e-Informatica Software Engineering Journal, Volume 1, Issue 1, 2007

Programming in the eXtreme: Critical Characteristics of Agile Implementations

Gerald DeHondt II*, Alan Brandyberry*

*Management & Information Systems Department, Kent State University

gdehondt@kent.edu, abrandyb@kent.edu

Abstract

The prevalence of systems development project failures has been well documented. eXtreme Programming (XP) is a software development methodology that seeks to eliminate many of the shortcomings of cumbersome life cycle oriented traditional methodologies. We explore some of the basic tenets of XP and Agile methodologies and present the thoughts of two of the proponents and early participants in the "Agile revolution", Chet Hendrickson and Ron Jeffries. We analyze this interview utilizing an interpretive field study employing a hermeneutical circle technique. Our analysis suggests some of the characteristics of XP implementations are more critical than others. We propose a more concrete definition of what XP represents and suggest areas for future research.

1 Introduction

Modern software development efforts often follow traditional software development methodologies such as the Systems Development Life Cycle (SDLC), with organizations typically making certain adjustments to bring these traditional methodologies in line with their organizational needs. While utilizing this method is acceptable, it often leads to cost overruns and longer development cycles than originally anticipated, and user requirements that remain unmet. These types of problems can wreak havoc with corporations' IT strategy and planning functions, as well as have significant negative impact on the business. Recently, certain "Agile" development techniques have been introduced that seek to provide shorter development cycles, with the associated cost savings.

Specifically, eXtreme Programming (XP) is an Agile development technique which emphasizes rapid and frequent feedback to the customers and end users, unit testing, and continuous code reviews. By focusing on rapid iterations of simpler code, XP seeks to identify and resolve potential pitfalls in the development process early, leading to projects that remain focused on the ultimate goal – timely delivery of a well-designed and tested system that meets customer requirements.

XP breaks down a project into sub-projects, each including planning, development, integration, testing and delivery [21]. Developers work in small teams with customers as active team members. Features are implemented iteratively during each development cycle with joint decision making occurring between the customer and the rest of the development team. Agile software-development methods use human- and communication-oriented rules in conjunction with light, but sufficient, rules of project procedures and behavior [8]. They

rely on planning, with the understanding that everything is uncertain, to guide the rapid development of flexible systems of high value [20, 21]. eXtreme Programming has been described in terms of the values that support it: communication, feedback, simplicity, courage, and respect [4]. This methodology works by bringing the whole team together in the presence of simple practices, with enough feedback to enable the team to see where they are and tune the practices to their unique situation.

2 Extreme Programming Begins

The eXtreme Programming Methodology was developed and first implemented by Kent Beck at DaimlerChrysler in 1996 as a way to accelerate development efforts, while producing better software for their Chrysler Comprehensive Compensation (C3) Project. The C3 Project had run into an impasse. Previous efforts to develop a new payroll system – one that would support over 86,000 international employees from union represented employees to upper management – had failed, and in early 1996 it was decided to start over from scratch [46]. As background, the new system was to replace three separate legacy systems that had been in place for over twenty years supporting the various payroll groups. Originally, Chrysler had purchased a payroll system from a leading industry vendor. After many months of effort, it was determined that neither this package, nor any on the market, could handle the complexities of the current payroll structure. At this point, it was determined that the system would have to be designed and built from scratch. The previous implementation attempt had used a traditional software development methodology, the "Waterfall" method. The team had found this method too complex and cumbersome and realized they needed to approach the problem in a different way. This is consistent with the previous research of Merisalo-Rantanen, Tuunanen, and Rossi [35] as one of the cases they studied suggested that traditional methodologies were too restrictive and slow, and hence evolved their corporate system development model into an XP-like framework. Additionally, Levina and Ross [28] suggest that having stringent development methodologies might prove detrimental to the success of more creative and innovative application development projects.

At this point, the team decided to apply the practices of eXtreme Programming to develop the C3 System. Over about the course of a year, they were able to architect, design, code, test and implement a system that supported the Chrysler Comprehensive Compensation structure. This is quite a feat when considering that the previous iteration of the project languished for a number of years before being deemed a failure and turning to XP. Fowler [14] argues that these lightweight, or Agile, approaches are necessary due to the high overhead and resource intensiveness of the existing and dominant methods of software development.

There is ample room for improvements to be made in the prevailing software development methodologies. It is well documented that systems development failures are all too common. Although different definitions of what constitutes a failure undoubtedly yield different rates, estimates of failure rates on systems development projects have been purported to range from 50 percent to 75 percent [12, 16].

eXtreme Programming, however, is not without its detractors. As a significant departure from traditional development methods, many corporations are hesitant to risk development projects to a relatively new development technique, especially one that significantly departs from conventional methods. Adoption of an Agile Methodology will also likely pose several challenges for organizations steeped in the traditional systems development methodologies, since the two software development methods are grounded in opposing concepts [37].

3 Background

An Information Systems Development Methodology has been defined as an organized collection of concepts, methods, beliefs, values and normative principles supported by material resources [22]. Initial efforts at formalizing and planning systems development efforts began with "traditional" methods such as the "Waterfall" [6, 23], which then evolved into the Systems Development Life Cycle – a commonly used systems development methodology. However, these traditional methods take a phased approach to systems development, requiring that one phase be completed prior to beginning the next phase [24] and the product is not delivered until the whole linear sequence has been completed. This effectively means that the first day that functionality can be delivered to the customer is the last day of the project (except for the ongoing system maintenance).

3.1 Satisfying Customer Requirements

Extensive upfront planning is the basis for predicting, measuring, and controlling problems and variations during the development lifecycle. The traditional software development approach is process-centric, guided by the belief that sources of variations are identifiable and may be eliminated by continually measuring and refining processes [9]. The primary focus is on realizing highly optimized and repeatable processes.

One measure of an organization's development process maturity is the Capability Maturity Model (CMM). The Capability Maturity Model is a project of the Software Engineering Institute (SEI) at Carnegie Mellon University seeking to identify best practices that may be useful in helping organizations increase the maturity of their software development processes [43]. Organizations will achieve a designation from 1 through 5, based on the repeatability, definition, management, and optimization of their software development processes. This process itself though, is very cumbersome. Even stripped to the bare essentials, the CMM comprises 52 primary goals and 18 key process areas [38]. As one official in the CMM project at Carnegie Mellon University noted, "You can be an [highest CMMrated] organization that produces software that might be garbage." [31] Focusing on the process used to develop and deliver software may not always lead to systems that meet customer requirements.

One of the primary drivers of XP is the focus on delivering the features the customer wants. Under traditional software development methods, planning and control accomplished by a command and control style of management provide the impetus for developing a software product [21]. Traditional methodologies assume that problems are fully

specifiable, and that an optimal and predictable solution exists for every problem [7]. In the instance that there is a change in design or user requirements over the course of the project – a situation all too common in real-world corporate development environments – the methodology begins to break down and efforts become focused on reworking previous design and development activities. These changing requirements cause additional rework in the project. Gopal, Mukhopadhyay, and Krishnan [17] found that clients who request rework later in the lifecycle increase rework – and cause project delays – considerably, with requirements volatility having an even stronger influence on rework. A methodology not appropriately suited to changing requirements causes delays in system delivery.

Traditional methodologies are also too set and too full of inertia to be able to respond quickly enough to a changing environment to be viable in all cases [12]. Existing heavyweight processes tend to be predictive and slow [45] and are unable to efficiently react to changing circumstances. It is often suggested that these traditional methodologies are useful in situations where requirements are well-known and unlikely to change but that assertion is also challenged by some in the XP community. They can also be used when significant project management overhead must be incurred, as in large, business-critical systems built with teams in excess of 50 members [14]. Lightweight or Agile methods, on the other hand, are seen to be more code-oriented, more people-oriented, and more adaptable to change suited to relatively small projects with rapidly changing requirements [19]. Others, however, find significant scalability in XP through project decomposition [4].

These evolving requirements are often viewed as an inherent problem of software development in traditional methodologies [47]. In a sense, taking one step forward and two steps back. On the other hand, the Agile community views requirements changes as an opportunity to provide software that can enhance the customer's competitiveness in a rapidly evolving marketplace [47]. Development teams that can handle these changes will produce software that is more useful to the customer, leading to more satisfied customers. XP seeks to support timely and economical development of high-quality software that meets user requirements at the time of delivery. XP utilizes an iterative approach that is helpful in developing, modifying, and maintaining systems more quickly and more successfully [2, 5]. It is these short iterations that provide the flexibility to accommodate the changes requested by the customers and allows the customer to increase competitiveness in the market [47]. In fact, XP seeks to implement the simplest design that will satisfy current user requirements [29] without attempting to anticipate future design or user requirements. The iterative nature of this methodology enables it to be tolerant of changes in requirements [4].

3.2 Organizational Influence

As the traditional SDLC has become embedded and institutionalized in organizations as the standardized method of systems development, any changes to a new approach will require a shift in the organizational norms. This can be one of the most difficult obstacles to implementing XP. Based on the team approach of XP [29], those with individual influence under the traditional methodologies will have to acquiesce for the greater good of the team. Organizations using heavier methodologies typically had trouble adopting the incremental

release approach of Agile Methodologies because of the implications of the core practices: simple design, testing, refactoring, and continuous integration [12]. These core practices require that everything be available. For example, a daily build means that the testing suite must also be ready daily, which also has implications for continuous integration and refactoring. Specifically, Project Managers have been viewed as the overall leader, with significant authority to make decisions impacting the project. In an XP environment, the project managers give up much of their decision-making authority to the team [35]. This "Method Adaptation" [34] is the process or capability of agents, through responsive changes in and dynamic interplays between contests, intensions, and method fragments, to determine a system development approach for a specific project situation.

It is generally accepted that there is no single process that will be applicable to all projects [47]. There are a number of best practices, techniques, and experiences that developers can use in appropriate situations. Software development teams with leaders that understand the situations in which particular processes are applicable are more likely to be successful within an Agile environment. Remember, Agile processes are not silver bullets. Because these assumptions are not met in all development environments, it is possible to extend Agile processes to address their limitations. These extensions can involve incorporating principles and practices associated with traditional development processes into Agile processes. Users of Agile processes need to ensure that practices based on invalid, environmental assumptions are modified accordingly.

3.3 Values and Practices of XP

XP, at its core, rests upon the following values: communication and feedback, simplicity, and responsibility [47]. Project success, and delivering software that meets the customer requirements, requires frequent, face-to-face communication between developers and customers. Not only is this explicit communication, but also delivering working code incrementally at frequent intervals. Ultimately, this is the best check to demonstrating understanding of customer requirements. Simplicity is embodied by delivering only solutions that meet current user needs. XP does not attempt to design or anticipate for future needs. Ultimately, the customer is concerned with addressing current needs, and is not interested in what may occur later. Finally, the responsibility of delivering high-quality code rests with the developers of the system. Agile methods focus on people as the primary drivers of development success [10]. They are those closest to the solution and should be the most knowledgeable about how the solution will be implemented.

These values are expanded into the twelve practices of the XP Methodology: planning, small releases, metaphors, simple design, testing, refactoring, pair programming, collective ownership, continuous integration, on-site customers, coding standards, and 40-hour week [3]. Similar values are embodied in the Agile Manifesto [11]:

1. Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

2. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

3. Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
4. Business people and developers must work together daily throughout the project.
5. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.
6. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
7. Working software is the primary measure of progress.
8. Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
9. Continuous attention to technical excellence and good design enhances agility.
10. Simplicity – the art of maximizing the amount of work not done – is essential.
11. The best architectures, requirements, and designs emerge from self-organizing teams.
12. At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

These values implement the best practices of previous Systems Development Methodologies representing an evolutionary process growing out of a natural and useful way to develop software [35]. They provide a number of instances in their study where XP-like methods simply evolved as logical ways of doing things. This evidence provided support for their claim that XP is more of a packaging of best practices, rather than a totally new way of doing things, formalizing habits that work in certain organizational settings for delivering better software.

This evidence remains consistent with the core XP beliefs, especially given the flexibility that the methodology provides in adapting the values to a given situation. These beliefs include user involvement, simple code, iterative development, and courage. McKeen, Guimaraes, and Wetherbe [33] argue that user participation improves the quality of the system in several ways such as providing a more accurate and complete assessment of requirements, expertise about the user organization the system will support, avoiding development of unacceptable or unimportant features, and improving user understanding of the system as it is being developed. XP engages the end users in the IS development process.

Kent Beck [4] states that the programming strategy of XP is to keep the code easy to modify. Iterative development requires that each developer must integrate and release code at least once a day after passing all the unit tests or completing a smaller part of planned functionality [25]. This continuous integration detects compatibility problems early and ensures everyone is working with the latest version of the system.

Along with the iterative nature, XP incorporates rigor into the methodology. XP stipulates that developers follow all of the practices to realize the benefits of Agile Development [47]. McBreen [32] points out that XP requires enormous discipline to implement, and in fact, some projects may find it difficult to adopt a true XP-compliant process.

Another of the main beliefs of eXtreme Programming is courage – for developers to rework their code when it doesn't perform as expected, to stick to the chosen approach when the going gets tough, and to make the hard decisions required to deliver the software [1]. In order for XP to work, corporations – or at least their IT departments – must be willing to take the necessary risks to depart from conventional techniques, and have the courage to continue along this path during the difficult times. IT departments must be able to adapt and implement new techniques when the situation is called for, as opposed to holding on to old methods that produce the same inadequate results. Organizations must be willing to change and implement new techniques to develop better software.

Exhibiting the fortitude required to undertake some of these efforts will allow additional flexibility – agility – in the development of the system that in the end will provide a better system, more flexible to changing user requirements, and implemented in a fashion that meets the goals of the organization.

Previous research by Merisalo-Rantanen, Tuunanen, and Rossi [35] provide anecdotal evidence that both developers and internal and external customers were satisfied with the results of their projects developed using eXtreme Programming. A number of other studies and experience reports also provide claims for the successful use of eXtreme Programming and Agile Methods [18, 30, 39, 42, 44, 46] in practice. Additionally, experimental studies [15, 36] have focused on particular techniques used within eXtreme Programming such as pair programming and test-driven Development (TDD). With the relative recency of eXtreme Programming as an approach to software development, results of a survey by Rumpe and Schroder [40] show that this methodology is still in the "hype phase". However, numerous opinion pieces and several surveys clearly demonstrate the growing popularity of Agile Methodologies [37].

3.4 A Broad-based Perspective of XP Implementations

The literature on eXtreme Programming is primarily focused on review of the practices and techniques, and how this methodology differs from traditional methodologies. In this context, there have been a few examples of anecdotal studies of individual projects and experience reports focusing on a small number of projects. This study reviews a multitude of projects from the practical experiences of two of the initial implementers of these practices. The purpose of this work is to gain insight across a number of projects from these practitioners based on their implementation experiences. It is from this broad-based perspective that we can learn the appropriate positioning of eXtreme Programming within an organization's systems development toolbox.

This interview focuses on the types of organizations that may successfully implement XP techniques, including some methods to integrate XP into an organization focused on rigid, top-down development methodologies, further explanation of how XP differs from

traditional methodologies, and provides a means of identifying a true XP Project, including ways to determine whether the practices are really being used.

As is commonly seen when innovations are introduced, there is some confusion as to what is and is not XP. In casual conversations with people generally knowledgeable in the area of systems development the perception that XP is using "no methodology" at all is sometimes expressed. This confusion is probably due predominantly to the emphasis on values [4] that, on the surface, appear to lack the structure that other methodologies emphasize. What is misunderstood is that these values must be manifested in principles and then actual practices. This allows substantial flexibility in how an organization actually implements XP. Although this is viewed as a strength by XP proponents, it also contributes to the confusion around XP. There exists no objective test as to whether an organization is employing XP. As the interview subjects suggest, there are cases where XP has been identified as the development methodology that others have questioned. Obviously an organization has not truly implemented XP just by uttering the words "eXtreme Programming". In the interview in the following section we engage these experienced practitioners in a general discussion of XP to attempt to discover the themes that they express as being critical to XP implementation. We then employ a hermeneutic circle technique to extract the critical characteristics of XP implementations.

4 Methodology

This analysis is an interpretive field study as used by Fitzgerald et al [13] and Sauer and Lau [41] when studying the practical use of a method. This has been suggested as an appropriate research method for explorative and descriptive types of research by Klein and Myers [26]. It examines the experiences of two well-known eXtreme Programming practitioners, Chet Hendrickson and Ron Jeffries. They were original team members of the seminal XP Project at Chrysler Corporation in 1996, and have since provided consulting and training services to Fortune 500 companies in the appropriate use and implementation of the eXtreme Programming methodology. They have also spoken at numerous academic and practitioner conferences expounding their knowledge of eXtreme Programming. The question and answer discussion presented in the next section was derived from a series of interviews and follow-up contacts. Gerald DeHondt facilitated these interactions.

4.1 Interview

DeHondt: What kind of organizational model will be most appropriate to implementation of eXtreme Programming techniques?

Jeffries: No project is completed with traditional software development methodologies; it is completed in spite of traditional software development methodologies. It is a combination of the process and the team that leads to successful projects. If it is the wrong combination, the organization will try to kill it.

DeHondt: In these instances, how would you get the organization to change?

Jeffries: A politically safe method is needed for implementing XP in organizations.

This can be accomplished by increasing user feedback to the development staff. In this, we focus on 8–10 stories per month. At the end of the month we determine how many we were able to successfully complete. From there we can adjust the timeline either up or down.

There also needs to be accountability. One of the things agile methods do is provide an API into the organization to produce clear unambiguous information.

DeHondt: Will XP work in all environments? For example, many organizations are focused on top-down, structured, rigid development methodologies. In these instances, how would XP be adapted?

Jeffries: XP has been used in a number of rigorous environments. NASA has developed projects using agile methodologies, McKeen implemented a SmallTalk project, and it has been used by an embedded medical equipment manufacturer for a product requiring FDA approval.

One of the key points is whether the project is utilizing Test Driven Development (TDD) and integrating customer acceptance testing into their development process.

We also want to make sure we are appropriately completing the tasks. For example, if we are 90 percent complete on all of our tasks, we haven't completed any. On the other hand, it would have been better to spend the time to complete 90 percent of the tasks completely.

DeHondt: How does XP differ from traditional development?

Jeffries: What we do is break up the project into smaller pieces. This way we can more effectively monitor whether we are on task with the smaller piece. The error bars will be smaller in this situation.

Consider driving a car. As we drive down the road, we are continuously making adjustments to our direction to stay in our lane. If we made corrections at longer intervals we would risk going off the road before changing our direction. This is the same thing with software development. We are continuously monitoring the project progress by getting feedback from the customer. If we are going a little off course, customer feedback will allow us to change direction before we go off the road. We are able to respond to the environment faster. We are able to learn faster.

DeHondt: There have been some cases where XP methods have not worked, and the projects have failed. Are there any common reasons why an XP project would fail?

Jeffries: In the instances where whatever you did didn't work, maybe these projects weren't using XP, only saying they were. If a project says they are using XP, but not any of the practices, then it is not XP.

In order for projects to be using XP, they need to be able to ship code at any time. In the Chrysler project, at any time, we could provide the customer with the latest build for installation and testing. This is XP.

You also want to get negative work as close to zero as possible. We don't want to have rework when customer requirements aren't properly implemented. By having the customer work closely with us, we are able to monitor and ensure that their requirements are being implemented as expected.

Hendrickson: With project status reporting, we are able to more closely monitor work completed and provide this information to the customer. Do you want to hear the truth about the project, or do you want to hear what you want?

When companies look at project completion measures, there are a number of measures that can be used to gauge project completion. These could include percent of time spent, percent of money expended, or other criteria. XP will look at percent of deliverables satisfied and are we are building what the customer asked for.

We want to minimize the process overhead not related to writing code.

Jeffries: The project needs to have a good communication feedback loop with the customer, be able to test quickly – this may even include an automated test cycle – simple design, code to support the test cases, small chunks of code, and integrated code.

One of the problems encountered is that teams don't know how to ship code regularly. They should be able to provide the latest build of the system at any time.

Hendrickson: Using traditional software development methodologies, a company may spend three months on the analysis phase and three months on the design phase. At this point, the project has been silent for six months. At that point the determination is made if they are on track.

Jeffries: In XP, the architecture and design grow along the way. XP seeks to ensure that the software is working as intended along the way, and the design is good. This will allow the software to change if the customer requirements change.

DeHondt: Now let's shift gears a little to team attributes. The literature has looked at XP and proposed that it is successful because it is staffed with highly qualified people. In this type of an environment, it may be the people that cause projects to succeed, not necessarily the process. What type of attributes does a team require to be successful at an XP project?

Jeffries: A good XP team will have people who are thoughtful about what they did, with good coaching, and determination. The customer has to know what they want in the software. There also needs to be support from the highest levels of the organization. Ultimately, they need to be good people, who are good programmers.

XP succeeds because of commitment from the company, team, management, and the customer. The team needs to embrace and execute the practices of XP, monitor and

adjust. There needs to be codification of best practices and feature by feature analysis. The development process needs to be molded to the situation with the agile development bias towards simplicity and action.

Companies that deliver software on time and on budget do not outsource.

Hendrickson: Agile focuses on people and delivering code to implement features. It breaks down a project into smaller iterations focused on delivering working functionality or requirements. It places emphasis on communication with the customer to determine if the code developed meets the customer's requirements.

Agile is also good at handling evolving requirements, whereas the traditional software development methodologies specify everything up front.

Jeffries: XP also requires everything to be available – the development environment, test environment, everything that will be needed to quickly move through the process and develop working code that meets customer requirements.

Hendrickson: Some XP projects will fail; it is just that these projects will fail sooner with XP. In traditional approaches, it may take longer to realize that the project is not viable. At that point, more time and resources have been expended.

5 Analysis and Discussion

We employ a meaning categorization and hermeneutical interpretation form of analysis as described by Kvale [27]. In the review of the literature in the first section of this research we identified the values, principles, and practices that have been commonly associated with XP implementations. We begin our analysis by viewing each of these as a general theme and shared symbolic vocabulary. We then iteratively explore the interview for the apparent level of importance of each theme by determining the frequency and context in which each theme appeared. In the tradition of hermeneutical circles we iteratively interpreted the meaning of the whole and the meaning of the parts or themes with the goal of deepening our understanding of the meaning within (a *circulus fructuosis*) [27]. We will discuss these identified dominant themes in the derived order of importance (based on number of occurrences and implied criticality) to the overall meaning.

5.1 Continuous Delivery of Working Software

This theme was repeated often and in different ways during the interview. They emphasize completing tasks rather than having many tasks partially completed. They state that projects using XP "need to be able to ship code at any time". Another related concept introduced is measuring percent of deliverables satisfied as the primary metric of project completion. A clear sentiment on this issue is their articulation that "teams don't know how to ship code regularly. They should be able to provide the latest build of the system at any time". Similarly, they state, "Agile focuses on people and delivering code to implement features. It breaks down a project into smaller iterations focused on delivering working

functionality or requirements". There is a clear message throughout the discussion that delivering working software continuously is critical to utilizing XP.

5.2 Customer-driven Process

The concept that the customer must be involved and must, in fact, drive the XP process is expressed several times during the interaction. This is exemplified by statements such as "The customer has to know what they want in the software" and "XP succeeds because of commitment from the company, team, management, and the customer".

5.3 Communication

"Do you want to hear the truth about the project, or do you want to hear what you want?" This statement underscores the importance of good communication in XP implementations. Other statements such as the "project needs to have a good communication feedback loop with the customer" and their reference to "increasing user feedback to the development staff" also demonstrate the importance of this concept.

5.4 Incremental Design and Acceptance of Changing Requirements

These are two related themes that are referenced by the subjects in several instances. We view them as related since the reason incremental design is important is that requirements do often change. Their analogy to driving a car (needing constant adjustment) is related to this theme. They also refer to breaking up projects into smaller pieces, implying separate design cycles for these functions. This theme may be well summated in the statement, "Agile is also good at handling evolving requirements, whereas the traditional software development methodologies specify everything up front".

5.5 Continuous Testing

The subjects specify that "One of the key points is whether the project is utilizing Test Driven Development (TDD) and integrating customer acceptance testing into their development process". They also make reference to being able to test quickly as well as utilizing automated testing.

5.6 Other themes

There were several other themes that were identified by this process. Both the number of occurrences of the theme and the implied criticality lead us to believe that these are less critical (though not necessarily unimportant) themes when trying to conceptualize those few important characteristics that would generally be considered essential. These themes include: the incremental implementation of XP, use of "stories", XP is lightweight, team-talent, all-constituents need to sign on, and that XP may fail but fail sooner.

5.7 Hermeneutical Interpretation of "the whole"

Each of the identified themes appears to have its own individual importance and, at least to these practitioners, certain themes seem to be more critical to the process than others. Additionally, their interaction with each other and less critical XP themes is also significant. For instance, continuous testing is critical if you are going to be able to ship the latest build of a software project at any time. An attempt to synthesize these critical themes into a single defining statement yields the following result:

XP is fundamentally the continuous delivery of incremental working software that is customer-driven and dependent on communication, incremental design, acceptance of changing requirements, and continuous testing.

Although this definition may be incomplete in some ways since XP is possibly more philosophy than methodology, it may serve as a useful starting point in bridging the gap of understanding in what is meant by the term eXtreme Programming. Certainly, it adds a way of expressing the XP concept in the more concrete terminologies that many methodologists prefer.

6 Conclusion

The views of these participants in the Agile revolution hold that eXtreme Programming has substantial benefits when applied appropriately and they report many experiences that support this view. There also needs to be consideration given to whether the project actually implemented XP processes and practices. Investigations of failed "XP" projects indicate that these projects were not, in fact, implementing the values and principles, but simply stating that they were an XP project.

As a relatively new methodology there is need for substantial research into the concept and its implementation. We have attempted to focus the discussion of "what is XP?" on several critical themes. However, this work has been based on the perspective of two individuals. No matter how experienced and well-versed they may be, studies that synthesize a broader array of viewpoints will also be important. Critical questions include: Are there project or environmental characteristics that make other XP themes more (or less) critical? Are there certain "brands" of XP where common sets of themes are utilized together? Are failures due to incomplete implementation of XP concepts, project characteristics independent of methodology, or other factors? The authors suggest performing follow-up work on comparable paired projects – based on application requirements, project size, or project type – that use eXtreme Programming and traditional methodologies as their development approach. This could allow comparability of the methods to similar situations and help determine factors in each methodology or particular project interaction that highlight strengths and weaknesses of each approach.

Many XP proponents seem to imply that XP would be the best choice for all development projects. Proponents of other methodologies seem more willing to say that there are certain project types that a particular methodology is better suited for than others.

An important question that needs to be answered is whether there are project characteristics that make XP more or less likely to succeed. Are there characteristics that would make a particular project unsuitable for implementation using XP?

Some people believe XP succeeds because of the above-average skill level of the team. This study reveals placing the overall attitude of the team as a more salient feature to project success than the aptitude of the team.

References

[1] S. Ambler. The Extreme Programming Software Process Explained. *Computing Canada*, 26:24, March 2000.

[2] V. Basili and A. Turner. Iterative Enhancement: A Practical Technique for Software Development. *IEEE Transactions on Software Engineering*, 1(4):390–396, December 1975.

[3] K. Beck. *Extreme Programming Explained*. Addison-Wesley, Boston, 2000.

[4] K. Beck and C. Andres. *Extreme Programming Explained: Embrace Change 2nd Ed*. Addison-Wesley, Boston, 2004.

[5] B. Boehm. A Spiral Model of Software Development and Enhancement. *ACM SIGSOFT Software Engineering Notes*, 11(4):22–42, 1988.

[6] F. Brooks. *The Mythical Man Month: Essays on Software Engineering*. Addison-Wesley, Reading, MA, 2003.

[7] S. Cavaleri and K. Obloj. *Management Systems: A Global Perspective*. Wadsworth Publishing Copany, California, 1993.

[8] A. Cockburn. *Agile Software Development*. Addison-Wesley, Boston, 2002.

[9] A. Cockburn and J. Highsmith. Agile Software Development: The Business of Innovation. *IEEE Computer*, pages 120–122, September 2001.

[10] B. Conrad. Taking Programming to the Extreme. *Infoworld*, page 24, July 2001. July 21, 2000. available online at http://www.infoworld.com/articles/mt/xml/00/07/24/000724mtextreme.html.

[11] W. Cunningham. Manifesto for Agile Software Development. *[WWW document]. URL http://www.agilemanifesto.org/principles.html*, n.d./2006.

[12] J. Erickson, K. Lyytinen, and K. Siau. Agile Modeling, Agile Software Development, and Extreme Programming: The State of Research. *Journal of Database Management*, 16:88–100, October–December 2005.

[13] B. Fitzgerald, N. Russo, and T. OKane. An Empirical Study of System Development Method Tailoring in Practice. *Proceedings of the Eighth International Conference on Information Systems*, pages 187–194, 2000.

[14] M. Fowler. The New Methodology. *[WWW document]. URL http://www.martinfowler.com/articles/newMethodology.html*, December 2005.

[15] B. George and L. Williams. A Structured Experiment of Test-Driven Development. *Information and Software Technology*, 46(5):337–342, April 2004.

[16] E. Germain and P. Robillard. Engineering-Based Processes and Agile Methodologies for Software Development: A Comparative Case Study. *Journal of Systems and Software*, 75:17–27, 2005.

[17] A. Gopal, T. Mukhopadhyay, and M. Krishnan. The Role of Software Process and Communication in OffShore Software Development. *Communications of the ACM*, 45(4):193–200, March 2002.

[18] J. Grenning. Launching Extreme Programming at a Process Intensive Company. *IEEE Software*, 18(6):27–33, November–December 2001.

[19] B. Henderson-Sellers and M. K. Serour. Creating a Dual-Agility Method: The Value of Method Engineering. *Journal of Database Management*, 16(4):1–23, October–December 2005.

[20] J. Highsmith. *Agile Software Development Ecosystems*. Addison-Wesley, Boston, MA, 2002.

[21] J. Highsmith. *Cutter Consortium Reports: Agile Project Management: Principles and Tools*, volume 4(2). Cutter Consortium, Arlington, MA, 2003.

[22] R. Hirschheim, H. Klein, and K. Lyytinen. *Information Systems Development and Data Modeling*. Cambridge University Press, New York, 1995.

[23] R. Hirschheim, H. Klein, and K. Lyytinen. *Information Systems Development and Data Modeling: Conceptual and Philosophical Foundations*. Cambridge University Press, Boston, MA, 2003.

[24] J. Hoffer, J. George, and J. Valacich. *Modern Systems Analysis and Design*. Addison-Wesley, Boston, MA, 1998.

[25] S. Joosten and S. Purao. A Rigorous Approach for Mapping Workflows to Object-Oriented IS Models. *Journal of Database Management*, 13(4):1–19, October–December 2002.

[26] H. Klein and M. Myers. A Set of Principles for Conducting and Evaluating Interpretive Field Studies in Information Systems. *MIS Quarterly*, 23(1):67–93, March 1999.

[27] S. Kvale. *Interviews: An Introduction to Qualitative Research Interviewing*. Sage Publications, Thousand Oaks, 1996.

[28] N. Levina and J. Ross. IT From the Vendors Perspective: Exploring the Value Proposition in Information Technology Outsourcing. *MIS Quarterly*, 27(3):331–364, September 2003.

[29] L. Lindstrom and R. Jeffries. Extreme Programming and Agile Development Methodologies. *Information Systems Management*, 21(3):41–52, 2004.

[30] P. Manhart and K. Schneider. Breaking the Ice for Agile Development of Embedded Software: An Industry Experience Report. *Proceedings of the 26th International Conference on Software Engineering*, pages 378–386, 2004.

[31] N. Matloff. Globalization and the American IT Worker. *Communications of the ACM*, 47(1):27–29, November 2004.

[32] P. McBreen. *Questioning Extreme Programming*. Addison-Wesley, Thousand Oaks, 2003.

[33] J. McKeen, T. Guimaraes, and J. Wetherbe. The Relationship Between User Participation and User Satisfaction: An Investigation of Four Contingency Factors. *MIS Quarterly*, 18(4):427–451, 1994.

[34] A. Mehmet, F. Harmsen, K. van Slooten, and R. Stegwee. On the Adaptation of an Agile Information Systems Development Method. *Journal of Database Management*, 16(4):24–40, October–December 2000.

[35] H. Merisalo-Rantanen, T. Tuunanen, and M. Rossi. Is XP Just Old Wine in New Bottles. *Journal of Database Management*, 16(4):41–61, October–December 2000.

[36] M. Mller. Preliminary Study on the Impact of a Pair Design Phase on Pair Programming and Solo Programming. *Information and Software Technology*, 48(5):335–344, May 2006.

[37] S. Nerur, R. Mahapatra, and G. Mangalaraj. Challenges of Migrating to Agile Methodologies. *Communications of the ACM*, 48(5):73–78, May 2005.

[38] M. Paulk. Extreme Programming from a CMM Perspective. *IEEE Software*, 18(6):19–26, November–December 2001.

[39] C. Poole and J. W. Huisman. Using Extreme Programming in a Maintenance Environment. *IEEE Software*, 18(6):42–50, November–December 2001.

[40] B. Rumpe and A. Schroder. Quantitative Survey on Extreme Programming Projects. *Proceedings of the Third International Conference on Extreme Programming and Flexible Processes in Software Engineering (XP2002), Alghero, Italy, May 26-30*, pages 95–100, 2002.

[41] C. Sauer and C. Lau. Trying to Adopt Systems Development Methodologies: A Case Based Exploration of Business Users Interests. *Information Systems Journal*, 7(4):255–275, October 1997.

[42] P. Schuh. Recovery, Redemption, and Extreme Programming. *IEEE Software*, 18(6):33–40, November–December 2001.

[43] Software Engineering Institute. Capability Maturity Model for Software. *[WWW document]. URL http://www.sei.cmu.edu/cmm/*, January 2006.

[44] W. Strigel. Reports from the Field: Using Extreme Programming and Other Experiences. *IEEE Software*, 18(6):17–18, November–December 2001.

[45] T. Chau and F. Maurer and G. Melnik. Knowledge sharing: Agile methods vs. tayloristic methods. *Proceedings of the 12th IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 03)*, pages 302–307, 2003.

[46] C. Team. Chrysler Goes to "Extremes". *Distributed Computing*, pages 24–28, October 1998.

[47] D. Turk, R. France, and B. Rumpe. Assumptions Underlying Agile Software-Development Processes. *Journal of Database Management*, 16(4):62–87, October–December 2005.