Challenges for Stakeholders in Adopting XP

Peter Bailey
Synop Pty Ltd
PO Box 1024
Artarmon NSW 1570 Australia
+61 2 9411 8744
peter@synop.com

Neil Ashworth
Synop Pty Ltd
neil@synop.com

Nathan Wallace
Synop Pty Ltd
nathan@synop.com

Abstract
Adopting XP is a challenge both to incorporate new practices into an organisation's software development methodology and for the individuals concerned. We identify several groups of stakeholders who must typically adjust to change, which aspects of XP are often challenging, and why this is so.

Keywords
Stakeholders, challenges, adoption of XP experience

INTRODUCTION

Adopting a new software development process, especially Extreme Programming (XP) with its strong emphasis on people rather than process, is always a challenge. There are many contributing factors to the difficulties involved. Classically, this can be considered an exercise in change management [4]. In this paper, we examine the twelve standard XP practices and how they challenge the main stakeholder groups in software development. We draw on our experience in introducing XP within three quite different companies to distil some common issues that arise.

RELATED WORK

A number of people have written about how to adopt XP. Beck in [1] following suggestions from D. Wells advises picking a worst aspect of current practice, address it with XP techniques, and repeat until all worst problems are solved. Wells and Buckley [13] provide an experience report on the process of adopting XP one practice at a time within a company. Fowler argues attempting to start (at least on a new project) with adopting all the practices "by the book", before adapting any of them [7]. Collins and Miller write on the need for adapting a new software process to suit local conditions [3]. They nominate several alternatives to follow, including doing it "by the book". Sommerlad [10] discusses how itopia used XP for an Internet server infrastructure product for external customers, and the difficulties in prioritizing the needs of multiple customers. Talbott and Miller discuss the challenges in persuading the "Gold Owner" role to support the adoption of XP [11]. Gittins *et al* provide a qualitative research study of the adoption of XP in a medium sized business with reference to the twelve standard XP practices [8]. Our paper differs from all of these prior works by specifically focusing on the different stakeholders who are involved in the process of adoption, and how XP practices impact on them.

STAKEHOLDERS

In the literature on XP (see [1, 2] as a starting point), there is considerable emphasis on the two most significant stakeholders in software development, programmers and customers. Our experience suggests that there are often four other stakeholder groups who affect or are affected by the adoption of a new software development process.

These additional stakeholders are:

- Quality Assurance (QA) – these people perform activities relating to system testing, code quality, defect analysis and so on.
- Documenters – various types of documentation are written for a software product, ranging from technical documentation to user manuals and marketing material.
- Project managers – have overriding responsibility for the delivery of a project, and typically act as the central link between the programmers and management.
- Management – overall responsibility for a company's performance, with reporting duties to directors and/or shareholders and investors.

Stakeholders vs Roles

In [1], responsibilities within the XP process are identified by role. Beck makes it clear that individuals may carry out one or more roles, and that a single role may be held by several people. To be similarly clear, the stakeholders we identify are simply groups of responsibilities that are carried by one or more individuals. In larger companies, individuals will often only belong to one stakeholder group. As XP is adopted, these responsibilities may change for some of the people involved. We use the distinction to clearly identify the responsibility groupings before adoption (stakeholders) and during and after adoption (roles).

Kinds of Development Organisations

In our experience, development organisations can have significantly different nature. One was a startup company, with no established development methodology. Another was a small development company, with less than 5 developers. The last was larger, and had a team of developers, testers, analysts, and managers.

Kinds of Users

XP talks about a customer role, but we believe it is also instructive to consider the final user of the software. In some senses, the customer role is a proxy for the perceived needs of the user(s). The XP practices require a

customer representative role within the team capable of making decisions about what features are built. The customer role is also responsible for representing the user's needs in a way that makes business sense. Sommerland argues in [10] that in practice this is not always possible depending on the kind of customer(s). They built a single product, with multiple external customers, so changes for one customer had to be balanced against changes for others. In our experience, we have seen at least three kinds of user. The first, with three real user organisations, but represented by a single on-site and available customer representative, was the most effective. Another was a single user organisation, located in another country. In this case, the development team had to play the customer role internally, using prioritisation advice from the user organisation. The final example was where there were no existing users at all, and so business decisions as a customer had to be made internally.

XP PRACTICES

Beck argues in [1] that the 12 standard practices of XP are interdependent, that the weaknesses of one practice are made up for by the strengths of the others. In fact, only a few of the practices are usually considered contentious by a new adoptee of XP in our experience. Some practices are relatively novel, but the rest are generally regarded as best practice for rapid development. McConnell's study [9] of rapid development practices identifies the benefits and risks of these. In [7] Fowler makes a case for why all practices should be adopted if possible when starting out with XP, before adapting to the local conditions. What follows is our observation of which of the practices challenge the different stakeholders when first adopting them.

Pair Programming

Pair programming is where two programmers work together to produce code. Pair programming is most strongly resisted by two stakeholders: programmers and management. Programmers typically resist because most have no experience of programming other than on their own. Some programmers enthusiastically adopt it after trying it, while others remain resistant. The challenge for project managers is to find ways to encourage pair programming, and to help find appropriate balances in the levels of experience to make it a satisfying experience for both people in a pair. There are many papers written on pair programming (see [5] for a start), so we will not consider the issues for programmers further. Management are likely to resist pair programming, but for different reasons. The standard reservation is that pair programming is expected to reduce productivity, and that programmers are expensive. The only ways to address this perception are by the body of published results on benefits of pair programming and by measuring its effects within the organization. QA people are likely to appreciate the benefits of pair programming due to the decline in defect rate that it tends to produce. There is low to no affect on customers or documenters. (In subsequent practices, we will not comment if we believe there is little or no affect on a stakeholder group.)

Collective Ownership

Collective ownership allows any programmer to change any code in the system. Collective ownership is often a challenge, especially for programmers, and to a lesser degree project managers, customers and management. Programmers conditioned to programming on their own find it difficult to accept that other people can modify their code. There is often a strong sense of ownership to written code that must be lost before collective ownership can be accepted. Often programmers need to understand how the whole system fits together before being confident to change it. This hesitation is particularly true for programmers who arrive partway through a project. The challenge facing project managers, management and customers is the loss of an identifiable single point of responsibility for any part of the system. In other processes, there is usually a single programmer who can be identified as the person responsible to talk to about some part of the system. With collective ownership, in theory they should be able to talk to any programmer at all. In practice, there will still only be a few programmers who have constructed any particular part of the system, but identifying who they are is more complex.

Testing

Testing in XP insists on a test-first strategy where unit tests are written before coding begins. Programmers again are usually the most reluctant to adopt this approach, as it requires a change to their habitual work practices. On projects which adopt XP part way through, there is even more resistance as there are already large bodies of code without any tests at all. Programmers are often reluctant to create tests for old bits of code. The time taken to write the tests is begrudged as it apparently does not contribute to producing code. Lastly, there is overhead in establishing the automated unit testing framework and learning how to write effective unit tests. Programmers tend to slip back into old habits and put off writing tests until the completion of a user story. Since this will often come near the end of an iteration, there is schedule pressure to complete the story, and tests are less effective. QA people are substantially affected by the XP testing regime. As programmers take on more QA responsibility, there is less classic QA work required. In the early stages, experienced QA people can play a valuable role in educating programmers about how to write good quality unit tests. They can also take responsibility for establishing the unit testing framework.

Project managers, management and customers are generally supportive of tests as it provides increased evidence that the system is working. As McConnell identifies [9], testing is used as an early warning indicator – if there is a problem, it is best discovered as early as possible so that there is time to fix it.

On-Site Customer

An on-site customer acts to represent and make clear decisions of the customer's business needs. Obviously, the biggest challenge is for customers themselves, who now take an active part in the process of software devel-

opment. The key change for customers is that they must make hard decisions about what goes in or out of the product. They must also learn how to determine whether user stories are accepted, which means they must learn to formulate acceptance tests. QA people are affected as they are the best people to assist the customer in developing formal acceptance tests or in translating the customer's specification of what constitutes acceptance of a user story into an automated (if possible) functional test(s). Programmers must learn to interact with the customer to discover what is required in each user story. Programming has often been taught or practiced as a solitary activity, so interpersonal communication skills may need to be learnt. Project managers and management may be faced by two challenges. XP allows little room for disguising the true rate of progress. An on-site customer is rapidly aware of exactly how fast development proceeds, and the planning game makes them aware of how it is likely to proceed in future. If this rate is not sufficient to meet deadlines, customers are prone to fall back on old habits and demand increased development speed by various flawed practices. Project managers and management are responsible for educating the customer on their rights in prioritising what is developed when, and preventing the return of bad old ways. XP tends to give frequent progress indications which let management demonstrate a working system is deliverable. Documenters may also find that an on-site customer alters their traditional activities. Much of the system (in terms of the details of how user stories are interpreted) is defined iteratively in interactions between the customer and programmers. Working to capture these details as they are discussed may provide more effective documentation of the system from a user perspective, at the cost of increased and/or incremental workload.

Metaphor

A metaphor provides a simple analogy for what the system should be like or do. Programmers typically are just as, if not more, comfortable working from a metaphor as they are working from a 500 page functional requirements document or an architecture diagram of "big boxes and connections" [1]. The same problems remain – how to translate the ideas into code. Customers too seem happy to accept working from metaphors. There is a degree of unfamiliarity, and sometimes a desire to see architecture diagrams. However, since many customers are not technically inclined, the best way to reassure people is with working prototypes. The stakeholders most affected by a switch to using a metaphor are the testers and documenters. In both cases, there are no longer design documents which in principle define how the system should operate. Thus they are required to spend more time discovering exactly how it does work, which involves more communication skills. Project managers are affected, as now they must explain initially in terms of the metaphor. Management is likely to be uncomfortable about the lack of details. XP tends to be vaguer than traditional approaches about how something is going to work in the early stages of a project, but then provide far more statistics and observable progress indicators as iterations and releases are completed. Unfortunately, at the adoption stage, management is left with less information than typical, which makes the risk factor appear higher.

The Planning Game

The planning game determines what stories get built each iteration and release. The primary parties involved are customers and programmers. The main challenges facing them when adopting XP are learning how to write good user stories, and how to estimate them respectively. Also both customers and programmers have to learn the boundaries of their responsibilities and rights. Again, this becomes a learning exercise in communication. Programmers particularly may initially find the planning game a waste of time when they could have been coding. Naturally it pays off in a better understanding of the customer's real requirements, and how the different stories fit together. There are several challenges for the project manager. First, they must adjust to a completely new way of deciding what schedule is followed for development as the customer now gets to choose, and adjust to using user stories as the unit of development. Second, they must let go of some of the responsibility in deciding what each programmer is working on. In early adoption stages, we have found that it is still helpful for the project manager to play a part in guiding each programmer towards particular user stories which have a natural fit with the programmer's abilities. Third, they must accept that the half day or day spent in the planning game is a necessary cost in the schedule of each iteration. Fourth, there is increased overhead in preparing for each planning game activity, both for iterations and releases.

Refactoring

Refactoring is the art of adjusting the code to make it simpler. Programmers unfamiliar with the technique may find this a challenge, as it is not an aspect of programming often taught, and is often best done by practice or observation. Project managers and managers are not substantially affected by the practice, other than needing to accept that refactoring does take time, and this must be allowed for in the schedule. Documenters and QA may both be affected by refactoring, as the process of refactoring may involve changes to existing code interfaces. Thus, depending on how much work has been done already around the code being refactored, they may need to redo some of it. Documenters should also consider refactoring their own documents frequently, to avoid unnecessary duplication.

Simple Design

Simple design is doing the simplest thing possible while keeping the rest of the system working. Programmers often find this a challenge, due to their fascination with new technology. McConnell [9] refers to it as "gold-plating" and nominates minimal specification and requirements scrubbing as practices for keeping design simple. Of course in XP, it is up to the customer to decide on which requirements can be scrubbed. The challenge for the customer is in accepting that their vision of the system almost always contains unnecessary or overly

complex features. The interactions with programmers on estimating a particular feature's likely development effort and the process of learning to sacrifice features to get a story delivered are two important early lessons for customers. The challenge for project managers is to help constrain the programmers' natural tendencies to develop more complex code than is necessary.

40-Hour Week

A 40-hour week dictates that programmers do not work more than 40 hours each week writing code. Programmers have little resistance to a 40-hour week, other than when they are highly motivated, in which case they may well wish to work a little longer. Customers, project managers, and management have the biggest challenge. In the face of schedule pressure, almost always the first response is to suggest programmers should work longer hours. There is a substantial set of published studies on the number of hours worked and actual productivity. McConnell [9] quotes figures and reports from numerous sources in support of his assertion that anything other than voluntary overtime has an immediate and dramatic negative effect on productivity.

Coding Standards

Coding standards require that all code be formatted and written in a common way, with meaningful naming standards. Programmers are the people most challenged by this practice if they have not previously had to work with other people or read other people's code. The challenge for the project manager is to find a coding standard acceptable to the programmer group, and to find ways to enforce it if required.

Small Releases

Small releases require the system be driven into a working and most business useful form every two to three months. The challenge for project managers is to allocate enough resources to accomplish the release process, which is more work than just having a working system. All stakeholders usually end up having to spend some effort working on the release specifically, rather than their normal activities. All stakeholders gain a sense of satisfaction in seeing a working release – it provides a tangible sense of accomplishment.

Continuous Integration

As with small releases, continuous integration is a fairly non-controversial practice. It is essential for ensuring that the system is always working, and everyone likes a working system. The main challenge arising from it is really for the project manager, to ensure that there are sufficient resources to allow integration by all programmers on a regular basis without undue delays. There is also the need to make sure that the source code control system is managed to allow testers access to versions which are stable for them to test against.

CONCLUSIONS

Of the twelve standard XP practices, only a subset actually tend to provide challenges for several of the six stakeholder groups we identify. Of the rest, many of them are regarded as best practice in achieving rapid development, and impose relatively small challenges to only one or two of the stakeholders. The common theme to adopting all the practices is that one or more changes are involved for the stakeholders. As such, we believe it is valuable to consider active change management practices when adopting XP, with an awareness of the systems of stakeholders who are going to be affected most. Programmers are perhaps most affected by adopting XP, and one of the key skills they require to accommodate the changes is communication – with each other, with the QA and documenter people, with their project managers (and management if required), and most importantly with the customer. As such, it reinforces Denning and Dunham's assertions in [6] that value skills are a vital core to IT professionals.

REFERENCES

1. K. Beck, *Extreme Programming Explained*, Addison-Wesley, (2000).
2. K. Beck and M. Fowler, *Planning Extreme Programming*, Addison-Wesley, (2001).
3. C. Collins and R. Miller, "Adaptation: XP Style", in *Proceedings of XP2001*, (May 2001), 54-57.
4. *Change Management Resource Library*, online at http://www.change-management.org.
5. A. Cockburn and L. Williams, "The Costs and Benefits of Pair Programming", in *Extreme Programming Explained*, Addison-Wesley, (May 2001), 223-243.
6. P. J. Denning and R. Durham, "The Core of the Third-Wave Professional", *Communications of the ACM*, Vol. 44, No. 11, (Nov 2001), 21-25
7. M. Fowler, *Variations on a Theme of XP*, online at http://martinfowler.com/articles/.
8. R. Gittins, S. Hope, and I. Williams, "Qualitative Experiences of XP in a Medium Sized Business", in *Proceedings of XP2001*, (May 2001), 122-126.
9. S. McConnell, *Rapid Development*, Microsoft Press, (1996).
10. P. Sommerlad, "Adopting XP", Ch. 24 in *Extreme Programming Explained*, Addison-Wesley, (May 2001), 423-432.
11. N. Talbott and R. W. Miller, "Selling XP to the People who Buy", in *Proceedings of XP2001*, (May 2001), 72-74.
12. D. Wells and T. Buckley, "The VCAPS Project: An Example of Transitioning to XP", in *Extreme Programming Explained*, Addison-Wesley. (May 2001), 399-422.