

# One XP Experience: Introducing Agile (XP) Software Development into a Culture that is Willing but not Ready

F. Grossman<sup>1</sup>

J. Bergin<sup>1</sup>

D. Leip<sup>2</sup>

S. Merritt<sup>1</sup>

O. Gotel<sup>1</sup>

Pace University<sup>1</sup>

Computer Science & Information Systems  
{grossman, bergininf, smerritt, ogotel} @ pace.edu

IBM<sup>2</sup>

Hawthorne Lab – Corp. Webmaster Team  
Leip @ us.ibm.com

## Abstract

The main question to be asked is "Does Extreme Programming (XP) make sense as a development methodology in a diverse, multidisciplinary web development environment? This environment includes diverse, and perhaps, distributed teams requiring close coordination with multidisciplinary skills -- information architecture, visual design, XML, Java and others. The potential is to make the development process more responsive to users' needs and changing business requirements. This could have high impact on outcomes of the development process, decreasing cost, decreasing time to deployment, and increasing user satisfaction. The challenges are to adapt and reconcile the corporate and the agile culture processes and methodologies without seriously compromising either. We will discuss our experience from conception into implementation of XP through the first release that incorporates several iteration cycles. We will discuss the positive and negative forces and how they have or have not been resolved to date.

## 1 Introduction

Corporate software development teams, such as IBM's own internal web development teams, have traditionally used heavyweight waterfall methodologies for developing most web-based applica-

tions. This has worked well in many situations but less well in others. In particular, it is not sufficiently responsive to changing requirements or to situations in which the ultimate clients have difficulty stating stable requirements with precision. The latter can arise when the business needs are volatile and also when the technology opportunities are not well understood by the "customer." Another difficulty is that the development team can be isolated from the ultimate client of the system by a long process in which strategies are set and information architects and visual designers (and others) contribute their work to the overall project. The implication is that a question on requirements from the software developers must filter back to the decision makers through many other people and groups, and the answer then filters forward through the work of those groups, resulting in long delays. During these delays it is difficult for the development team to be productive on the project unless they make (likely incorrect) assumptions about what is required.

The above unhappy scenario is not unique to IBM's development team, actually, and has been observed in many organizations. A response to this has been the development of agile methodologies that tend to shorten the distance between decision-makers and developers and to increase flexibility. Whereas the traditional development methodology envisions gathering all the requirements at once before other work begins, agile development works by gathering requirements "just in time." The development team works closely with a designated "customer" who makes all business decisions and who specifies important

---

Copyright © 2004 Dr. Fred Grossman, Dr. Joseph Bergin, and IBM Corp. Permission to copy is granted provided the original copyright notice is reproduced in copies made.

features of the growing application determined by the immediate needs prioritized in business-value order. The "customer" in effect steers the team toward the goal over the life of the project, and the developers realize the current features/needs on short iterative cycles. Every few weeks the customer has business value delivered by the project team. The cost of change in this scenario is exactly the cost of adding a new feature, making change manageable. The result is that it is much easier to hit a target that could not easily be seen at the beginning of the project.

Extreme Programming (XP) is one example of such an agile methodology. It is a high discipline process in which a number of synergistic practices bring business value to the customer in short (two week) iterations with deployable software coming at a slightly slower (four to six weeks) rate. At any time, the business stakeholder has a good idea of the current state of the project and also the cost of continuing. Normally the cost per feature rises over the life of a project while the value per feature declines (build the high value things first). When the curves cross, it is time to quit. One important overriding feature of an XP project is that iteration and release schedules never slip, though some of the tasks may not be completed (feature slip). The consequence is everyone really knows both the value of the current "build" and the cost of continuing.

Others have investigated using XP or other agile methods as the official corporate software development methodology [11]. This paper discusses early experience of applying XP to a single, albeit significant, project in the ibm.com domain that has characteristics that may not make it particularly suited to their traditional (waterfall) development methodology. For this project it would be difficult to specify requirements and design choices well in advance, necessitating many questions back and forth. With the traditional methodology, the ensuing time lags would cause problems.

The main question to be asked is "Does XP make sense as a development methodology in the ibm.com environment?" This environment includes distributed teams and close coordination with information architects, visual designers and others. The potential is to make the development process more responsive to users' needs and

changing business requirements. This could have high impact on outcomes of the development process, decreasing cost, decreasing time to market, and increasing user satisfaction.

The challenges are to adapt and reconcile the corporate and the agile culture processes and methodologies without seriously compromising either. These involve recognizing and understanding

- who the XP "Customer" really is
- test-first development is unnatural to several of the disciplines
- pairing with a limited staff and diverse staff-discipline sets
- XP assumes that all developers have more or less the same kind of skills, e.g., Java programming, but in a multidisciplinary team this is not the case
- the interface between the agile and the non-agile aspects of the project and the organization
- team members work on multiple projects which affects maintaining sustainable pace and scheduling
- different practices/culture for different parts of the team -- the 12 XP practices may not be natural or even valid for different sorts of practitioners
- distance, or more precisely lack of strong communication, is expensive

We will discuss our experience from conception into implementation of XP through the first release that incorporates several iteration cycles. We will discuss the positive and negative forces and how they were or were not resolved.

## 2 XP and Web Projects

Web development projects require the skills from such disciplines as information architects, visual designers, programmers (XML, HTML, Java), and user-centered design specialists. They each have their own established practices and bring different perspectives to the web project development. [12]

Will XP work in a culture where

- traditional development methodology is waterfall
- there are a large number of stakeholders in diverse organizational roles and responsibilities

ties (management distributed across independent authorities)

- diverse skills (information architects, visual designers, XML, HTML and Java programmers) require a multidisciplinary team with continuous synchronization and integration

This paper explores the early decisions about the introduction of XP into the ibm.com development environment and some of the early experience (successes and problems). We examine the interaction of organizational culture and agile development methodology, and to what extent organizational and methodology changes are necessary to successfully employ an agile web development process.

Traditionally this organization operates in a staged waterfall scheme in which information architects and strategy specialists work with real customers to develop a complete set of requirements for a project. This specification is refined through a process of showing alternatives to real users. This specification then goes to a team of visual designers who realize the designs of the information architects. At this point the requirements and external design work is finished, but it is only then that the programmers begin the task of implementing the functionality and actually creating web pages and the application structure. In the past, the programming team thought of the information architects and visual designers as their customer; but this is a long way from the real customer. It became obvious that a question from the programmers on the team about requirements would potentially take weeks to be answered, and a change in the real requirements of the real customers would only filter to the programming team after weeks of work. Therefore there was no way to be agile solely within the software developers' domain. Instead, a decision was made to include everyone with essential skills in the development of the project in a "whole team" approach. This puts the team close to the real customer, but also implies that the skills of the team members are not interchangeable.

## 2.1 XP Practices

Kent Beck in his seminal work [1] specified 12 XP practices:

- The Planning Game

- Small releases and short iterations
- Metaphor
- Simple design
- Testing
- Refactoring
- Pair programming
- Collective ownership
- Continuous integration
- 40-hour week (sustainable pace)
- On-site customer
- Coding standards

In the next sections we will discuss which aspects of these practices were easily adopted and worked from the beginning, and which were more difficult to inculcate. We also incorporated some additional practices, e.g., daily stand-up meetings and retrospectives after each iteration cycle.

## 2.2 Introducing XP to a customer: a representative customer is willing

Stakeholder managers representing strategy and design, that we thought might be the "customer", were given a short overview of XP from a customer perspective (XP in an Hour). As it turned out, only one of these stakeholder representatives (design) would be a "customer" for the XP project.

Each agreed to select a member of their group to be a potential onsite customer. These two customer representatives would attend the training sessions for the team. As it turned out, neither of these selected customer representatives became real customers on the XP team. More will be said about this later.

## 3 Training the development team: the developers are willing

The Webmaster development team wanted to try XP. They had read some of the XP literature [1, 2, 12] and believed that it might work well in their development environment. The team was primarily composed of two programmer skill groups -- XML/HTML and Java. The initial plan was to select an initial XP project that was not mission

critical and whose requirements and design specifications could not be completely determined in advance of development efforts.

Two of the authors (Bergin and Grossman) were brought in to train the team and to act as coaches for the initial XP project. The entire Webmaster development team (including the distance members) was assembled for two half days and one full day in between. This amounted to about 24 people including programmers, strategists, information architects, visual designers and user-centered specialists.

Note that this training session was held one month after the "XP in an Hour" presentation to the anticipated customer.

The training methodology used was the Extreme Construction Game [4]. This took the full day. The two half days were used for an introduction to XP (XP in an Hour) and two retrospectives [6]. The first retrospective was to look at the way web projects have been developed in the past by this group. They were asked to envision a composite project that was representative of all the projects that they had worked on. This was held prior to the Extreme Construction Game. As a result of this retrospective of prior project development projects, most of what was determined to be in need of improvement should be resolved using agile development practices. The second retrospective was held on the second half day following the day of the Extreme Construction Game, and was a review of the game and the XP process overall as they understood it.

After this training session, the team members went back to their normal project work. It would be two months before the actual project work would begin. During this two-month period, additional "XP in an Hour" sessions were held with various management personnel from the stakeholder divisions of the organization.

### 3.1 Extreme Construction Game

Once it was decided that the team included people with many diverse skills, it became necessary to initially train everyone in XP practices. Traditionally an XP team includes a few non-programmers, e.g., the customer, but this was an extreme case as programmers were actually in the minority. On the other hand, XP and its practices were origi-

nally considered to be directed at programmers. There are a number of available exercises that are used to train diverse groups quickly in XP practices. One of the most common is the Extreme Hour [8] in which people draw pictures at the direction of a "customer" who writes stories specifying requirements for the picture. While it is quick, it isn't very deep. Two of the authors of this paper have developed a more elaborate "game" [4] that takes about a day, including a retrospective, and includes most of the XP practices. Its important features are:

- it is accessible to anyone -- developers, customers, managers
- it gives a basic understanding of the agile XP practices and how they fit together
- it expects the participants will make many mistakes in following the rules (XP practices) that cause them problems in the game, providing grist for the important follow up retrospective.

The basic idea of Extreme Construction should be familiar to anyone who has been to kindergarten. The facilitators gather a large collection of simple construction materials, from paper and glue to modeling clay, sticks, scissors, and the like. Some of the participants are chosen to be "customers" and they come up with an idea for something to be built with the available materials. Teams of about 10 participants work with a customer. In our game session, one customer decided to specify a "playground" and the other a "castle" complete with dragon. The customers then write stories for features of their project. The rest of the team estimates the stories and computes a "velocity" as in XP. The customer then chooses important stories from those estimated up to the given velocity and the team then begins a short (20-30 minute) iteration. We enforced pairing and test-first development for the construction phase, and emphasized the importance of communication with the customer.

The facilitators watch what is going on and coach the process, but mostly prepare for a retrospective in which the team tries to discuss what should have happened, what did happen, and how the process could have been improved. The participants came away thinking that they had a good idea of XP, but in fact it was quite shallow and

needed much reinforcement. In particular, we have not yet determined a manner in which to reinforce the concept of test-first development in anything but a superficial way.

Many mistakes were made, of course, that caused the construction to be less efficient and effective than it could have been:

- assuming (wrongly) that the developers knew what the customer wanted rather than asking the onsite customer
- stories were not well written by the customer
- turning design into a big bottleneck up front.
- building infrastructure that wasn't needed as the project evolved
- pairing was abandoned when time was running out for an iteration
- estimates were (not very good) guesswork

An additional problem is that many of the people that we trained in this way were not part of the real project team, including both of the real customer representatives.

## **4 The composition of the XP team**

As in any XP project, the team consists of developers, customer representative(s), tracker(s), coach(es) and manager (big boss). What makes our team different is that the developers and customers come from diverse and distinct organizational areas and with diverse non-overlapping and non-shared skill sets. This presents both a challenge and an opportunity.

The initial multidisciplinary team was composed of developers (information architect, visual designer, XML & Java programmers), trackers (project manager for webmaster programmers and project manager for the design group), coaches, and a customer entity.

### **4.1 Who are the XP developers in a web development project?**

At the onset of discussions about doing an XP project, both the Webmaster and the XP trainer/coaches assumed that the XP developers would be the Webmaster programmers (XML/HTML and Java). However, after the training session, discussions with the original "cus-

tomers" representatives in attendance revealed that this was probably not a valid assumption, at least not if we wanted to get the maximum benefit from an agile approach. The roles of these "customer" representatives were strategy and information architecture, and in the waterfall approach to project development, they did play the role of the customer. What became obvious was that these roles (or at least the information architect role) are actually part of the developer group and not the customer group since they play essential roles in the creation of delivered artefacts for the project. Thus it was necessary to push the role boundary between the developer and the customer back. The question was, how far. Throughout we have attempted to enforce the "whole team" concept in which everyone considers herself to be a "developer" who embraces the XP values: communication, simplicity, feedback, and courage.

### **4.2 Who is the XP customer?**

Initially, it was assumed that the XP "customer" was to be the Webmaster's traditional customer, i.e., those who gave the specifications to the programmers. These typically were the information architects and the visual designers. However as stated previously, these roles were now incorporated as developers on the XP team. So then who is to play the XP customer role?

The obvious place to start was with those who would normally provide the specifications for web development projects. Initially managers from these groups (strategy, information architecture and visual design) were presented with an overview of the XP methodology focussing on the "customer" role. The "customer" is willing.

## **5 Thinking agile in a highly disciplined, structured and distributed culture**

When a software development project is done utilizing an agile methodology such as XP, the agile thinking is focussed on getting the code written, tested and accepted by the customer -- pair programming, small releases, simple design, test first, refactoring, collective ownership, continuous integration. It is usually clear who the developers are and who the customer is. The agile practices fit nicely into the organization, which as

a result of doing an agile development has become agile. But note that for most agile projects, the primary development task is programming. Here this is not the case, with information architects, especially, taking a key creative role.

Now consider a scenario in which the developers come from different parts of the organization with diverse kinds of skills and different management reporting lines, and where the boundary between the XP developer and the XP customer is blurred because they come from the same group. In our web development environment, it is clear that the XML and Java programmers are developers. Their management reporting lines are clear and consistent with traditional software development organizations.

However, the multidisciplinary team also includes developers who are information architects and visual designers. They have a management reporting line separate and distinct from the programmers. Yet they work on the same customer stories with the programmers and participate in the estimation process and velocity determinations that drive the planning game. What's more they are still operating in a "waterfall-like" culture in which large amounts of information architecture and design are expected to take place before a set of XP stories can be written to develop what they have architected and designed. As Kent Beck says, "The biggest barrier to the success of an XP project is [business] culture [1]".

In the multidisciplinary "whole team" approach that we are following, when the customer writes a story, all the developers (information architects, visual designers, XML programmers and Java programmers) will consider it for customer questions, estimation and tasking.

## 6 The Project

Against standard wisdom of not picking a mission critical project as the first pilot XP project, we did, in fact, pick an important project. We did this because the business stakeholders and the Webmaster did not think that it would be possible to complete the project in the desired schedule using a traditional waterfall model.

Requirements are assumed to change. They submit regularly to user testing, and then make adaptations. This is done often. In the traditional mode they built prototypes (not fully functional). XP

gives them the ability to do this and have a robust product reducing duplicate effort.

The project has a fixed delivery date and a strong sense of what high-level functionality and "design" must be in place in order to deploy.

## 7 The XP practices in the context of the culture

The project has been agile since its inception. There is more to XP than the practices. In fact, Kent Beck does not list them in his seminal work, *Extreme Programming Explained* [1], until the reader is into the last two-thirds of the text. While the focus of bringing XP into a project tends to stress the practices, they are really instantiations of the four values of XP – communication, simplicity, feedback and courage. Robinson and Sharp [10] talk about the XP culture and the significance of the XP practices, and present empirical evidence that the XP practices create and support a community in which the XP values are sustainable.

It is for that reason that we intended to introduce and use all twelve practices immediately. However, this did not actually occur. The following describes which practices fit into the existing culture well and were successfully used, and which are requiring more effort and adaptation.

It should be noted that the project is going very well. The customer is happy with the functionality that they are getting and the developers are happy with what they are able to deliver. This is happening in spite of the fact that the practices are not yet well installed into the culture, and some of the most important practices – pairing, test-first development, common code ownership, and continuous integration – have not been used very much in the early iterations. The coaches have explained to the team that this is not unusual. The power of the synergistic effect of using *all* the practices together is only going to become important as the project size and complexity begin to grow. XP has also allowed the project to progress further in a short time than previous methodologies would have allowed.

### 7.1 The practices that worked

We continuously reinforced the notion that while many of the practices might seem to have value as

a stand-alone method, the power of the XP approach is they are synergistic and are designed to work together. The acceptance and observance of the practices is an evolutionary process. What we describe here are some observations of what worked and why.

### **7.1.1 The Planning Game (story writing and prioritizing)**

The Planning Game is an activity that develops and depends upon a sense of common purpose, responsibility and understanding. A large amount of discussion takes place among the “whole team”, which includes the customer, and a lot of “what ifs” are considered. In fact it is the only time when our entire team is together in one place. This is important because the customer is represented by two organizational entities that are working together to produce a single end product (web site), but have different areas of responsibility. Each customer entity contributes different stories and has their own story priorities. This has not been a problem, and the two customer entities work and coordinate very well.

The first planning game produced about 40 stories and took more than a full day. Subsequent sessions have been taking one half day or less. The project is using two-week iteration cycles and the customer has not had any difficulty in prioritizing and selecting stories for each iteration.

### **7.1.2 Small releases and short iterations**

It was easy to convince everyone to use a two-week iteration cycle, and all have been pleased with the results. In one case, we tried a longer three-week iteration due to holiday considerations, and the results were not as satisfactory. This solidified the resolve to work in small bits. Unfortunately we were not initially able to convince the customer to think in terms of releases, and as a result they were unhappy at the end of an iteration when they didn't have any new business value. This pushed them back toward the proper path. The first release is usually an anomaly anyway, so not much has been lost. Most of the stories written are sized at around two ideal developer days, and this has made a good fit with a two-week iteration. A story larger than six ideal days won't fit into an iteration unless it is broken into tasks, and this forces the team to keep the work units small.

One special aspect of this project needs to be mentioned. The staged nature of the process has not disappeared completely. The information architects must do their work before the visual designers do their tasks, and both must complete before the programmers (XML and Java) can finish. We win in two ways here, however. We learned that usually each set of skills can begin before the one it depends upon completes, providing quite a lot of overlap. Second, the short iterations have kept this staging simple and quick, without requiring management, and yet able to respond quickly to changes.

### **7.1.3 Metaphor**

Since the project is a redevelopment of an existing web site, everyone on the team understands the essence of the project. Additionally there are design mock-ups and prototypes produced by the strategy, design and site architecture groups, as well as preliminary feedback from various sets of users.

### **7.1.4 Simple design**

If anything, this has been overdone. The rule “do the simplest thing that could possibly work” was overused a bit at the beginning. Eventually everyone learned that you do what you know you have to do and you do it well, not skimping on the essentials. So the developers are building a quality product without overbuilding it or anticipating the customer too much. There is a bit of disconnect between the customer who is used to simpler projects, and the team who knows what it requires to build a stable, scalable product, but this has been relatively minor. The team is building what is needed today, keeping flexible for changes that will occur tomorrow.

### **7.1.5 40-hour week (sustainable pace)**

XP is not just about the quality of code, but it also about the quality of life. It is not just about working a 40-hour week; the goal is for each developer to work at a comfortable pace so that they can leave at the end of the day without feeling stressed and exhausted, and then be fresh when they return the next day. Each team member should be in control of setting their divide between work and home, and to be able to move between the two comfortably. This also gives business value to the business stakeholders since

the developers are able to do their best work at all times, making far fewer errors than when stressed and tired.

We have included this practice in our success column because the overall atmosphere in the XP room is calm and relaxed. There have been some instances where because of the initial shortage of developers, some found it necessary to shoulder more of the load than is normally expected in an XP project. But even in these cases, a sustainable pace was maintained. Once the team expanded, this was no longer an issue.

We now have a thermometer drawn on a white board that is kept up to date measuring the "temperature" of the iteration, i.e., the risk of non-completion of the current tasks. It has only gone high once. Everyone can see at a glance the current state. The daily stand-up meeting ends with resetting the thermometer.

It isn't always easy to ensure that people aren't overworked in this environment since everyone has some responsibility for other projects. The coaches are vigilant about this, and one of the programmers has expressed his thanks for providing him some space in which to do his best work by relieving outside pressures.

The sustainable pace was reinforced because of the high degree of communication among the team both in the Planning Game and during the development cycles. We see the beginning of a self-managing and self-organizing community with a clear acceptance of a sense of shared responsibility.

### **7.1.6 On-site customer**

Even though the XP customer is representing several different stakeholder groups, there is one customer representative on the team who is always available on site. While this has been problematic in many XP projects, for our project this has worked extremely well. The customer quickly learned how to steer the project and how to write appropriate XP stories. Developers regularly approached the customer for answers to questions about a story. The customer also has not had any inhibitions about approaching the developer for information about status and to volunteer additional or clarifying information about a story.

### **7.1.7 Coding standards**

This has not been formalized, but is about to be. Most of the team has worked closely in the past, though not in pairing. Styles are similar and people seem comfortable with the style used. As part of our push to increase pairing and test-first development, we are also trying to get a statement of the coding standard that everyone will agree to.

### **7.1.8 Additional practices not in the original twelve**

#### **Stand-up meetings**

The stand-up is a short (15 minutes) daily meeting of the whole onsite team. This includes the programmers, tracker, coach, manager and customer. Missing are the design group members – information architects, visual designers and their customer representative. They are located at a different site, and their absence is offset by telephone conversations when necessary. While we believe that their presence on site would be beneficial, it has not yet been shown to be problematic.

During these meetings each team member shares experiences and issues. Everyone is kept up to date on the progress. Issues such as infrastructure, tools, XP practices are discussed as necessary. Protracted discussions are continued after the meeting with the appropriate parties as necessary. The stand-up meetings enhance the sense of community and help to further the XP culture development for the team.

#### **Retrospective**

As mentioned elsewhere, we did two "retrospectives" as part of the training process. One was to set a baseline against which this project could later be measured for success, and the other to solidify the Extreme Construction learning. We have also scheduled one day per iteration to use for the Planning Game and for iteration retrospectives. The half-day for retrospectives also provides a bit of schedule slippage, when necessary, in which case the retrospective is forgone for that iteration. This has been infrequently needed, and the frequent mini-retrospectives have been very valuable in pointing out the effects of not maintaining the practices when that has occurred. This has been a great way to reinforce learning. This has also been a time during which the XP values

have been embraced and reinforced – communication, feedback and courage.

### **Pair coaching**

Since this is a first for the organization, two external coaches (among the authors) are part of the team. One or both are available nearly every day: both for Planning Game and retrospectives. We have fallen into a practice quite naturally that is much like pair programming. One of us will take the lead in coaching and will say most of what is needed at the time. The other watches the process, and thinks more strategically about what is occurring and makes comments as necessary. This combining of the micro and macro views of the situation is very valuable.

## **7.2 The practices with issues**

As was previously stated, we intended to introduce and use all twelve practices immediately. However, this did not actually occur. The following describes which practices did not fit well into the existing culture or for which the culture was not ready without more effort and adaptation.

### **7.2.1 Planning game (estimating, velocity)**

While we have faithfully practiced the planning game, with help from the coaches, some aspects have been less than smoothly done. The team has had a hard time picking up the concept of "ideal time" for purposes of estimating, instead lapsing quite frequently to estimating in elapsed time. The customer has also done this. This is especially difficult here since most of the staff has responsibilities for several projects, and one person's typical day is quite different from another's. This is improving, but it has taken time and effort. We are also starting to decouple the idea of a story point and an ideal developer day, making it more abstract, which makes it easier to think about. This may sound backwards, but it is true as the team develops its skill.

There also has been difficulty in determining a velocity each period. The concept of "yesterday's weather," using the story points actually completed in the previous iteration as the basis of velocity in the next, has not worked well. Things have been too volatile. People coming and going, and stories vastly different in kind have hurt us

here, but, again, it seems to be coming together after a few iterations.

It has also taken a while for the team to become comfortable with a velocity that is a small fraction of available time. They now recognize how much they do that is not associated with the task at hand, so this is improving also. However, the team initially did not have sufficient time available to estimate new stories as they were written, making the Planning Game sessions longer than they should have been. This too is improving, especially as the customer has been willing to say that getting estimates was more important than making progress in at least one situation. The team is developing a rhythm, also, which helps.

Another issue of concern is that the customer did not want to set release dates. They were content to view each iteration as a release. This was problematic because it permitted the customer to impose an XP-violating pressure on the developers to move at the customer's pace rather than the developer's pace. We have since convinced the customer as to the wisdom of thinking in terms of multiple iteration release cycles. This should improve the situation.

### **7.2.2 Test-first development and acceptance testing**

This has been the most difficult aspect and leaves us at some risk. We started the project with a team only partially trained (either that or never get started) and without all infrastructure in place. While the programmers have written unit tests, these are usually done after the fact. Until recently, however, there has been no platform on which to share the tests, so each programmer has a few tests that can't effectively be run by others. This is being addressed, and a solid testing architecture for unit tests is being established.

Similarly, we have had, and continue to have, problems in automating acceptance tests. This is partly due to the nature of the project, but also due to both unfamiliarity with the technique and a lack of appropriate testing infrastructure. A new fixture [3] for fitness/fit was developed by one of the coaches to aid in this.

Most important in this difficulty, however, is the lack of experience with test-first development and

some scepticism about it. The programmers are not yet convinced that testing speeds you up, rather than slows you down. We recently had a training session in which this was addressed, both intellectually, and with a hands-on demonstration. It remains in need of improvement.

### 7.2.3 Refactoring

The team has not yet done much refactoring since we are still in the early stages of development, and there hasn't been much need yet. On the other hand, the current lack of sufficient tests will make this especially difficult when it occurs, since the tests are not in place to tell you that you have broken something when you inevitably must refactor.

### 7.2.4 Pair programming

The developer team consists of members with different skills (information architects, visual designers, XML/HTML and Java programmers) and as such, they are not interchangeable. Initially there were two full-time XML/HTML programmers, one full-time Java programmer, two one-third-time Java programmers, two information architects and one visual designer. After the third two-week iteration, two more full-time Java programmers were added and one of the part-time Java programmers became available full time.

There was a strong desire to pair from the beginning. As an example of this, Govind Nishar, one of the Java programmers, read the following quotation from Homer's *Iliad* at one of the daily stand-up meetings. It was discussed briefly and everyone understood and appreciated its import.

*...If some other man would go along with me there would be more comfort in it, and greater confidence. When two go together, one of them at least looks forward to see what is best; a man by himself, though he be careful, still has less mind in him than two, and his wits have less weight. Homer, The Iliad, Book Ten, lines 222-226*

What has made pairing problematic has been the lack of bodies with which to pair. Even though there were three and a fraction programmers working, they had different skill sets and were working on different aspects of the development - XML, HTML pages and Java servlets. There

was minimal pairing by the Java programmers when the part time programmers were available to pair with the full time programmer.

We have stressed the value and importance of pairing all along knowing that it would be difficult to do under the circumstances. Since the Java programmer resources were increased, pairing has increased to the extent that pairing has become less of an issue. In this organization, pairing is used to satisfy the code review process requirement. There is already a sense that this could prove to be a more effective and efficient review process than the traditional code walkthrough sessions.

### 7.2.5 Continuous integration

Since we started without a common infrastructure (everyone was working on their own local workstation), this has been a difficulty. A test server has now been established on which to maintain the integrated system, and the automated tests, but the continuing lack of tests hurts us here again. The customer sees and approves the completed individual stories, but it isn't until late in the iteration that he sees things working together. Everyone agrees that this needs work.

### 7.2.6 Collective ownership

This has not really been much of a difficulty, except that initially we had only one full-time Java programmer, so he became the "owner" of the code. He has no issues with collective ownership; it is just that he knows more about the code than anyone else. The pairing difficulties mentioned above have contributed to this, and as that improves, it is expected that collective code ownership will be successfully in place. In general, as with most of these issues, the developers are willing.

## 8 Management Support

Many large companies such as IBM, have invested heavily into highly disciplined, well documented software engineering practices that can be measured against the Software Engineering Institute's Capability Maturity Model [9] or the even newer Capability Maturity Model Integration (CMMI) [5]. These models offer much to comfort management, and also comfort customers in commercial engagements. Most of the agile software development practices, such as XP, are ori-

ented in a different direction. As a result, individuals who have bought into the philosophy of these models might be faced with a greater challenge to see the value of a methodology such as XP.

Even so, executive support has been good for piloting the methodology. Management in the IT organization and the business or customer organization are optimistic, and are hopeful that XP can help us become more agile in meeting the needs of the business.

Some of the nomenclature used in the XP world however can clash with the business world. For example, in XP, planning is done through something known as “the Planning Game”. While it is possible to sell corporate executives on the merits of XP, such as lower cost of change, improved schedule tracking, frequent deliverables, etc., nomenclature such as “the Planning Game” makes the sell harder. In our case, after observing some management wince when hearing the term in some early management briefings, we started simply referring to it as “the Planning Process”, which everyone seemed perfectly comfortable with. Terms such as “Game” trivialize the seriousness of the activity for those who only hear about it from the periphery, which can be a large number of people.

## 9 The future

There remain several challenges that need to be addressed before larger scale adoption of XP within large corporate environments will be possible.

### 9.1 Distributed Teams

In most large corporate environments the staff are geographically distributed, and are not likely to be in the same location as the customer, as is normally required for Extreme Programming. While there are clear advantages to having everyone together in one location from a convenience and social-dynamic perspective, that is not the reality that large corporations are faced with. They have skilled people in various locations spread across the globe because (i) there are business needs for the distributed population, (ii) in some cases, they can not find people with the skills required in the locations that they prefer, (iii) they have acquired people in other locations through mergers and acquisitions, and (iv) the only way to retain some

of their top talent is to be flexible around certain work-life trade-offs, such as an employee who wants to follow a spouse to another location. Where outsourced application development services are used it is also unlikely that the development team and customer will be co-located. Lack of communication (not necessarily distance) is expensive in XP. While there has been some work done on distributed extreme programming [7] [13], effective methods for distributed pair programming, sharing XP user stories, dealing with off-site customers, trackers or coaches, or conducting daily stand-up meetings, or release planning meetings, are not widely understood or used.

### 9.2 Managing User Stories

Many people in the IT industry are not nearly as experienced or good at applying discipline to managing paper documents as they are electronic documents. Story cards are prone to being lost inside a reference book, taken offsite inadvertently, or ending up on the floor to be swept up by the cleaning folks. While there are some inherent advantages to using the cards, such as the limited amount of text that can fit onto the card, most of those advantages can be replicated in a spreadsheet based user story management application, and can in fact help in estimating velocity trends based on the story estimates, and facilitate the customer’s prioritizing stories for an iteration. This is also important for allowing distributed teams to better work together on the user stories. The challenge is not to let the “helpful tools” get in the way of agility.

### 9.3 IT Governance

Large organizations typically have strict governance models for internal IT projects that try to keep some control (mostly over budgets), but also other items such as synergy with certain IT strategies. The projects are typically driven through a series of executive decision check points where an executive body decides if a project should be funded through the next phase (concept, plan, qualify, deploy, etc.)

It is not completely clear how to integrate XP as a methodology into the current governance systems. This is a critical issue to solve in order to accommodate more pervasive adoption of XP in corporate environments. There likely will need to be adjustments on both sides to make this possible.

On the one hand XP tries to provide feedback to the business stakeholders, so in theory this management should be easy, but on the other hand, the consultative nature of executive decision-making often values process over agility.

## 10 Conclusion

At the beginning of this paper we asked the question: "Does Extreme Programming (XP) make sense as a development methodology in a diverse, multidisciplinary web development environment? We think that the answer is a qualified yes. We have discussed the issues of adopting all of the 12 XP practices from the beginning and the difficulties that developed in trying to do this.

Overall the culture changes necessary to adopt XP appear to be easier to effect than we had expected probably because there is a significant buy-in from the business stakeholders and the developers – they are willing. We are continuing to work hard at getting them ready.

One of the anomalies we are coping with is that while the team is not following all the practices, the methodology appears to be working, although exposing them to some risk. In some cases the team has been following modified practices. The full set of practices is necessary when building a community with an agile culture.

## Acknowledgements

The authors would like to acknowledge the following individuals who all contributed toward the initiation of this Extreme Programming pilot at IBM. Govind Nishar, Matt Ganis, Kapil Gupta, Ning Yan, Katie Dang, Amy Schneider, John Jimenez and Ajay Raina. Also deserving of recognition are Darryl Turner and Vince Ostrosky for their interest and executive support in this endeavour.

## About the Authors

Dr. Fred Grossman has been teaching for more than 30 years and has been involved in software development for 40 years. He is a professor and Program Chair of the Doctor of Professional Studies in Computing in the School of Computer Science and Information Systems of Pace University.

Dr. Joseph Bergin has been a teaching for over 30 years and involved in object-oriented development for 20. He has authored several books and edits a column for educators in SIGPLAN Notices. He has developed several training exercises for agile practitioners as well as some support software for testing.

David Leip is a senior technical staff member at IBM, and the senior manager of IBM's corporate webmaster organisation. He has been active in internet application development for over ten years. David has an MSc in Computer Science from the University of Guelph.

Dr. Susan M. Merritt is professor of computer science and dean of Pace University's School of Computer Science and Information Systems.

Dr Olly Gotel is an assistant professor in the School of Computer Science and Information Systems at Pace University. She has been involved in software development research and practice for over 10 years. Her interests include systems requirements engineering and software process improvement.

## 11 References

- [1] K. Beck, *Extreme Programming Explained Embrace Change*, Addison-Wesley Longman, Inc., 2000.
- [2] K. Beck, M. Fowler, *Planning Extreme Programming*, Addison-Wesley Longman, Inc, 2001.
- [3] J. Bergin, HtmlFixture, <http://fitnesse.org/FitNesse.HtmlFixture>, accessed June 2004.
- [4] J. Bergin, F. Grossman, Extreme Construction, <http://csis.pace.edu/~bergin/extremeconstruction/>, accessed June 2004.
- [5] CMMI Product Team, *Capability Maturity Model Integration (CMMI) Version 1.1: CMMI for Software Engineering (Continuous Representation)*, Technical Report CMU/SEI-2002-TR028 & ESC-TR-2002-028, August 2002.
- [6] N. Kerth, *Project Retrospectives A Handbook for Team Reviews*, Dorset House Publishing, 2001.

- [7] M. Kircher, P. Jain, A. Corsaro, D. Levine, Distributed eXtreme Programming, <http://www.agilealliance.org/articles/articles/DistributedXP.pdf>, accessed June 2004.
- [8] P. Merel, Extreme Hour, <http://c2.com/cgi/wiki?ExtremeHour>, <http://home.san/rr.com/merel/xhour.ppt>, accessed June 2004.
- [9] M. Paulk, B. Curtis, M. B. Chrssis, C. V. Weber, *Capability Maturity Model for Software Version 1.1*, Technical Report CMU/SEI-93-TR-024 & ESC-TR-93-177, February 1993.
- [10] H. Robinson, H. Sharp, XP Culture: Why the twelve practices both are and are not the most significant thing, *Proceedings of the Agile Development Conference*, IEEE Computer Society, 2003.
- [11] M. Spayd, Evolving Agile in the Enterprise: Implementing XP on a Grand Scale, *Proceedings of the Agile Development Conference*, IEEE Computer Society, 2003.
- [12] D. Wallace, I. Raggett, J. Aufgang, *Extreme Programming for Web Projects*, Addison-Wesley Longman, Inc, 2003.
- [13] N. Wallace, P. Baily, N. Ashworth, Managing XP with Multiple or Remote Customers, <http://www.agilealliance.org/articles/articles/Wallace-Bailey--ManagingXPwithMultipleorRemoteCustomers.pdf>, accessed June 2004.