

**Extreme Programming Practices and Their Effectiveness in  
Requirements Gathering**

A Final Paper

Presented to the

Faculty of the

School of Engineering

Kennedy-Western University

In Partial Fulfillment

Of the Requirements for the Degree of

Bachelor of Science in

Software Engineering

Craig Thomas

York, PA

© 2005, Craig Thomas

## **ABSTRACT**

### **Extreme Programming Practices and Their Effectiveness in Requirements Gathering**

By  
Craig Thomas  
Kennedy-Western University

#### **Problem:**

During the software requirements phase, a detailed description of the software to be developed, referred to as the software requirements specification (SRS) is provided. The IEEE Software Requirements Standard (IEEE Std 830) was published with the intention of guiding the development of a well-written SRS by defining the fundamental traits of a good requirements specification as being correct, unambiguous, complete, verifiable, consistent, ranked for importance and/or stability, modifiable, traceable and usable (Bennatan, 2000). However, several of the reasons why projects fail point directly to the reliance upon a detailed software requirements specification as being the definitive description of the software to be built. Specifically:

1. There was not clear communication concerning the requirements

2. The requirements as written do not ultimately solve the business problem the customer wanted to solve
3. The requirements have changed before project completion
4. Commitment to a delivery schedule is made before the technical staff fully understands the requirements, which hinders their ability to properly analyze the risks involved (Lindstrom and Jeffries, 2004).

Recent attempts to improve a project's success rate have focused on four separate areas of software development:

- requirements gathering
- design and development
- testing
- project management.

For some projects though, these efforts have done nothing more than to exacerbate the communication and accountability issues they were initially intended to resolve. They have also limited the adaptability of project teams when requirements or implementation details change once development has begun (Lindstrom & Jeffries, 2004).

Extreme Programming (XP) (Beck, 1999) is a rather controversial and much touted methodology that is considered by many software developers and project managers to be a workable solution to software development failures that are related to the issues above. Extreme Programming is viewed as controversial because in XP, software development is incremental, with requirements identified in tandem with planning, coding, testing and design. Additionally, there is a focus in XP on delivering some valuable functionality in 2 to 4 week iterations (McBreen, 2002). Can XP actually overcome these four issues? Does the adoption of Extreme Programming require that all its practices are utilized in an interdependent manner? Can a development team utilize some of XP's practices and still reap the benefits?

**Method:**

This paper is based upon the review of relevant literature that focuses on the thoughts and opinions of other researchers and professionals within the software development and project management communities. Numerous views have been written on the effectiveness of the Extreme Programming (XP) methodology in gathering software

requirements and embracing -- instead of avoiding -- changing requirements.

The cited observations in this paper impart a variety of opinions, both for and against the efficacy of Extreme Programming.

### **Findings:**

This researcher began this study with a preconceived view regarding Extreme Programming practices as they relate to requirements gathering; specifically, that without honoring the interdependency of these practices, they cannot successfully supplant the need for a written requirements specification. Nevertheless, when the XP practices of Planning Game, Small Releases, Metaphor, Testing, and On-site Customer (Beck, 1999) are used collectively, they can be a viable and effective method for eliciting customer's needs and avoiding many of the afore-mentioned pitfalls associated with more classic requirements gathering practices.

However, evidence indicates that the Extreme Programming methodology does have its shortfalls. Rarely can the client provide an On-

Site Customer – if they can provide one at all -- with the necessary domain knowledge and experience needed to move the project along at a sustainable and productive pace. Furthermore, the data suggests that the degree of difficulty inherent in writing User Stories can become insurmountable, fueling a growing reluctance on the part of the customer to be intimately involved in the development process.

Finally, none of the resources spoke to the issue confronting an organization's sales and marketing departments as they relate to actually selling XP. It appears that this overarching difficulty in marketing the idea that the customer will be billed based upon the number of iterations required to successfully complete the project may render Extreme Programming impracticable in a client-vendor relationship.

**Table of Contents**

Chapter 1	Introduction	1
	Problem Statement	1
	Purpose of Study	5
	Importance of the Study	6
	Scope	7
	Rationale	9
	Definition of Terms	10
Chapter 2	Review of Related Literature	15
Chapter 3	Methodology	42
	Approach	42
	Data Gathering Method	43
	Database of Study	43
	Originality and Limitations of Data	44
	Summary	45
Chapter 4	Data Analysis	46
	Overview of the Analysis	46
	Limitations of the Data	48
	Reliability of the Data	49
	Significant Findings	49
Chapter 5	Summary, Recommendations, and Conclusions	61

Summary	61
Conclusions	65
Recommendations	71
Bibliography	80

## **CHAPTER 1**

### **Extreme Programming Practices and Their Effectiveness in Requirements Gathering**

#### **Introduction**

##### **Problem Statement:**

There is ongoing debate in the software development community regarding the communication gap that exists between those who want new software to be developed and those who eventually develop it. No matter how precise the Systems or Business Analyst is in gathering and documenting the customer's requirements through interviews, brainstorming, role-playing and observation, the customer will request a change in the system being built. And, there is the opinion that the cost of change grows exponentially as a project progresses through its lifecycle (Boehm, 1981).

The inaccuracies of written language can harm a project's success because words, especially when in print, can be easily misinterpreted by the technical team, customers, users, analysts, domain experts, and so

on. In other words, successfully eliciting software requirements relies heavily upon precise communication (Cohn, 2004). The challenge lies in bridging the communication gap between the developers -- who want the software requirements expressed in an ordered fashion -- and the customers -- who want the requirements to be easily understood to better facilitate their verification.

If the technical team misunderstands the requirements, then estimating how long the project will take to complete becomes a difficulty. It is much simpler to estimate a small chunk of work than it is to estimate a year's worth of development (McBreen, 2002). If there is a misunderstanding on the part of the customer as to what is to be built then, when the software – or a portion thereof – is delivered, the customer comes to a more clarified realization of what it was they really wanted in the first place.

Recently, this researcher had the opportunity to develop a web-based application, called WebExec, for the Sales & Marketing Department of a Pennsylvania-based snack food manufacturer that would enable all sales brokers to manage their customer contact information, note details

of meetings and conversations with clients, enter information about special promotions, and add or update corporate, billing and shipping addresses.

One of the requirements specified was the need to retrieve specific salesman and customer information from company's legacy system -- a J.D. Edwards system supported by IBM AS400 database servers -- and populate a new Microsoft SQL2000 database designed specifically for the new web-based solution. From this requirement, the development team wrote the SQL transactions that queried the AS400 and populated the Salesmen, Customers, CustomerTypes, Addresses and AddressTypes tables in the new database. They then worked on designing a web interface that would deliver the data through the web-enabled presentation layer.

After four weeks of development, the client saw the data being presented and he met with the project manager requesting information about the customers and salesmen very different from what was originally requested. The project manager made the appropriate changes to the requirements specification and relayed the alterations to the development team. Not only were the developers back to the drawing table in

developing appropriate SQL queries, they also had to develop new class files and redesign how the web pages displayed the information.

Two weeks later, the same scenario repeated itself. However, on this occasion, the client, project manager and developers spent a day in a conference room querying and re-querying the AS400 database, displaying the results on an overhead screen. When the client was fully satisfied that the query results were exactly what he desired, the meeting was concluded and the requirements were amended to reflect the most recent changes. The developers wrote new code, altered class files, adjusted the SQL queries and performed a second redesign of the user interface, delivering the finished application twelve weeks after the project was initiated.

In the final team meeting it was determined that, if the client had been more intimately involved in the day to day development process and the development team had delivered smaller chunks of functionality for the client to test, much of what occurred during the development of WebExec could have been avoided. In other words, if some of the core practices of Extreme Programming had been integrated into this project, the team

believed that the advantages would have been immediately evident, resulting in a shorter time to implementation and lower man-hour cost.

**Purpose of Study:**

It is the purpose of this study is to evaluate the effectiveness of replacing a written requirements specification with select core practices of Extreme Programming (XP) (Beck, 1999), an agile software methodology that actually embraces requirements drift as a naturally occurring event during the development process. XP puts forth the view that while it can be more costly to modify code already written than to alter the software requirements specification, with contemporary programming languages and development techniques the expense is *not* increased exponentially. Beck suggests that instead of expending the extra effort nailing down all of the customer's requirements up front, accept that changes will be dealt with later in the development process (Beck, 1999). This is based upon XP's argument that the cost of modification will be minimal compared to the costs associated with analyzing or development based upon requirements that are incomplete or subject to change.

Specifically, this paper will explore three aspects of this debate:

1. Whether the practices of Extreme Programming are so thoroughly interdependent that they prohibit the ability to “pick and choose”?
2. Do the XP practices of Planning Game, Small Releases, Metaphor, Testing and On-site Customer successfully overcome the problems associated with a detailed requirements specification?
3. If a customer requires a written software requirements specification, can it be written during or after development?

**Importance of the Study:**

The avoidance of a project’s failure or abandonment, especially in this highly competitive and budget-conscious era, is of foremost importance to an organization determined to taking advantage of advances in technology to improve their business’s productivity and profitability.

If, by adopting some of the core practices of XP, software development teams and organizations can deliver a software product on time and within budget that solves the business problem it was intended to solve -- all the while fluidly working with the drift in software requirements, then it would certainly be in the best interest of both developers and

customers to give strong consideration to integrating these Extreme Programming practices.

Additionally, it is imperative that software development teams understand the potential problems associated with adopting the XP methodology. Extreme Programming is considered by many to be a process comprised of interdependent practices; you can't adopt some of the practices, forgoing the remainder, and expect to experience the success teams who have adopted the complete XP methodology have.

Finally, it is crucial for developers and project managers to gain an understanding of the overall workability of the Extreme Programming methodology. Is it even worth attempting? Is XP a truly valuable practice, or is it simply a fad that has been embraced by dissatisfied programmers who can't help but want to buck the norm?

**Scope:**

For the purpose of this paper and its discussion of the viability of replacing a written requirements document with aspects of the XP

methodology, the focus will be on five core practices of Extreme Programming. They are:

1. The Planning Game –

a. business people set forth

- i. what needs to be developed to solve the business problem (Scope)
- ii. what needs to be developed in what order (Priority)
- iii. how much needs to be developed before the business realizes a benefit (Composition of Releases)
- iv. when would it be best to have the software (or components of the software) delivered (Dates of Releases).

b. The development team determines

- i. how long it will take for them to put a feature into operation (Estimates)
- ii. what risks there are in choosing a programming language, database, user interface, etc. (Consequences)
- iii. the organization of the development team (Process)
- iv. what, within a short release, will be done first (Detailed Scheduling).

2. Small Releases – getting something of value to the customer in two to three weeks iterations.
3. Metaphor – a story that can act as the overall driving viewpoint for the system to be built.
4. Testing – ensuring that each of the Small Releases (see #2 above) work properly by utilizing automated tests that the On-site Customer (see #5) runs.
5. On-site Customer – a real customer or domain expert who makes himself available to the team by remaining on-site during development

**Rationale:**

The core reasoning behind the desire to explore the usefulness of Extreme Programming practices as they relate to software requirements specifications stems from this researcher's experience working on projects both as a developer and business analyst. Repeatedly this researcher has experienced projects that drifted off the detailed requirements document, causing what this author considers avoidable delays in project completion.

Though far from exhaustive, a study of the Extreme Programming methodology has instilled a strong belief that core practices of XP can eliminate the adverse effects of requirements creep and miscommunication. However, some of the core practices of XP (On-Site Customer, for example) appear to be difficult to properly and effectively implement.

And, it should be noted that this researcher has yet to see firsthand how effective these Extreme Programming practices would actually be in improving a project's ultimate success.

This study will not only act as an academic requirement, it will also aid in improving and streamlining the development processes this researcher utilizes on a day-to-day basis.

### **Definition of Terms**

**Collective Ownership** - An Extreme Programming concept that states that the customers and developers own an equal share of the project. (Beck, 2001)

**Metaphor** - Extreme Programming teams develop a common vision of how the program works, which we call the "metaphor". At its best, the metaphor is a simple evocative description of how the program works, such as "this program works like a hive of bees, going out for pollen and bringing it back to the hive" as a description for an agent-based information retrieval system.

(Source: <http://www.xprogramming.com/xpmag/whatisxp.htm#metaphor>)

**Pair Programming** - The Extreme Programming practice of having two programmers work on one User Story. One programmer codes (drives) while the other gives input and feedback (steers). (Source: Beck, 2001)

**Project Management** - the discipline of defining and achieving targets while optimizing the use of resources (time, money, people, space, etc). Thus, it could be classified into several models: time, cost, scope, and intangibles. (Source: [http://en.wikipedia.org/wiki/Project\\_management](http://en.wikipedia.org/wiki/Project_management))

**Refactoring** - In software engineering, the term *refactoring* is often used to describe modifying source code without changing its external behavior, and is sometimes informally referred to as "cleaning it up". (Source: <http://en.wikipedia.org/wiki/Refactoring>)

**Requirement** - In software engineering, a requirement is a description of *what* a system should do. Systems may have from dozens to thousands of requirements. (Source: <http://en.wikipedia.org/wiki/Requirements>)

**Requirements Creep (or Drift)** - (also known as functionality-creep, feature-creep, mission-creep and scope-creep) is a problem in project management where the initial objectives of the project are jeopardized by a gradual increase in overall objectives as the project progresses.

(Source:

[http://en.wikipedia.org/wiki/Creep\\_%28project\\_management%29](http://en.wikipedia.org/wiki/Creep_%28project_management%29))

**Requirements Traceability Matrix** - A document that maps the parent-child relationships of requirements.

(Source: [www.pmforum.org/library/glossary/PMG\\_R02.htm](http://www.pmforum.org/library/glossary/PMG_R02.htm))

**Requirements tracing** - The linking of a *requirement* to other requirements and to other associated project elements. (Source: [www.pmforum.org/library/glossary/PMG\\_R02.htm](http://www.pmforum.org/library/glossary/PMG_R02.htm))

**Small Releases** - XP teams practice small releases in two important ways:

First, the team releases running, tested software, delivering business value chosen by the Customer, every iteration. The Customer can use this software for any purpose, whether evaluation or even release to end users (highly recommended). The most important aspect is that the software is visible, and given to the customer, at the end of every iteration. This keeps everything open and tangible.

Second, XP teams release to their end users frequently as well. XP Web projects release as often as daily, in house projects monthly or more frequently. Even shrink-wrapped products are shipped as often as quarterly.

(Source: <http://www.xprogramming.com/xpmag/whatisxp.htm#small>)

**Traceability** - The management of the parent/child relationships of all system requirements.

(Source: [www.pmforum.org/library/glossary/PMG\\_R02.htm](http://www.pmforum.org/library/glossary/PMG_R02.htm))

**Unit Test** - In computer programming, a **unit test** is a method of testing the correctness of a particular module of source code.

(Source: [http://en.wikipedia.org/wiki/Unit\\_testing](http://en.wikipedia.org/wiki/Unit_testing))

**User Story** - In the eXtreme Programming (XP) methodology, a User Story is a way for customers to communicate, to the programmers, functionality that they would like to see in a product. A User Story usually consists of 1 piece of functionality, that is, one "something" that the user will be able to "do" with the software.

(Source: [http://en.wikipedia.org/wiki/User\\_story](http://en.wikipedia.org/wiki/User_story))

## **CHAPTER 2**

### **Extreme Programming Practices and Their Effectiveness in Requirements Gathering**

#### **Review of Related Literature**

##### **Introduction**

Since 1999, there have been numerous white papers and articles published addressing the need for a new methodology of software development that successfully overcomes the increased proclivity towards failure that projects based upon detailed requirements specifications possess. A variety of these resources were researched in the compilation of relevant literature. Sources ranged from published books and articles to on-line commentary. The referenced research material detailed below is separated by book or article title.

Additionally, viewpoints expressed during and after a Process Meeting at Cimbrian, Inc. of Lancaster, Pennsylvania where the implementation of Extreme Programming was discussed are summarized at the end of this chapter.

## **Extreme Programming Explained: Embrace Change**

Based upon practices developed during the latter part of the 1990's while working at Daimler-Chrysler, Kent Beck's Extreme Programming Explained: Embrace Change (Beck, 1999) has become the de-facto Extreme Programming (XP) "bible." It is in this book that the core practices of XP are described in detail in addition to Beck's compelling arguments as to why these practices should be adopted.

In 1996, the automaker Daimler-Chrysler hired Kent Beck, Martin Fowler, and Ron Jeffries to rescue a struggling development project known as Chrysler Comprehensive Compensation (C3). The intention of the C3 project was to revamp Chrysler's payroll system and had come upon some serious setbacks that threatened the success of the project.

Dismissing half of the C3 project team, Mr. Beck and his new, streamlined group began to experiment with practices which would eventually become known as Extreme Programming. Utilizing these new practices, the development team was able to produce in eight months a payroll system that Chrysler believed contained the desired features and functionality. Though eventually scrapped in February of 2000 because

the system was only handling 1/3 of Chrysler's payroll, the software community at large had already become aware via the Internet of this new and exciting methodology dubbed Extreme Programming. And the rest, as they say, is history.

As explained in Kent Beck's Extreme Programming Explained: Embrace Change, risk is the fundamental problem in software development. Each time a software project begins development, risks such as slips in schedule, project cancellation, defects, failure to solve the business problem, and the development of interesting – but worthless – features, expose the project to an increased potential for failure. (Beck, 1999)

- **Risk of Schedule Slips:** to combat the possibility that a software project will not be complete within the desired time, the XP methodology utilizes the practice of Small Releases (short release cycles; on average, 6 weeks) to reign in the scope of any slip. In addition, each release is comprised of smaller one- to four-week iterations. These small iterations allow the customer to see continual progress and offer more finely-focused feedback. XP's further breaking down each of these iterations into one- to three-

day tasks allows the development team to solve business problems within each and every iteration. Lastly, the most important features are implemented first to ensure that only the least valued features are remaining if there is a schedule slip. (Beck, 1999)

- **Risk of Project Cancellation:** by focusing on high-value short releases, XP minimizes the chance that the project will be cancelled as the customer is seeing business value often and repeatedly. (Beck, 1999)
- **Risk of Defects:** each Small Release is tested thoroughly by the development team and by a real customer who sits with the development team full-time (see below) prior to acceptance. This practice of testing each Small Release also adds the additional benefit of eliminating the lengthy testing cycle at the end of development. (Beck, 1999)
- **Risk of Failure to Solve the Business Problem:** a core element of the XP methodology is the On-Site Customer. It is XP's goal to have a customer as an essential part of the development team. Specifically, someone who will actually *use* the software would act as an On-Site Customer. The XP intention is to have this individual sit with the development team, thus the designation "On-Site." As the project is being developed and tested, the On-Site Customer

would refine the project's specifications, and rearrange the order in which functionality is to be built. Additionally, the On-Site Customer writes the User Stories during what is called in the Planning Game. These User Stories are nothing more than written stories describing the things the new system needs to do. Usually written on index cards, the stories are given a name and contain a short paragraph which describes each story's purpose. These User Stories – and the technical estimations associated with them – act as an ongoing and fluid guide for the developers. Beck further states that these stories are to be viewed as “reminders to have a conversation.” (Beck, 1999)

- **Risk of Developing Interesting – but Useless – Features:** because the XP methodology dictates that only the tasks considered high priority are dealt with, features that would be nice to have, but offer no intrinsic value to the business, are simply not developed. (Beck, 1999)

### **User Stories Applied: For Agile Software Development**

Mike Cohn's views expressed in his book User Stories Applied: For Agile Software Development (Cohn, 2004) address how there is an added

potential of upfront requirements gathering and documentation actually harming the success of a project. His view is that a requirements specification should only be written if it will actually help achieve the goal of software delivery. Cohn puts forth the opinion that when the requirements specification itself becomes the goal there is an increased possibility for failure. He also states that the “inaccuracies of written language” further decrease the requirements document’s value to the project. (Cohn, 2004)

Cohn believes that User Stories offer numerous advantages over requirements or use cases. For example:

- User stories focus on verbal communication instead of written
- User Stories can be easily comprehended by the customers and developers
- User Stories are easy to plan with
- User Stories work with the practice of Short Releases
- User Stories support the practice of putting off detail until the XP team understands what is really needed. (Cohn, 2004)

The practice of having an On-Site Customer working alongside the development team is also discussed. Cohn states that the luxury of having

one person who writes and prioritizes the User Stories, answers all the developer's questions and uses the software when it's complete is often impossible to achieve. Instead, he suggests that the role of On-Site Customer be a group of individuals comprised of users, testers, managers and designers. Once this group is identified and assembled, they all should take an active role in writing the stories. (Cohn, 2004)

Mr. Cohn offers two reasons why the customers, as opposed to the developers, write the stories. First, the stories need to be written in a language the customer will understand so that the stories can be prioritized and considered for inclusion in a specific Small Release. Second, as the ultimate users of the system, the customer is naturally in a better position to describe how the system should eventually work. (Cohn, 2004)

Cohn also supports the XP practice of testing each story upon its development. The On-Site Customer runs these acceptance tests, which in most cases should be automated, to verify that the application is so far acceptable. Additionally, all previous tests must be run to ensure compatibility with the new components. In other words, the process of

testing will become more prolonged with each new iteration. For this reason, automated tests become especially vital. (Cohn, 2004)

Finally, certain drawbacks to using User Stories are described. One issue is that the relationship between User Stories on a large project might be difficult to comprehend. Cohn suggests using roles and keeping the stories at a moderate to high level of ambiguity until it is time to develop them. There's also the problem of having a customer who mandates additional documentation for requirements traceability. In this case, documentation at the beginning of each iteration specifying the traceability – for example, from specification to test – is recommended. (Cohn, 2004)

### **Questioning Extreme Programming**

This book by Pete McBreen undertakes two of the issues addressed in this paper, User Stories and On-Site Customer.

**User Stories:** McBreen states that “estimating is much easier when there is only a small amount of work to estimate.” He believes that estimating two to three months of work is simpler than estimating the amount of work a development team can do in a year's time. And, in his

opinion, adding the practice of User Stories helps because it compels the team to estimate small, one- to two-week bits of functionality. (McBreen, 2002)

He also agrees with Kent Beck's (Beck, 1999) assumption that instead of attempting to control requirements creep, it makes more sense to have a process where developers can rapidly respond to change. However, Pete McBreen considers requirements traceability to also be a valid method for handling change – especially on larger projects. He believes that by having a process for recognizing changes as they occur – and documenting the approved changes – a development team can successfully work with any requirements drift or creep. (McBreen, 2002)

**On-Site Customer:** McBreen says that many software development approaches could gain additional benefit from an On-Site Customer. Stating that documentation can be a useful repository for software requirements, the development team can get a much better idea of what the customer wants if they can actually talk to them. He believes that many projects experience difficulty due to inadequate communication – suggesting that the development team be more aggressive in requiring on-going customer contact. (McBreen, 2002)

Though his opinion is that an On-Site Customer is a good practice, he questions whether they will have ample domain knowledge to explain what is needed and act as an effective intermediary between the developers and the rest of the organization. If the On-Site customer lacks the essential domain knowledge, they have to go back and consult with their colleagues to get the right information; and that is likely to impede progress. Further, it is Pete McBreen's opinion that an On-Site Customer who possesses deep domain knowledge is a rare commodity within an organization. (McBreen, 2002)

Finally, McBreen questions whether the necessary time commitment can be obtained from the project's sponsors. It's quite possible that an individual having broad enough domain knowledge might be considered more valuable elsewhere – not sitting with a team of developers. (McBreen, 2002)

## **Extreme Programming and Agile Software Development Methodologies**

In an article published by Information Systems Management magazine for their Summer 2004 issue, Lowell Lindstrom and Ron Jeffries

discuss several lightweight development methodologies, with a major emphasis on Extreme Programming. In this article, the authors describe some of the reasons behind project failure – many of which have to do with poor communication and faulty requirements. They proceed to say that demand for new software continues despite these ongoing issues with delivery, citing a still existing shortage of software professionals. (Lindstrom and Jeffries, 2004)

Lindstrom and Jeffries state that efforts to lessen the failure rates of projects center on four separate areas: requirements, design and development, testing, and project management. According to them, these efforts did bring about greater success with some of the adopters, but many projects continued to fail using the same methods. And, there were projects that succeeded without written requirements or project plans. (Lindstrom and Jeffries, 2004)

The authors state that a number of papers being written touting these new, lighter approaches to software development catalyzed the formation of a community that went on to codify practices that did away with many of the activities required by more defined development

methodologies. One of these practices came to be called Extreme Programming (XP). (Lindstrom and Jeffries, 2004)

Lindstrom and Jeffries define Extreme Programming as “a discipline of software development based on [the] values of simplicity, communication, feedback, and courage” and it is the presence of these values that should be used to measure whether a project team is a proper match for the XP methodology. The authors also indicate that XP is the only agile “methodology that is explicit in its values and practices.” (Lindstrom and Jeffries, 2004)

In exploring the XP practice of On-Site Customer, the writers feel that – realistically – this role would be comprised of a team of people that “communicates with one voice”, referring to this role as the “Customer Team.” They explain that, although the idea of having a customer present is desirable, it can seem to be an impossible goal, especially if the most appropriate person to work with the team does not live nearby, or is always away on business. Additionally, the developers – having worked for so long from requirements documents – may feel uncomfortable interfacing with a customer or Customer Team. (Lindstrom and Jeffries, 2004)

Another aspect of XP investigated in this article is the practice of Planning Game, which is described as having as its focus two key issues within software development; 1) calculating what will be completed by the due date, and 2) deciding what to build next. (Lindstrom and Jeffries, 2004)

As to calculating what will be completed in the next iteration, the authors explain that the aspect of the Planning Game called Release Planning satisfies that requirement. They explain that the On-Site Customer describes the features he wants to the programmers, then the programmers estimate the difficulty. With estimates of time and cost in hand, the customer lays out the project plan. (Lindstrom and Jeffries, 2004)

The authors go on to explain that XP's Iteration Planning is a core practice where the customer gives the development team direction every two to three weeks, thus fulfilling to need to determine what will be built in the next iteration. (Lindstrom and Jeffries, 2004)

### **Extreme Programming Perspectives**

Published by Pearson Education in 2002, this book is a compilation of articles written by various authors within the XP community. To facilitate review, each article pertinent to the subject of this paper will be allocated its own sub-section, separated by article title in italics.

### *Chapter 18 - Refactoring or Up-Front Design*

This article, written by Pascal Van Cauwenberghe, addresses whether up-front design – design supported by a written requirements document and modeling – or Extreme Programming’s approach of iterative development is the best method for system implementation. The author argues that all “experienced software engineers use a combination of both methods.” He believes that the waterfall method – where a system is analyzed, designed, coded, integrated, tested, and released without necessity to reiterate earlier steps – is rarely used, even by developers who allege to employ it. (Marchesi, et al, 2002)

Van Cauwenberghe’s opinion is that all software development methods utilize some variety of incrementally driven development process. Hence, as it pertains to XP’s practice of Planning Game, User Stories and

Small Releases, there's nothing truly "extreme" about Extreme Programming. (Marchesi, et al, 2002)

However, he feels that it is important to evaluate whether to utilize incremental design on a case by case basis by choosing the method that implies the least risk, which – in some case – is not a strict adherence to XP's iterative approach. (Marchesi, et al, 2002)

### *Chapter 31 - Lessons Learned from an XP Project*

Natj Kini and Steve Collins document their experience while working with a software company that used the XP methodology in this 2003 article. There are a number of issues that they bring forward that relate to the subject of this final paper.

Because their client was a start-up company, they were found to be open to utilizing XP. The company was at a stage where dedicating someone the act as an On-Site Customer was possible and convenient. They were also well aware of the inadequacies associated with more conservative software development methodologies. (Marchesi, et al, 2002)

One interesting approach the development team took was to bill the customer by the iteration instead of by the hour to avoid any disagreements over the number of hours the developers worked on the system. The team felt that this would help demonstrate to the customer that they were sharing the risk together, and that the team was truly invested in the project. The initial contract was augmented every two weeks with a new list of User Stories for the following iteration (Marchesi, et al, 2002)

The team also discovered that the customer desired a much more detailed status report than what was originally planned. A compromise was agreed upon whereby the team would e-mail a daily status report to the stakeholders. (Marchesi, et al, 2002)

Desiring to impress the customer, the team initially committed to more than they could actually complete in an iteration. Subsequently, they fell short on creating the acceptance tests and had a more difficult time adhering to the XP practices when they felt “under the gun”. (Marchesi, et al, 2002)

Another outcome of the team's lax attention to testing was that -- for each subsequent iteration -- more and more time was spent fixing defects from earlier iterations. Though the customer saw these issues as "bugs", the team saw them as new User Stories to be built, increasing the amount of work to be done and billed for. This became an even greater point of contention because the customer felt that they were "owed bug fixes as part of an earlier iteration." (Marchesi, et al, 2002)

The team had to also reconsider the amount of documentation necessary. The customer's developers were going to be responsible for maintenance and enhancements, so they desired more detailed documentation than what would usually be provided by an XP project. (Marchesi, et al, 2002)

### *Chapter 32 - Challenges for Analysts on a Large XP Project*

Gregory Schalliol's contribution to this collection of articles discusses the issues encountered by a group of system's analysts on a large software development project that converted to the XP methodology. (Marchesi, et al, 2002)

One such issue was the lack of a “readily available, holistic picture” of what was to be built, and how each of the system’s components interacted with each other. Because this was such an expansive project, the system seemed to many of the developers as being a collection of unassociated components. The lack of this “holistic” picture was especially a challenge for those who did not have direct experience with the business (leasing) for which the system was to be built. Consequently, the author felt that there should be some form of overarching reminder of the interdependence and relationship between the various functionalities built from separate User Stories. (Marchesi, et al, 2002)

The team also experienced issues related to separating the complete system into User Stories. A seemingly interdependent component from one iteration became, a few iterations later, so completely intertwined with newer functionality that it created a necessity for the team to rewrite User Stories and developing new, more inclusive tests to validate the new dependencies. This unearthed a “tension between two goals of XP.” Iterative development is supposed to deliver to the customer usable functionality at “frequent intervals”, allowing the customer at any given time to terminate the project and walk away with a system that has some business value. However, the development team

found that it was nearly impossible to deliver functionality that had business value when they developed iteratively. (Marchesi, et al, 2002)

Another issue was the difficulty experienced in finding an On-Site Customer who could devote her time to the team on a full-time basis. To overcome this obstacle, the role of On-Site Customer was comprised of multiple individuals. However, difficulties arose when individual members had “peculiar requirements that were not always compatible with one another.” (Marchesi, et al, 2002)

The final issue the team experienced was the difficulty they had with getting the customer team to develop the functional tests that would verify the completion of requested functionality. The authors feel that part of the problem was that the customers were hesitant to delve into the complexity of the application. Only after much coaching did the customer feel comfortable with writing functional tests. (Marchesi, et al, 2002)

In his 2003 contribution to *Extreme Programming Perspectives* (Marchesi, et al, 2002), Amr Elssanadisy of ThoughtWorks, Inc. relates his experience working on a large development project that utilized the XP methodology. On this project were thirty-five developers, eight business analysts, and seven QA testers. The analysts were relied upon in place of an On-Site Customer, working closely with the actual customers of the project. (Marchesi, et al, 2002)

During development, it was found that there was a necessity for a group of people to act as the On-Site Customer in order to generate enough work for the thirty-five developers. However, the fulfillment of this need can be hindered by a lack of sufficient domain knowledge. (Marchesi, et al, 2002)

Additionally, the XP practice of Metaphor was found to be too complex for such a large project, and some User Stories took up more than one iteration due to the amount of code. (Marchesi, et al, 2002)

Ann Griffin's 2003 article discusses the "physical and organizational changes that occurred" in her company "as a result of XP implementation". Prior to this implementation, the company had a history of failure in on-time delivery of software that satisfied their customer's requirements, revealing the necessity for a more standardized development process. The Extreme Programming methodology was chosen as the process. (Marchesi, et al, 2002)

Initially, estimating releases took longer than expected, due to the developer's lack of understanding in regards to the "big picture." There was also an issue with the User Stories lacking enough detail to work from, ultimately failing to provide a solution to the business problem they were intended to address. The transition from developers making many business decisions to the customer making them was also found difficult. (Marchesi, et al, 2002)

Due to the specialization of the development team, work was distributed unevenly. While some of the team was bogged down in too much work, others were left with little to do. (Marchesi, et al, 2002)

Some of the team members were reluctant to utilize XP as a methodology and were eventually selected to work on the legacy portion of the application. Though it appeared that the non-XP team made more rapid progress early on, it was found that the code they produced required more testing and fixing than the code produced by the XP practitioners. (Marchesi, et al, 2002)

Additionally, the company found it difficult to allow the development team to set their own time estimates, instead of management basing estimates upon the assumed capacity of the team. (Marchesi, et al, 2002)

### *Chapter 35 - Learning by Doing: Why XP Doesn't Sell*

In 2000, the authors, Kay Johansen, Ron Stauffer, and Dan Turner decided to try Extreme Programming on a project where they were going to be respectively, team lead, senior programmer, and product manager. The project was to accomplish a major upgrade to “a small to medium-sized enterprise software system.” (Marchesi, et al, 2002)

The product manager – Dan Turner – was impressed with how simple and flexible User Stories were in planning and steering the project.

He also found it to inherently encourage open dialog between himself – acting in the role of On-Site Customer – and the development team. Many on the team found the process of moving from story to development alleviated much of the stress associated with more conservative methodologies. The team was able to accurately estimate how long each User Story would require to complete, and additional features were simpler to add. (Marchesi, et al, 2002)

However, despite the opinion on the part of the development team and product manager that the project was a success, management felt that the team had delivered less than satisfactory performance. They considered the team's estimates to indicate a "lack of commitment" to not working faster. When a subsequent round of layoffs occurred, the XP team was let go. (Marchesi, et al, 2002)

### *Chapter 36 – Qualitative Studies of XP in a Medium-Sized Business*

Robert Gittins and Sian Hope of the University of Wales, and Ifor Williams of Secure Trading, Inc. conducted a qualitative study of Secure Trading's implementation of the Extreme Programming methodology. A team of nine developers were committed to a progressive implementation

so as not to disrupt the normal business process. Their paper reveals a number of issues associated with the adoption of XP. (Marchesi, et al, 2002)

Because Secure Trading serviced a large number of clients, the role of On-Site Customer was impractical. So, they developed what they called a customer proxy. The proxy, armed with an understanding of the customer's needs, acted on their behalf with their best interests in mind. Consensus between interested parties was facilitated by the XP developer manager. The manager felt that having "a representative from customer services" proved to be "hugely beneficial," and was able to provide "immediate feedback of the system's successes and failures on a day-to-day basis." (Marchesi, et al, 2002)

In the area of testing, 71% of Secure Trading's developers considered XP's unit testing practices to be very poor. They felt that with a complex legacy system, testing was difficult. Their opinion was that it would be easier to implement unit testing on a system started from scratch. (Marchesi, et al, 2002)

Finally, the practice communicating a clear mental image of what a system is to be like (Metaphor in XP), so that all involved would have a grasp of the project's essence, was found to be so difficult that the practice was ultimately abandoned. (Marchesi, et al, 2002)

The authors concluded that XP practices are, in fact, interrelated with and dependent upon each other. They also deduced that the adoption of a test-first strategy was difficult, especially with developers new to XP. (Marchesi, et al, 2002)

### **Extreme Programming Process Meeting, Cimbrian, Inc.**

On June 17, 2005, this researcher was invited to sit in on a Process Meeting at Cimbrian, Inc. of Lancaster, Pennsylvania. Cimbrian was 9 months into implementing the Extreme Programming methodology into their development process and met on a monthly basis to discuss how the implementation was coming along and what issues they were experiencing as their shop migrated to XP. (Barrett, 2005)

Cimbrian does 90% of their development in-house, physically separated from their client and rely upon a Business Analyst (BA) to act as

the On-Site Customer. The BAs reported that they were experiencing ongoing issues with receiving timely responses from the customer when a question arose which was outside the analyst's domain knowledge. These delays resulted in work stoppages or slowdowns until the customer responded. Kirk Barrett, Cimbrian's CEO, expressed that they have yet to find a client willing to provide a domain expert to be the On-Site Customer. He suggested that the clients they have are still "old school" and reluctant to become intimately involved with a process that appears "out of the ordinary." (Barrett, 2005)

As with most development shops, Cimbrian's business is generated by word-of-mouth and marketing. A potential client is contacted or referred, then wooed by Cimbrian's sales department, ultimately resulting in the gathering of preliminary specifications to facilitate a quotation. Mr. Barrett stated that they've been unable to find a way to sell their customers on agreeing to a per-iteration pricing structure that aligns with Extreme Programming's process. Subsequently, a best-effort is made to predict how many man-hours a project is going to take based upon the preliminary specifications. As this practice is in opposition to the tenets of XP, it by nature fails to avoid the unavoidable time- and cost-overruns and

– much to Cimbrian’s chagrin – has resulted in some loss of profit.

(Barrett, 2005)

## **CHAPTER 3**

### **Extreme Programming Practices and Their Effectiveness in Requirements Gathering**

#### **Methodology**

##### **Approach**

The approach utilized in the research of this final paper was the Applied Research methodology comprised of reading and evaluating literature written by proponents and exponents of the Extreme Programming methodology. The focus was to collect information from sources that was pertinent to the subject of this paper and, where possible, unbiased. As this researcher has no personal experience using the XP methodology, it was deemed imperative that the literature reviewed emanated from experienced sources.

In addition to published books, magazine articles, and papers found on the Internet, this researcher was invited by Kirk Barrett, CEO of Cimbrian (<http://www.cimbrian.com>), a Lancaster, Pennsylvania-based

development shop, to sit in on a Process Meeting where the progress of adopting the Extreme Programming methodology was discussed in detail.

### **Data Gathering Method**

The primary method used to gather information relevant to this paper's subject was to search the Internet for recommendations for literature that examined how Extreme Programming worked, why there was a need for it, how it was implemented, and what successes and failures were experienced.

### **Database of Study**

In researching this subject, numerous books written on Extreme Programming were collected, as well as the afore-mentioned article. Additional research was performed via the Internet by "Googling" (entering search criteria at [www.google.com](http://www.google.com)) "extreme programming", "xp", "requirements creep", "requirements drift", "agile methodologies", etc.

A wealth of information was discovered using this method. However, most Internet-based resources referred back to the literature this researcher ultimately used as research material. Several articles and papers were also discovered through Internet searches. It was these resources where this researcher was able to find opinions in disagreement with the practices of Extreme Programming.

### **Originality and Limitations of Data**

The most glaring limitation of the data collected – as far as this researcher is concerned – is the lack of published disagreement with the practice of Extreme Programming. Tome after tome regaling XP's strengths can be found, but it is quite difficult to find books written that do not praise the effectiveness of XP. It was incumbent upon this researcher to "read between the lines" while reviewing the literature used in this study to facilitate a balanced conclusion. In addition, it was necessary to disregard the multitude of opinions found during the numerous Internet searches this researcher performed.

## Summary

To summarize, it is important to note once again that this researcher – though presently employed as a programmer – has no personal experience using the Extreme Programming methodologies. It was an effort to compile literature that presented a balanced opinion in regards to the pros and cons of XP, not only to satisfy the requirements of this paper, but to also satisfy this researcher's desire to discover a more effective way to work with constantly changing requirements. In short, what literature found available was simply disappointing.

## **CHAPTER 4**

### **Extreme Programming Practices and Their Effectiveness in Requirements Gathering**

#### **Data Analysis**

##### **Overview of the Analysis**

The increasing dissatisfaction among software developers and their managers -- fueled by inaccurate software specifications, cost overruns, time overruns, unpredictability, and unavoidable requirements creep -- has acted as a catalyst behind the growing popularity of agile methodologies of which Extreme Programming (XP) is a member. Coupled with this rising interest in alternate approaches to the software development lifecycle is the recognition by many developers that there exists an opportunity to capitalize on the continued emergence of XP into the software development mainstream. This has, in turn, resulted in numerous books written on the advantages and utilization of XP.

Locating material that supports – sometimes reverentially -- the practices of XP proved to be an all-too-easy task. Finding material that criticizes Extreme Programming was, in comparison, challenging.

A concerted effort to review literature and data that both supported and contradicted the XP methodology compelled this researcher to not only comb library and bookstore shelves, but to search the Internet for criticisms of this somewhat idolized practice. Though such criticism does exist, it is overshadowed by the amount of material written by XP's proponents. Even Pete McBreen's Questioning Extreme Programming (McBreen, 2002) was light-handed in its objections to some of what XP extols – almost as if Mr. McBreen was already an XP convert, but wanted to ensure literary originality by mildly disapproving.

However, as the analysis of information progressed, it became clear that there did exist a strong dissenting voice, for the most part relegated to articles and papers published on the Internet. Additionally, the opportunity of attending the Process Meeting at Cimbrian in Lancaster, Pennsylvania proved to be a valuable and informative experience in that Cimbrian's CEO, Kirk Barrett, was able to provide first-hand insight into some of the business challenges Extreme Programming presented.

## **Limitations of the Data**

Lacking any first-hand experience with the utilization of XP, this researcher was reliant upon published materials and information derived from attending the Process Meeting at Cimbrian to formulate the content and ultimate conclusion of this final paper.

The data collected was limited by the desire to accumulate enough material both pro and con to offer a balanced assessment of the Extreme Programming methodology. In pursuit of this balanced review, many resources that openly supported XP were ignored as they were found to be somewhat redundant in their praise of the Extreme Programming methodology. Instead, this researcher worked vigorously to locate and review articles, papers, and books that were more critical of XP and its effectiveness in overcoming drifts in a customer's requirements.

Nonetheless, this researcher is confident that the data collected was sufficient to achieve the goal of this paper, which was to evaluate the usefulness of Extreme Programming in overcoming the perceived shortfalls in requirements gathering.

## **Reliability of Data**

The authors of the books and papers cited as references throughout this final paper all have backgrounds in software engineering and/or project management. Furthermore, they all possess experience working with the Extreme Programming methodology. Their opinions come from collective years involved in development projects where systems were built from a detailed software specification and with applications developed utilizing XP.

Additionally, the information gathered from attending a Process Meeting and interviewing Kirk Barrett, CEO at Cimbrian, Inc. was invaluable due to Cimbrian's first-hand knowledge of working with and without the aid of the Extreme Programming methodology.

## **Significant Findings**

### *Planning Game:*

In Extreme Programming and Agile Software Development Methodologies, authors Lowell Lindstrom and Ron Jeffries describe the

XP practice of Planning Game to be focused on two important aspects of software development; estimating what will be accomplished by a particular due date, and what features will be developed in the next release. (Lindstrom and Jeffries, 2004) As Kent Beck states, "Software development is always an evolving dialog between the possible and the desirable." Part and parcel of XP's Planning Game is to foster an on-going, productive environment where communication between the customer and development team is supported and encouraged. Beck goes on to say, "The more you communicate, the clearer you can see exactly what needs to be done and the more confidence you have about what really doesn't need to be done." (Beck, 1999)

The research indicated that User Stories -- an integral part of XP's Planning Game -- provided numerous advantages over detailed software requirements and their related use cases because User Stories were comprehensible by the customer and developers, facilitated easy and straightforward estimation, and provided an uncomplicated mechanism with which to plan. (Cohn, 2004) The utilization of User Stories also minimalized the sometimes adversarial relationship between customer and developers due to the simplicity of moving from story to development. (Marchesi, et al, 2002)

The references reviewed unearthed how – as a project increases in scope and size – User Stories can be challenging when it comes to maintaining the relationships and dependencies between them. (Cohn, 2004) When new interdependencies were unearthed as the project progressed, User Stories required rewriting to reflect the new dependent functionalities. (Marchesi, et al, 2002)

One final aspect of User Stories discovered during this research was the difficulty experienced by customers who were tasked with writing the stories. Often the customer did not possess ample domain knowledge to write the stories which resulted in the User Stories being deficient in enough detail from which to develop a system that solved the business problem. (Marchesi, et al, 2002)

Another essential facet of the Planning Game is allowing the development team to perform estimations of how long a specific feature will take to develop, determine the consequences of utilizing a particular database system or programming language, the composition of the development team, and what – within a User Story -- will first be developed. (Beck, 1999)

In Extreme Programming Perspectives, authors Johansen, Stauffer, and Turner relate how their company's management felt that the team's estimations signified a reluctance on the team's part to working at a quicker pace. This is in spite of the development team feeling as though they were able to estimate with accuracy. (Marchesi, et al, 2002)

Another company had difficulty in allowing their development team to determine their own estimates as management was accustomed to estimations predicated upon what they assumed the team could accomplish. (Marchesi, et al, 2002)

Additionally, one development team – in an attempt to impress their customer – actually overestimated what was possible to achieve within one iteration and ultimately fell short in delivering the promised functionality. (Marchesi, et al, 2002)

It was also discovered during this research project that if the development team did not have a clear idea of what Ann Griffin in her article from Extreme Programming Perspectives called “the big picture,” then the estimation of releases took more time than what was anticipated. (Marchesi, et al, 2002)

The intention of Extreme Programming's Planning Game is to supersede the need to develop from a Software Requirements Specification that describes in detail what is to be built, how many man-hours it will take to develop it, and how much the project will cost the customer. Kirk Barrett of Cimbrian, Inc. described to this researcher how difficult his sales and marketing staff found selling Cimbrian's services without the aid of at least some sort of preliminary software specification. (Barrett, 2005)

One development team took the novel approach of striking a contract with their customer where it was agreed upon that the client would be billed on a release by release basis, with the contract amended every two weeks. (Marchesi, et al, 2002)

*Small Releases:*

Schedule slips and project cancellation are the targets of the XP practice of Small Releases. The intention is to limit the scope of any slip by aiming for short release cycles of four to six weeks. Furthermore, each release cycle is made up of multiple iterations that allow the customer to see that progress is being made and to facilitate the customer in providing

less generalized feedback. Join this limitation of scope with the aim to develop the most important functionality first and – as Kent Beck claims – schedule slips and the risk of project cancellation are minimized. (Beck, 1999)

Pascal Van Cauwenberghe, in his contribution to Extreme Programming Perspectives, says that all knowledgeable software developers use an amalgamation of up-front design -- backed by detailed software requirements specifications and modeling -- and iterative development. Van Cauwenberghe is convinced that it is the rare developer who uses only a methodology such as the Waterfall method where a system is analyzed, designed, built, tested, etc. (Marchesi, et al, 2002)

Van Cauwenberghe feels that it is important for the development team to weigh the risks before choosing to strictly adhere to Extreme Programming's iterative approach. (Marchesi, et al, 2002)

*Metaphor:*

According to Kent Beck in Extreme Programming Explained: Embrace Change, an XP Metaphor is intended to be a shared vision of

what the complete system to be built is supposed to do. This vision -- or picture -- is to act as a compass for the customer and development team. (Beck, 1999)

Christopher Matts, on his website (<http://abc.truemesh.com>) puts forth the view that Extreme Programming's attempt to develop a single Metaphor for everyone involved in the project (customer, developers, project manager, etc.) is a flawed concept. He believes that the XP practice of Metaphor "has proven to be the least successful of the XP practices" because an individual's understanding of the overall vision of a system is filtered by their background and business knowledge. (Matts, 2003)

Gregory Schalliol, in his article featured in Extreme Programming Perspectives, describes the experience of working on a project where the development of a "readily available" Metaphor was complicated due to the complexities and size of the system his team was developing. (Marchesi, et al, 2002)

Amr Elssanadisy's contribution to Extreme Programming Perspectives also details the difficulty the development team experienced

formulating a Metaphor for the large system they were building. (Marchesi, et al, 2002)

Finally, an article written by Gittins, Hope and Wells for Extreme Programming Perspectives found the Extreme Programming practice of Metaphor so complicated that they decided to abandon it. (Marchesi, et al, 2002)

*Testing:*

Extreme Programming addresses two risks of software development with frequent and all-inclusive testing. The risk of a system in production having to be replaced because it is full of bugs and maintenance becoming too costly, and the risk of a defective system that is never put into production are combated in XP by running comprehensive tests after every change in code. These tests are written by both the programmers and customers; the programmers focusing on designing tests that check the system's stability on a function-by-function basis (called Unit Testing); the customers developing tests that check the system's workability on a feature-by-feature basis (called Acceptance Tests). The Acceptance Tests are intended to be automated to facilitate ease of use for a non-technical savvy customer. The ultimate goal is to

eliminate the need for a drawn out cycle of testing at the end of development and to act as a sign of progress for the customer. (Beck, 1999)

As new features are developed, the need for automated testing becomes more apparent because compatibility of new features must be tested, requiring not only the new tests to be run, but the former tests as well, prolonging the time required to run all the tests. (Cohn, 2004)

Kini and Collins, in their article Lessons Learned from an XP Project, found that the adoption of a laidback approach to testing resulted in increased time spent fixing defects. (Marchesi, et al, 2002)

Also, during this research project, it was discovered that development teams can find it a challenge to compel the customers to develop and run the Acceptance Tests due to the customer's reluctance to dive into the perceived intricacies of the system. (Marchesi, et al, 2002)

And finally, Gittins, Hope, and Williams concluded that a project that involved updating or improving an intricate legacy system did not easily lend itself to unit or acceptance testing. (Marchesi, et al, 2002)

*On-Site Customer:*

Three software development risks are addressed by the Extreme Programming practice of On-Site Customer, the risk of defects; the risk of not solving the business problem; and the risk of developing useless features. The On-Site Customer supports the XP intentions of having

1. continual feedback from the customer
2. a customer readily available for domain questions or clarification of features
3. a customer who will write the User Stories
4. a customer who will choose which User Stories are the highest priority
5. a customer who will perform the automated acceptance tests.

(Beck, 1999)

It is believed that numerous software development methodologies – XP or not -- could benefit from an on-site representative of the customer because the developers can get a much better grasp on what exactly the customer wants if the customer is readily available to communicate with. An On-Site Customer is considered so valuable a benefit that many suggest that the development team be a great deal more insistent in

requiring continual customer accessibility, regardless of methodology.  
(McBreen, 2002)

The practice of On-Site Customer in XP is considered by many in the field of software engineering and project management to be a somewhat pie-in-the-sky aspiration. An aspiration too difficult to satisfy with just one individual, because finding a customer representative who can write the User Stories, prioritize them, answer domain and business questions, and represents someone who will actually use the system upon its completion, is often unattainable. (Cohn, 2004)

Pete McBreen feels that – if the customer is deficient in indispensable domain knowledge – taking the time to find someone within the customer's organization who can provide the proper information can delay progress on the system's development. McBreen also stated that it may be quite difficult convincing the customer to lend someone with so much knowledge to the development team for the lifetime of the project.  
(McBreen, 2002)

Repeatedly, the reference material exposed the opinion that a team of individuals (Customer Team or Customer Proxy) would be a much more

realistic approach to satisfy the need for an On-Site Customer. (Lindstrom and Jeffries, 2004) However, amassing a group of individuals to act as the On-Site Customer could cause issues if each Customer Team participant had a personal agenda relating to the software system's scope and capabilities. (Marchesi, et al, 2002)

Another approach to the practice of On-Site Customer was relying upon the development company's Business Analysts assuming the role. This was often the preferred alternative when the developers were part of a third-party company and not the customer's employees. If the customer was reluctant to provide an employee with ample business and domain knowledge, then the Business Analysts stepped into the role. Nevertheless, no matter what the extent of the analyst's knowledge about their customer's business, there were always questions that required a response from the customer. Subsequently, delays in response from the customer had an adverse effect upon the system's development. (Barrett, 2005) (Marchesi, et al, 2002)

## **CHAPTER 5**

### **Extreme Programming Practices and Their Effectiveness in Requirements Gathering**

#### **Summary, Conclusions and Recommendations**

##### **Summary**

As Kent Beck states in Extreme Programming Explained: Embrace Change, “The basic problem of software development is risk.” (Beck, 1999) Whenever a department or company decides that they need some new software or web-based application to solve a particular business problem, they are dependent upon a development team to build that system for them. There begins a process where the development team needs to elicit from the customer what it is the customer wants the system to do and what business problem the customer wants the new or revised system to solve; then – from that information – the developers are tasked with building a system that actually does what the customer wants it to do and solves the customer’s business problem. And, that process is risky.

Will the system be ready when promised? Will the system be virtually free of bugs and defects? Will the application solve the business problem? Will the business change and render the problem the system was intended to solve inconsequential? Will the system be burdened with unnecessary features? Will the project take so long to complete that all the qualified developers leave?

For decades, there has been reliance within the software development community upon a detailed software requirements specification (SRS) to act as a roadmap in the development process. This specification is often the end-product of requirements analysis, which can be an extensive and difficult process. Business or systems analysts conduct interviews, observe processes, conduct focus group sessions, etc. Then the analysts document what they have discovered and – often with the addition of repeated steps – produce the software requirements specification. With the software requirements specification in hand, the development team goes off and begins development of the system described.

Critics of this methodology believe that development based upon a written software requirements specification is, because of the inexactness

of written language, imprudent, because it does not foster an environment of on-going communication and interaction between customers and developers. Subsequently, as the customer's business changes, certain sections of the software requirements specification become obsolete while new, previously undiscovered features beg for addition to the document. (Cohn, 2004)

Furthermore, the unavailability of the customer as an intimate participant in the on-going process of development further attenuates the potential for failure because the software requirements specification – and the analysts, project managers, and developers interpretation of it – becomes the final word on, and the overarching standard by which, the system being developed is compared. (Beck, 1999)

Extreme Programming (XP), a so-called agile methodology, came into being due to these shortfalls and risks. Spearheaded by Kent Beck following an experiment in the XP methodologies at Daimler-Chrysler in 1996, Extreme Programming was embraced by many in the software development community as a workable solution to the perceived shortfalls of developing from a written software specification. Here they envisioned a way to avoid lengthy requirements documents. Here they saw a method

that would get the customer more closely involved in the day to day development process. Here the development community envisioned an opportunity to obtain immediate feedback from the client. Here they saw a way to ensure that the system being built would in fact solve the business problem. Here was a way to avoid building a costly, resource-consuming, time-consuming, and obsolete system that would sit on the shelf collecting dust. (Beck, 1999)

The goal of this final paper is to evaluate whether the stated effectiveness of Extreme Programming in solving the risks in developing from a detailed requirements specification were more than half-empty promises. Consequently, in this paper, five core practices of XP are reviewed:

1. The Planning Game – customer determines scope, priorities, composition and dates of release. The development team makes the estimates, considers the consequences, determines the process, and decides what will be done first within an individual release
2. Small Releases – getting something of value to the customer quickly and continually

3. Metaphor – an overall shared vision or description of what the system will be
4. Testing – test frequently and continually. Customers run the acceptance tests that validate that the requested functionality has been built
5. On-site Customer – a real customer on-site during development.  
(Beck, 1999)

### **Conclusions:**

In the not-so-perfect world of software development, where customers can be half way round the world, where development might be outsourced to India, and where some developers may be reluctant to change their long-practiced ways, the practices of Extreme Programming could be considered as lofty and unreachable goals.

However, if an On-Site Customer who:

1. truly understands the ins and outs of his business
2. possesses domain knowledge sufficient enough to be an invaluable asset to the development team

3. is open to writing User Stories
4. is willing to write and run the Acceptance Tests
5. is readily available to answer questions and provide clarification

...is provided by the client, then Extreme Programming's four core practices of Planning Game, Small Releases, Metaphor and Testing can most likely be executed with a degree of success. And, if these core XP practices can be successfully executed, Extreme Programming is documented as being an effective method in overcoming the shortfalls of developing software or web-based applications from a detailed software requirements specification.

As the relevant literature was reviewed, it became apparent that in order to have a Planning Game you need an On-Site Customer to write the User Stories and set the priorities. To practice Small Releases, you have to have an On-Site Customer available to choose which release to develop first, and then next. To devise a Metaphor of the system to be built you need an On-Site Customer with which to brainstorm. And to run the acceptance tests, you need an On-Site Customer present to analyze the results and determining whether the release being tested functions properly. (Beck, 1999)

This research further revealed that a great deal of Extreme Programming's effectiveness hinges on the On-Site Customer. So much that the reported problems and failures documented in the literature were in some way related to the lack of an available and/or well-informed individual to fill that role. (Lindstrom and Jeffries, 2004) (Marchesi, et al, 2002)

In much of the material reviewed, development teams were required to substitute a team of client representatives -- each with her own more narrow understanding of the business's domain and grasp of the business problem -- for the On-Site Customer. (Marchesi, et al, 2002) And, when the client could not or would not provide resources to be in charge of the On-Site Customer function, the development team had to rely upon a business or systems analyst from the development team's own department or organization to assume the On-Site Customer role. And, evidence suggested that as the role of On-Site Customer became altered to fit the circumstances of a particular project, the effectiveness and value of having a knowledgeable client representative was diminished. (Marchesi, et al, 2002) (Barrett, 2005)

However, nowhere in the research material was an anecdote regarding what the experience was like to actually have access to a true XP On-Site Customer. As this research was far from exhaustive, there may be literature that describes what that experience might be. But none was reviewed that had such information.

Additionally, the avoidance of detailed documentation also affected the efficiency of marketing and sales efforts. The challenge arose where the sales department discovered that the marketing of Extreme Programming's iterative development process where documentation was limited – or, in some cases, eliminated – complicated and ultimately unsuccessful. (Barrett, 2005)

This research also revealed the difficulties associated with the Extreme Programming practice of Metaphor. Kent Beck states that the Metaphor is intended to be a shared vision of how the system to be built functions. (Beck, 1999) The successful development of a system Metaphor was especially challenging when large systems were being developed or in the redevelopment of legacy systems. Subsequently, in one circumstance, the practice of Metaphor was discarded. (Marchesi, et al, 2002)

According to Kent Beck, the practice of Small Releases is focused on overcoming the risks of schedule slips and project cancellation. Each Small release is intended to span a four- to six-week period and be additionally divided into smaller iterations. An adjunct to the practice of Small Releases is the User Story, part of XP's Planning Game, which is to be written by the customer – because the customer is in a better position to describe how the system should work (Cohn, 2004) -- and act as “a reminder to have a conversation.” (Beck, 1999)

By their nature, User Stories encourage an open dialog between the On-Site Customer and the development team because they are descriptions about a single piece of functionality. To migrate from story to development requires a conversation to facilitate the accumulation of enough detail from which to begin coding. (Marchesi, et al, 2002)

In non-XP development projects, strict adherence to development utilizing the Waterfall method is uncommon, having been replaced by a more iterative approach quite analogous to Extreme Programming's practice of Small Releases. It has become natural for developers to move back and forth between analyzing, designing, coding, integrating, testing and releasing as the customer's requirements change or as new iterations

expose aspects of earlier functionality which inhibits dependencies.  
(Marchesi, et al, 2002)

The Extreme Programming practice of allowing the developers to estimate the time necessary to complete a release does have its negative aspects. For example, if the development team commits to more than they can complete in a particular iteration, they may be tempted to forego sufficient testing or take some other shortcut to make up for lost time.  
(Marchesi, et al, 2002)

It is also apparent that upper management may be wary of allowing the developers to “take the reins” on estimating the releases and ultimately driving the pace of the project or by feeling that the development team was accomplishing less than they were capable of accomplishing.  
(Marchesi, et al, 2002)

Finally, there will always be customers who still require some sort of documentation as a means to satisfy requirements traceability by documenting each iteration as it is identified and tested, and any changes as they occur. (McBreen, 2002)

## Recommendations

It is imperative to note that a great deal has been written about the Extreme Programming methodology, volumes more than what was researched for this final paper. Subsequently, this paper represents research that is far from comprehensive. However, this research project did unearth numerous positive and negative implications related to utilizing XP practices in place of a detailed specification that plead for further analysis.

It would be of enormous benefit to those in the software and application development fields to conduct additional investigation into the effectiveness of XP practices in substituting the need for a written software requirements specification with Extreme Programming's core practices of Planning Game, Small Releases, Metaphor, Testing and On-Site Customer. It is particularly recommended that further extensive research be performed in how the role of On-Site Customer can be *successfully* replaced or augmented by customer teams or, more importantly, business analysts. There's simply not enough anecdotal and objective evidence readily accessible related to overcoming the unavailability of a qualified XP On-Site Customer.

Extreme Programming's popularity is motivated by the software development and software project management community's desire to build systems that actually satisfy their client's requirements and resolve the client's business problem. Cost overruns, schedule overruns, obsolescence, etc. are much too prevalent in software development to be disregarded. In response, many in the software development community have embraced Extreme Programming as a viable solution to these issues. However, not unlike the plague-infected townspeople queuing up at the back door of a traveling medicine show wagon for a miracle cure, those in the software development community may be waiting in line for a magic elixir that often leaves a bad taste in their mouth and doesn't always work as promised.

Development teams must realize that accomplishing the goal of filling the role of On-Site Customer with an individual who has abundant domain knowledge may be unattainable. It is important to understand that acting as an On-Site Customer requires a commitment of time from an individual who will most likely be considered more valuable elsewhere (McBreen, 2002), or be unavailable due to other business commitments or physical location. (Lindstrom and Jeffries, 2004)

Because the effectiveness of Extreme Programming relies so heavily upon the availability of a domain-savvy On-Site Customer, development teams are advised to be aggressive in requiring that the client provide someone to fill that role (McBreen, 2002). In fact, Mike Cohn, in his book User Stories Applied: For Agile Software Development, recommends that the On-Site Customer role be a team of individuals consisting of users, quality analysts, project managers, etc. (Cohn, 2004) Lowell Lindstrom and Ron Jeffries also propose that a team of people who communicate with one voice assume the On-Site Customer role. (Lindstrom and Jeffries, 2004) However, it is suggested that developers and project managers be aware of issues arising where members of the customer team may have individual requirements that are incompatible with other members of the team. (Marchesi, et al, 2002)

This research indicates that there exists a high probability that the client will not be able to make available anyone to fill the On-Site Customer role, requiring instead that the development team utilize business or systems analysts from within their own department or organization to act in that capacity. In his book, Questioning Extreme Programming, Pete McBreen advises that this questionable substitution

be avoided as it may expose the project to “risks related to misunderstanding the business.” (McBreen, 2002)

Assuming that the role of On-Site Customer has been satisfied and she has defined the scope of the project, set the priorities, defined how much needs to be produced before her business realizes a benefit, and determined when it would be best to have the system completed, then the User Stories can be written. Mike Cohn agrees with Kent Beck (Beck, 1999) in recommending that the stories be written by the On-Site Customer. The User Stories must be composed in language that is understandable by the customer as stories that are simple to comprehend can then be prioritized and considered for addition into a particular release. (Cohn, 2004) The development team is also advised that the customer may require more documentation than the stories provide. Daily status reports documenting what was accomplished that day may suffice (Marchesi, et al, 2002), or – to satisfy the customers desire for requirements traceability, for example – documentation at the onset of each iteration cycle should be considered. (Cohn, 2004)

Dividing the project into Small Releases allows the customer to see progress often and regularly, and affords the development team the luxury

of estimating small chunks of functionality instead of a complete system. Pascal Van Cauwenberghe recommends that developers give strong consideration to determining whether utilizing Extreme Programming's strictly iterative approach to development actually decreases the risk. A combination of the standard Waterfall approach and an iterative approach may be a more prudent route to travel. (Marchesi, et al, 2002)

Formulating the Extreme Programming Metaphor that acts as a holistic vision of what the system will actually do can provide an insurmountable challenge to the project team, especially when the system to be built is large and complex. Gregory Schalliol, in his article Challenges for Analysts on a Large XP Project, recommends that the project team put together a chart that displays the relationship between the functionalities of the system. (Marchesi, et al, 2002)

Kent Beck recommends that programmers write unit tests and customers write acceptance tests. The unit tests check that the functionality developed actually works. The acceptance tests demonstrate that a feature has been completed. (Beck, 1999) He goes further to suggest that the customers work closely with the development team to learn what aspects are helpful to test and to avoid redundancy. Beck also

recommends that each individual on the development team assume the role of XP tester. The XP tester's responsibilities include helping the customer select and write the acceptance tests, and displaying the outcome of the tests in some prominent area for all on the project to see and review. (Beck, 1999) It is also suggested that customers may require additional coaching if she does not feel comfortable with writing tests. (Marchesi, et al, 2002)

Mike Cohn advises automating the acceptance tests because – as the system increases in complexity – testing can be protracted owing to the additional complexity and interactivity of each new release. (Cohn, 2004) Robert Gittins and Sian Hope, in their article entitled Qualitative Studies of XP in a Medium-Sized Business, go so far as to propose that the development team may decide to forego unit testing altogether when working on updating a complex legacy system. (Marchesi, et al, 2002)

In an article entitled The #1 Serious Flaw in Extreme Programming, Conrad Weisert recommends that, prior to an company or organization can make an evaluation as to the appropriateness of software systems, it must have a strong grasp of the business problem the application is intended to solve. Weisert suggests that – instead of Extreme

Programming's no upfront documentation approach – the most proven methodologies require business or systems analysts to discover and document detailed requirements. He proposes that – as changes to the requirements become necessary – the development team evaluate how these alterations will affect the project's schedule and cost, and make prudent decisions as to whether they should be integrated into the system. (Weisert, 2002)

Pete McBreen states that “XP projects have woefully inadequate documentation because from a literate culture standpoint, an oral tradition is a completely inadequate means of remembering information.” (McBreen, 2002) He recommends that the On-Site Customer write and schedule User Stories that specifically request what he calls “maintenance documentation,” documentation that enables those responsible for maintaining the system to understand its overall design. He continues by suggesting that the XP development team not disperse until they have created a document that details the internal workings of the system. (McBreen, 2002)

A statement made by Kirk Barrett after the process meeting at Cimbrian, Inc. summarizes much of what has been discovered during this research project:

“When I first heard about XP, I wanted it to work. We needed to pull the reins in on our customers changing their minds halfway through a project. And, since most of our contracts are a result of word-of-mouth, we wanted to produce applications that would help grow our business. XP seemed to be the ticket. But it’s become obvious – though some of my team might disagree – that we can’t sell XP to our customers if we have to require an on-site representative. They just won’t make the commitment. And I don’t see any way to do all the other practices [of XP] if we don’t have someone we can communicate with on a day-to-day basis.” (Barrett, 2005)

There is an interdependent relationship between the core practices of Extreme Programming. Separately, the practices are somewhat inconsistent. But in tandem, there is evidence that they can result in a strong and reasonably workable methodology. However, it could very well be this almost symbiotic connection that renders XP an undesirable alternative.

Only time, experimentation, and further investigation will tell if the Extreme Programming core practices of Planning Game, Small Releases, Metaphor, Testing and On-Site Customer can successfully supplant the need for a detailed software requirements specification. The evidence is, frankly, inconclusive.

## Bibliography

Barrett, Kirk (2005) Interview and Process Meeting

Cimbrian, Inc., Lancaster, PA

Beck, Kent (1999). Extreme Programming Explained: Embrace Change.

Boston: Addison-Wesley

Bennatan, E. M. (2000) On Time Within Budget: Software Project

Management Practices and Techniques

New York: Wiley & Sons

Boehm, Barry (1981). Software Engineering Economics

Upper Saddle River, NJ: Prentice-Hall

Cohn, Mike (2004). User Stories Applied: For Agile Software Development

Boston: Addison-Wesley

Emery, Patrick (2002) The Dangers of Extreme Programming

Downloaded from the Internet 7/25/2005

<http://members.cox.net/cobbler/XPDangers.htm>

Lindstrom, Lowell and Jeffries, Ron (2004). Extreme Programming and

Agile Software Development Methodologies

Information Systems Management Magazine,

Summer 2004 Issue, 41-52

Marchesi, Michele; Succi, Giancarlo; Wells, Don; Williams, Laurie; Wells,

James Donovan (2003) Extreme Programming Perspectives

Upper Saddle River, NJ: Pearson Education

Matts, Christopher (2003) Metaphors and XP

Downloaded from the Internet 8/13/2005

<http://abc.truemesh.com/archives/000090.html>

McBreen, Pete (2002). Questioning Extreme Programming

Boston: Addison-Wesley

Weisert, Conrad (2002). The #1 Serious Flaw in Extreme Programming

Downloaded from the Internet 8/15/2005

<http://www.idinews.com/Xtreme1.html>