance testing. The former requires tests to be written prior to code which means a focus is kept on what is required of the code and the latter ensures that what is accepted was required. The illustration which follows is a depiction of the interdependent XP practices.
In extreme programming, stories reflect requirements, these requirements are then broken down into a series of tasks where a series of tests are then written for each task by a pair of programmers. These tests must be successful when the code developed for the current iteration is integrated into the existing system. Once successful, the software is released for evaluation and the cycle repeats.
5 POSSIBLE APPLICATIONS AND LIMITATIONS

Let this section begin with a statement that would say, prior to discussing possible applications and limitations, what XP is not about. This methodology is not about covering the whole life cycle of system development, nor does it promote hacking (English, 2002). This statement puts some constraints and defines somewhat vague boundaries to adequate applications of XP. The Philosophy section which outlined structure as well as aims of the methodology should make it clear for the reader what the statement communicates.

Having said that, it is time to identify the possible applications of XP and take a look at its limitations. In the inception phase, any software development project starts within problem owner’s context, where problem occurs. Occurred problem is defined and a decision is made whether it is appropriate to contact a software development team (may it be internal department within organization or external service provider) to come up with a solution to a defined problem. Let’s assume that our software development team’s business strategy is built upon XP and its underlying principles, practices and values, discussed in context and philosophy sections. What would we be able to see if we would look closely at that business?

5.1 POSSIBLE APPLICATIONS
First of all, an organization with such underlying business strategy, would be of small- to medium-size (Michela Dall’Agnol, 2003) This is a crucial factor for the methodology to pay off, since constant face-to-face communication is one of the main requirements to be able to cope with regular uncertainty in an environment, may it be the lack of requirements, lack of knowledge or decision making. If organization is to cope with such factor, its maturity level, as defined by Capability Maturity Model, apriori is high, since to be able to comply with uncertainty, organization must learn from previous projects. Such hypothetical organization might be distributed to some extent, with team members of different languages, cultures, values, expectations as well as mental constructs. (Scott Millett, 2011) In such case, organization would need to adopt a set of conjectures in order to succeed (Lucas Layman, 2006) Distribution both opens new possibilities to businesses and is a major source of stress and disappointment at the same time. Despite such complexities, distribution should be considered when thinking of exploration of the available resources, but it must be handled with care. NIMSAD framework does not explicitly specify how to tackle distribution of problem solver (a team in organization’s case) albeit problem solver context evaluation may be applied to a distributed nature of a team. In this case,
organization would need to deal with this factor based on review of its goals and values and how ‘non-traditional’ team may be able to be as one preserving the motivation and apprise organizational goals and values. An alternative, and more sensible way for an organization, is to select a local representative that would act as a direct problem solver on behalf of a distributed team. (Fitzgerald, 1998) Points that “methodologies tended not to be used when the levels of in-house development were low and the levels of outsourcing and customization of packages were high”, as this statement is valid for distribution of a team as well.

Second, organization would be constituted of cohesive, experienced and settled members. (Reifer, 2002) Natural consequence of methodologies principles, practices and values would suggest that team members would have to be a part of the same team for a long time to be able to keep and build on the existing knowledge base of an organization/team. Such consequence together with XP, in turn, would imply that the team would have solid collective ownership of the produced code and devised metaphors. A long-term relationship within an organization would suggest high-level (as well as similar) experience of every team member. Such factors would in turn positively affect swiftness and quality of produced functionality. (Maurer, 2002) In addition, such organization would bridge, otherwise common and negative, cognitive and social chasms within the team, which were mentioned in (Dubinsky, 2003) paper. But, on the other hand, NIMSAD is concerned whether experience and stagnated behavior of team members separately and as a whole would impact overall bias and prejudice towards separate projects as XP does not address possibility of occurrence of these traits.

Third, such hypothetical organization would, in theory, not require closely controlling managerial organ due to the nature of the practices XP promotes (Gert van Valkenhoef, 2010). Despite the obvious risks, XP tries to reduce managerial overhead with a number of its practices (i.e. planning game, collective ownership, standardisation, etc.). Setting as this would put more pressure on the team members, but extensive experience and cohesiveness would support coping process in deliverance of high-quality products. CMM tells us that weak management would degrade maturity of an organization, on which XP replies by enforcing separate team members to take responsibility of a managerial organ inside their teams.

Based on these properties, XP would be appropriate for a company which would be able to appropriately mimic setting and behavior of hypothetical organization described over.
5.2 LIMITATIONS

XP falls victim to similar criticisms as its agile counterpart since the promoted values, principles and practices foster an agile approach. XP is mainly criticized as being only fit for small to medium sized organisations and problem situations; according to Beck, (Beck, 2000, p. 155) XP can only be used in small to medium sized groups. Glass (Robert L. Glass, 2003, Questioning the Software Engineering Unquestionables) suggests that XP should not be used for life or safety-critical systems.

XP prioritizes development over documentation, however, documentation allows for knowledge and project management (Neill, 2003). This indicates that the lack of formal documentation contributes to a system that is difficult to understand to those external to this development methodology. This leads to further difficulties in questioning design decisions and assumptions made by system developers; the customers and future developers can potentially fall victim to these difficulties. In essence, a system consists of more than “getting the code right”, formal documentation plays a key aspect.

The human-centric approach of XP where the methodology surrounds the developer (Neill, 2003). The skills and knowledge of the developers are therefore critical for the project success, but XP considers no analysis phase of the developers (NIMSAD: analysis of methodology user). The report writers believe that this leads to question whether it should be assumed that developers are willing to share code, work in teams and to support big change in their culture. To ensure that developers (methodology users) incorporate these features, there must be an analysis prior to project commencement. XP only provides a description of team member responsibility (Greg Pearman, 2006, p. 19) but does not guide the methodology user on identifying such individuals.

Another point of critique lies with the customer involvement that is required for the duration of the process. It may not always be possible for the customer to provide an on-site employee capable of fully expressing all requirements and making decisions with respect to acceptance testing. The on-site employee is yet another element of XP which contributes to its human-centric approach that requires no prior analysis to determine suitability. The methodology promotes the on-site employee as the driver for development who is responsible for steering the priorities and defining the business value (Greg Pearman, 2006, p. 19). According to Avison and Taylor (Taylor, 1997) this suggests that the methodology can only be applied in situations where the customer has clear objectives.

To the aforementioned authors, XP’s incremental approach, which lacks any analysis phase, may not necessarily capture the underlying requirements of complex problem situations. Requirements are
primarily provided by the on-site customer and is often blindly accepted without any evaluation (similar to RUP). The values, principles and practices as put forth by XP are interdependent and should be questioned since problems in one core element can potentially destroy the entire construct. These authors therefore argue that the methodology’s lack of high efforts on maintaining and fulfilling all elements is essential to its success.

In the authors’ opinion, an evaluation of XP using NIMSAD would probably come to the conclusion that it neglects many of the elements that could be considered in a problem situation / problem solving process.

6 CONCLUSION

Discussion section “Possible Applications and Limitations” looks independently on possible and challenging applications of XP within an organization, critically justifying decision based on analysis of the methodology and its practices. Analysis is performed from both sides, which allows dichotomous arguments of the subject. The key points that were pointed out on the possible side of employing XP inside an organization are: 1. Size of a company 2. Cohesiveness and 3. Management reduction. On the other hand this essay questions some of the principles and values that XP promotes. Mentioned authors state that XP makes a lot of assumptions, that can fail in software development teams and that this can destroy the whole XP approach. In addition, we can see that CMM would make it difficult for XP to be chosen in one or another organization, but XP, in its turn, responds by placing many constraints on distribution of responsibilities and its peculiar style of documentation.

Now, conclusion can be deduced regarding applicability of the methodology. XP would be applicable as long as an organization would strictly follow principles and practices of the methodology (principles may be specifically adjusted). Organization would also need to appreciate the values that are imposed by Agile Manifesto and are consumed by XP. Any adjustment to the structure of the methodology would need to be weighed carefully and organization would have to make sure that such a change would not violate agile values and, consequently, XP principles and that it would support current inner workings of the team. Additionally the urge for using XP has been identified as being strongly influenced by the environment that surrounds organisations.
7 References

- Donald J. Reifer, 2002, How to Get the Most out of Extreme Programming/Agile Methods
- Frank Maurer, 2002, Supporting Distributed Extreme Programming
- Lucas Layman, Laurie Williams, Daniela Damian, Hynek Bures, 2006, Essential communication practices for Extreme Programming in a global software development team
- Michela Dall’Agnol, Andrea Janes, Giancarlo Succi and Enrico Zaninotto, 2003, Lean Management – A Metaphor for Extreme Programming?
- Pete McBreen, 2002, Questioning Extreme Programming, Pearson Education
- Robert L. Glass, 2001, Extreme Programming: The Good, the bad and the bottom line, Software IEEE, Volume 18, Issue 6
- Scott Millett, Jerrel Blankenship, Matthew Bussa, 2011, Pro Agile .Net Development with Scrum
- Brian Fitzgerald, 1998, An empirically-grounded framework for the information systems development process
8 Bibliography


