

Deployment of Agile Development Methodologies

A Whitepaper

© Business Agility, 2004

Date: 21/04/04
Version: 1.0
Authors: Robert Gittins and Dr Julian Bass



EMAIL: info@business-agility.com

WEB: www.business-agility.com

Copyright Notice

This document may not, in whole or in part, be copied, photocopied, reproduced, translated, or reduced to any electronic medium or machine-readable form without prior consent in writing from Business Agility.

Trademark Notice

All product and company names mentioned are the property of their respective owners and are mentioned for identification purposes only.

Contact

For more information about this document, contact:

Business Agility Limited
The Spirella Building
Bridge Road
Letchworth Garden City
Hertfordshire
SG6 4ET
United Kingdom
Telephone: +44 (0)1462 476160
Email: info@business-agility.com

Contents

Deployment of Agile Development Methodologies	1
A Whitepaper	1
1. Abstract.....	4
2. The Current Software Development Climate	5
2.1 Scope and Project Phase Relationships	5
2.2 Methodology Costs and Benefits.....	6
2.3 Towards Agile Software Development Methods.....	7
3. Dynamic Systems Development Method (DSDM)	9
4. Unified Software Development Process	10
5. Extreme Programming (XP)	12
5.1 The Core XP Practices:	13
5.1.1 The Planning Game	13
5.1.2 Small Releases	13
5.1.3 Metaphor	13
5.1.4 Simple Design	13
5.1.5 Continuous Testing	13
5.1.6 Refactoring.....	13
5.1.7 Pair Programming	13
5.1.8 Collective Code Ownership	14
5.1.9 Continuous Integration	14
5.1.10 40 Hour Week	14
5.1.11 On Site Customer.....	14
5.1.12 Coding Standards.....	14
5.2 Benefits of Extreme Programming for Large Enterprises	15
5.3 Challenges When Using XP	15
6. Comparison of DSDM, RUP and XP.....	18
7.1 Proof of Concept.....	19
7.2 Hybrid Methodologies	19
7.3 Strategic Agile Adoption.....	20
8. Conclusions.....	21
9. References.....	22
About The Authors	23

1. Abstract

Agile software development methodologies have attracted considerable interest. These methods offer both improved software product quality and reduced development times. The key features of agile development methodologies in general are presented with a brief overview of Dynamic Systems Development Method and the Unified Process. The values, practices, benefits and some challenges in the adoption of Extreme Programming (XP) are described in more detail.

The whitepaper discusses the adoption of XP in smaller software development organisations. Some first-hand experiences of the adoption process are presented. A comparison of DSDM, RUP and XP is given along with some shortcomings that require attention. Finally, a discussion of the adoption of XP in large enterprises is considered.

2. The Current Software Development Climate

2.1 Scope and Project Phase Relationships

Most Global 500 enterprises have established a well-defined software development methodology. In the financial services sector, our experience suggests, most large enterprises use a project management methodology to encourage best practice. The most commonly used methodologies implicitly (or explicitly) assume a waterfall development model.

Inherent in waterfall models, (Figure 1), is a simple sequential relationship between one project phase and the next. The whole software development project cascades sequentially through the project phases. There are many variations of the waterfall development model, distinctive by their phases, which can change in number and name. Also the degree of iteration and/or feedback between phases varies. But, in general, all projects using a waterfall model follow a similar pattern of:-

- Requirements analysis,
- Requirements specification,
- Design,
- Software construction (or low-level design and coding),
- Testing,
- Maintenance and Enhancement.

This type of project management methodology has numerous attractions. There are clear demarcations between the different activities that must be conducted during the project. Management reviews and checks, sometimes with contractual requirements can be inserted between the phases. The waterfall model also brings considerable transparency to a development process that was previously ad-hoc and error prone.

However, the waterfall approach is both inflexible and unrealistic, real projects rarely followed the prescribed phases in an orderly manner, and customers were frustrated by being bound by their early requirements. Customers needed the flexibility to periodically interject and change requirements according to evolving circumstances. Also the completed project version was often only seen late in the development process. Mistakes and misunderstood requirements emerged late in the project making it necessary to repeat some or all of the phases.

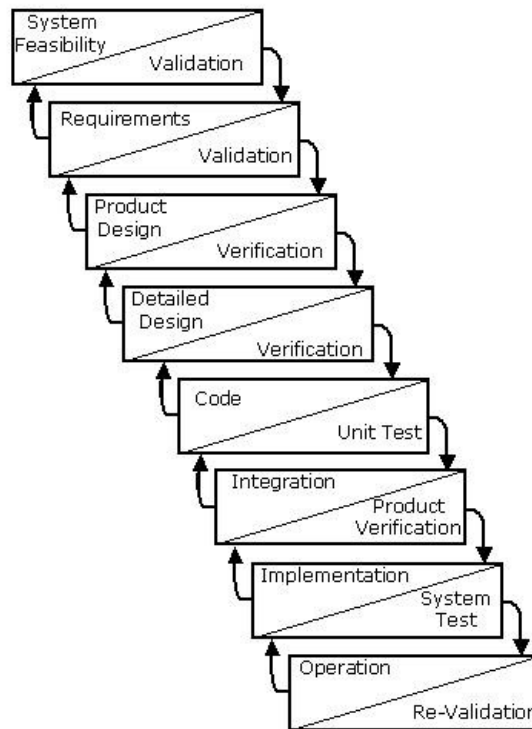


Figure 1: Waterfall Lifecycle Model (from [1])

In addition, evidence shows that it is easy to introduce errors and misunderstandings during the requirements specification, design and construction phases. It is difficult to identify and correct these errors until after initial testing has been completed. Attempting to deliver an entire software product in one linear process introduces a long time delay in identifying and correcting this type of error, and the longer the delay the higher the impact.

Customer stakeholders cannot wait months or even years to receive a software product whose requirements had not kept pace with the market. Business is conducted in a rapidly changing environment; this change must be accommodated during the development process of software products.

2.2 Methodology Costs and Benefits

It has been established that investing more resources into methodology and compliance does not always bring commensurate benefits in terms of resulting product quality. It is recognised that there are diminishing returns, where staff are distracted from development effort by the need to prepare deliverables that add little or no value the resulting product.

Figure 2 shows how modest investments into a lightweight methodology can bring dramatic benefits in terms of reliable product delivery dates and product quality. The diagram also shows that increasing levels of investment in a sophisticated methodology bring more modest benefit improvements. The exact scaling and shape of curves A or B in Figure 2 varies from project to project.

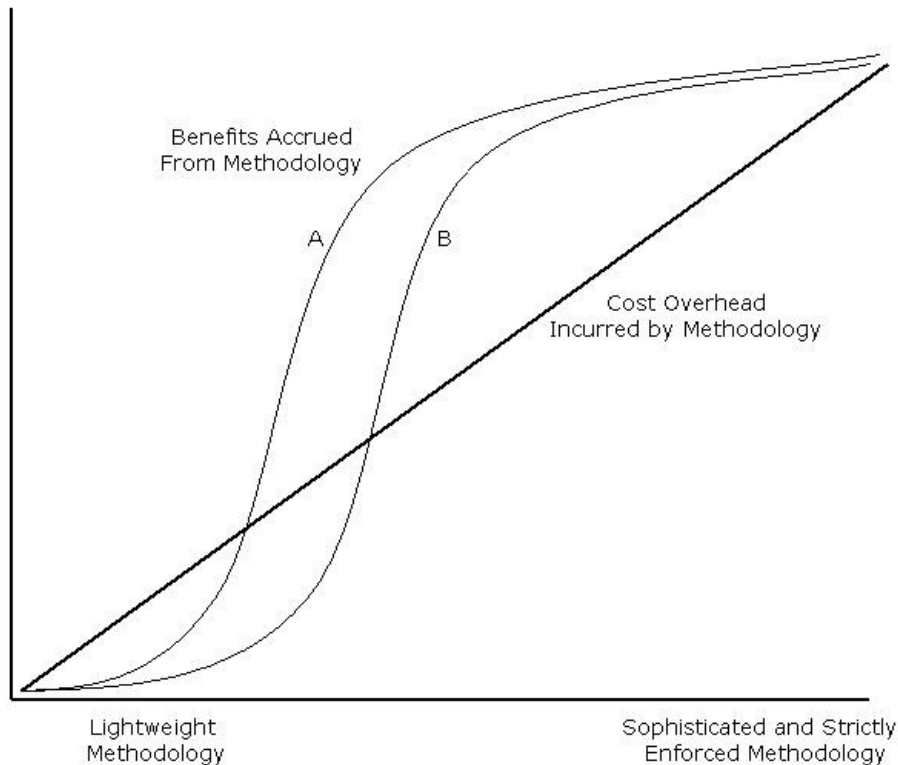


Figure 2: Simple Graph of Methodology Costs and Benefits

In sectors such as transportation, the safety-critical nature of the software product may warrant a significantly higher investment in methodology. Such sectors require that methodologies are extended with additional activities and deliverables to minimise risks. For example, independent teams may be required to build simulation models of software that has already been constructed. The simulation models can in turn be used to test scenarios and demonstrate correct interpretation of requirements.

In sectors such as financial services and telecommunications, the cost of such additional activities cannot be justified. The challenge is to find the right balance between investment in methodology and quality assurance, and the consequential product quality-benefits.

2.3 Towards Agile Software Development Methods

During the 1980s and 1990s several important new ideas emerged that have had a profound impact on software development methods. The ideas of explicitly assessing risk and incremental development within the development process gained influence. Performing a risk assessment at each project phase was proposed in the seminal work by Barry Boehm [1]. This approach recognised that different projects had different priorities and challenges. It now seems intuitively sensible that there will be distinctive risks for a given project at each phase in the development process. A further innovation that emerged was the idea of dividing the development of a software product into a series of increments. Each increment offers a carefully selected series of enhancements over previous increments. The incremental approach offers opportunities to learn from experience and mistakes correcting errors from one increment to the next, improving the quality of the final, delivered software

product. Further there is a greater opportunity to accommodate a changing business context during the life of the development project.

Various so-called Agile methodologies have emerged independently since 1995. In 2001 it was recognised by practitioners that methodologies such as Dynamic Systems Development Methodology (DSDM), Feature Driven Development (FDD), Crystal, Adaptive Software Development (ASD), SCRUM and Extreme Programming (XP) shared many of the same characteristics, and in consequence the Agile Alliance [2] was founded for methodologies which shared the same lightweight and flexible features [3].

Agile methodologies are recognised by the way they balance 'no-process' with 'too-much-process' to provide a better platform for change. Too much process with heavy documentation tends to become predictive and Agile methods with their emphasis on low documentation aim to be adaptive and flexible to change by reduced process and therefore the cost of change

3. Dynamic Systems Development Method (DSDM)

DSDM evolved in 1993 as the answer to an unstructured Rapid Application Development (RAD) community of developers (who opposed the slow and cumbersome waterfall paradigm), and is important as it provides a high-level method describing best practice and sound software development project principles.

DSDM resembles more a framework than a methodology, it has five basic steps preceded by a Pre-project phase and followed by a Post-project phase [4]. The five true project steps are: Feasibility Study, Business Plan, Functional Model Iteration, Design and Build Iteration and Implementation. The framework of activities are supported by nine underlying principles:

1. Active user Involvement
2. Empowered teams
3. Frequent product delivery
4. Fitness for business purpose
5. Iterative and incremental development
6. Reversible changes
7. Baseline high-level requirements
8. Integrated testing throughout the life cycle
9. Global collaboration between stakeholders

The system for handling flexibility of requirements in DSDM is called the 'Timebox'. Completion dates provide an overall timebox for the work to be carried out. DSDM refines the concept of timeboxing using shorter nested timeboxes of between two to six weeks within the temporal framework.

Timeboxes have three phases:

1. Investigation – are the team going in the right direction?
2. Refinement – based on feedback from the investigation
3. Consolidation – Tying- up the loose ends

Prioritisation of requirements through the timeboxes are checked and perhaps reassigned using the 'MoSCoW Rules'. The rules are used to guide decision- making about team work load activity throughout the project.

Must Haves - fundamental to the projects success

o

Should Haves - important but the projects success does not rely on these

Could Haves - can easily be left out without impacting on the project

o

Won't Have - can be left out this time round and done at a later date

Prioritisation is developed so that the deliverables are completed within the given timeframe.

4. Unified Software Development Process

The Unified Software Development Process (also known as the Rational Unified Process, RUP or simply the Unified Process) divides development cycles into four phases; Inception, Elaboration, Construction and Transition. The process is use-case driven, architecture centric, iterative and incremental [5].

The functional requirements of the system are derived in terms of the responses that should properly be made to interactions with outside users (human or otherwise). Use-cases provide the mechanism for defining these interactions in an orderly manner. A complete collection of use-cases, known as a use-case model, defines all the required functional behaviour of the system.

The process is architecture-centric in the sense that models of software are built to explore system structure and behaviour from different perspectives. The architecture is defined, in part, by the needs of the enterprise and also non-functional requirements of the system. Architecture defines such aspects of the system as computer and network topologies, integration with external heritage systems, and the software technology stack (operating systems, application servers etc.).

Increments in RUP refer to the growth in completeness of the product. Incremental development is cyclic adding functionality in an orderly and managed fashion. Specific 'Workflows' are used to express a sequence of activities conducted, to some extent, in parallel during an increment.

There are five core-process workflows and three core-support workflows:

RUP Core-Process Workflows

1. Requirements
2. Analysis
3. Design
4. Implementation
5. Test

RUP Core-Supporting Workflows

1. Configuration and Change Management
2. Project Management
3. Environment

The core workflows are reflected by multiple iterations in each of the four phases of the development cycles. Workflow's core-processes bear similarities to the traditional Waterfall Model; however RUP's development cycles have a multiple iterative structure within each phase rather than the single-pass life-cycle of the waterfall model.

One of the problems with RUP is that although it was developed as an iterative development process, it is fundamentally a process framework that can be adapted to accommodate a wide variety of processes from Agile, so-called flexible processes, to heavyweight and traditional waterfall methods. It's adaptability is seen as one of RUP's weaknesses, as Martin Fowler maintains [6]

"...since it [RUP] can be anything, it ends up being nothing. ... it all depends how you tailor it in your environment."

RUP can therefore be used in a traditional waterfall style or in an Agile manner, that is to say it can be a lightweight or flexible process, or a heavyweight process.

5. Extreme Programming (XP)

Extreme Programming was conceived by Kent Beck [7, 8, 9 and 10] and others to address the specific needs of software development, in the face of vague and changing requirements. Extreme Programming is a new approach to requirements gathering that involves customer, manager and development team collaboration at 'Planning Games'. Planning games develop planned short releases; short releases are important in lowering negative development impact, which reduces project risk. Work is done first, on 'stories' that can give the most business value. Developers work on simple tasks and quickly learn to accurately estimate duration of planned simple tasks. Realistic time estimates provided by developers improve customer confidence in planned release dates.

Customers sitting with the team, update plans based upon the realistic estimates provided. In addition, customers sitting with the team spot both problems and opportunities, allowing for timely improvement. Customers have growing confidence with the team, better communications and improved negotiations that together promote the development of better products, within a shorter and more predictable time frame.

XP was founded upon sets of values, variables, rules and practices. The fundamental values that define the XP methodology are shown in Table 1.

The Four Values	Important Aspects
Communication	Person to person communication breakdown causes problems. Timely questions are critical. Communication permeates throughout XP. A COACH monitors problems and opportunities and provides local leadership. A TRACKER monitors task progress and is a support facilitator.
Simplicity	What is the simplest thing that could possibly work? The simpler the system is, the less there is to communicate.
Feedback	Feedback is the treatment for optimism. More feedback breeds better communication. Tests provide feedback, which in turn focuses on how simple the system can be.
Courage	To go like hell. To throw away code. To get over the 'hill climbing algorithm' problem by taking a longer leap. Courage supports simplicity. Crazy ideas can slash complexity. Concrete feedback supports courage by giving confidence.

Table 1: Values upon which XP is founded

5.1 The Core XP Practices:

5.1.1 The Planning Game

A new approach to requirements gathering that involves customer, manager and development team collaboration. Planning games develop planned short releases - hence there is only low impact if mistakes are found. Work is done first, on stories that can give the most business value. Developers work on simple tasks. Developers quickly learn to accurately estimate duration of planned simple tasks. Realistic time estimates provided by developers improve customer confidence in planned release dates.

5.1.2 Small Releases

Short releases provide early feedback to enable changes to be applied. The longer it takes to release to the users, the less time there is to apply changes and the greater the impact of problems.

5.1.3 Metaphor

Metaphors are a shared vision or common understanding that reveals some additional information about the system. Metaphors provide an abstract view, usually a simple shared story, which conveys a mental image, universally understood by the stakeholders, that facilitates cross-discipline discussions.

5.1.4 Simple Design

Simple designs facilitate easier testing and therefore assist a testing first strategy, Simple designs take less time to complete than complex ones and those designs can be estimated easier than complex ones. XP supports the benefits of replacing a complex thing with something simple and to always do the simplest thing that could possibly work. XP supporters say "Never add functionality early – You Aren't Going to Need It!"

5.1.5 Continuous Testing

Both customers and programmers write tests. Tests are automated in XP, or they are less efficient. Tests are written before coding – the programmers then have a clearer vision of objectives. Traditional methods (post-coding tests) are code-oriented; testing in this manner is analogous to having tunnel vision. Initially test-first seems to slow productivity but is rewarded by increased efficiency and less debugging. Function tests (acceptance tests), are written by the customers to confirm business objectives have been met, whereas Unit-tests are written by programmers to confirm their programs work to their understanding, or vision, of the program.

5.1.6 Refactoring

Software code is often used long after it has ceased to be efficiently maintainable. Programmers and managers are often confronted with "*It works, so don't tinker with it!*" attitudes, symptomatic of legacy code problems. XP states that refactoring through the entire project cycle saves time and improves quality. Merciless refactoring keeps the design simple, and avoids clutter and complexity. Refactoring gives cleaner code that is more easily understood.

5.1.7 Pair Programming

In pair programming, code is produced by two people working at a single machine. One person thinks tactically and types code, while the other thinks strategically. Solo programmers are more likely to over-design and make mistakes than pair programmers. Traditional lone-programmers, often referred to as 'hero developers'

are rarely good communicators and don't take kindly to pair programming. Pair programming however is a learnable skill. Pair programming supports collective code ownership, where code is no longer in the hands of a guru or hero developer, it resides amongst the team who have supported the code production during pairing activity.

5.1.8 Collective Code Ownership

Collective code ownership practice means that anyone can change any piece of code in the system at any time. This translates into a situation where complex code doesn't live long in a system that anyone can review. Collective code ownership prevents the team having to live with stupid code. Testing supports collective code ownership as it prevents poor code entering the system. Knowing that others will be looking at your code provides an incentive to make sure it is correct and respectable before it hits the system. Collective code ownership spreads knowledge of the system around the developer team.

5.1.9 Continuous Integration

Tests are automated and can be run quickly. You know your code works and you haven't broken anything. Integration in XP should take place every few hours, preferably no less often than a day. Pair programming promotes confidence to integrate. Refactoring reduces the chances of conflict. The cost of continuous integration can be offset to some extent by the reduction in risk to the project. Continuous integration provides natural breaks and rapid feedback on project development

5.1.10 40 Hour Week

Overtime is symptomatic of a project having a serious problem. Planning games produce simple tasks that guide the delivery of business value in a realistic time. Keeping to the 40 hour week reduces stress and keeps the team fresh and energetic. This in turn supports clear and imaginative thinking.

5.1.11 On Site Customer

The customer is always available for collaboration and guidance. Particularly customers write function tests and are heavily involved in planning games and the development of short release cycles. The customer sitting with the team, updates plans based upon the realistic estimates negotiated. Customers spot problems and opportunities, allowing timely improvement. Customers develop increased confidence with the team, better communication and negotiating positions that promote confidence in product quality and from shorter times.

Customers on site reduce the risk of project degradation by responding quickly to problems as they arise and steering the project away from adverse developing business situations, and into business opportunities. The problem is persuading a client to commit to providing a 'customer' on site. The problem is often overcome by having a proxy customer, a specialised role adopted by a member of the host company.

5.1.12 Coding Standards

All code is written in accordance with rules emphasizing communication through the code. Pair programmers swapping frequently, reviewing code freely, and refactoring each other's code, must conform to a common set of rules – coding standards. Coding standards must be adopted by the whole team voluntarily.

5.2 Benefits of Extreme Programming for Large Enterprises

XP works best in small to medium sized businesses, teams typically between 5 to 10 developers and become less effective as their size reaches 20 and beyond. XP uses technologies such as objects, patterns and relational databases through short iterative cycles. XP makes a distinction between business interest decisions and those made by project stakeholders; testing-first - and keeping tests running at all times with integration and testing for the whole system daily, and support for pair-programming, collective code ownership and simple design evolution, leading to flexibility and removal of unneeded complexity. In addition, XP advocates simple but quick system production giving early business value, leading to organised change and growth.

XP advocates the rapid early deployment of important functionality, promises better customer relations, early delivery of business value, lower defects, and the flexibility to change in timely response to emerging opportunities. Software development has long been characterised as taking place in an environment of vague and rapidly changing requirements, and the creators see XP as an antidote to this dilemma by providing a flexible approach to customer's changing requirements and by its freedom from heavy documentation. In other words, XP is the antithesis of plan-driven development characterised by heavy documentation and low flexibility characteristic in development that follows a rigid system of requirement documents, prepared by customers who are unsure of (or find it difficult to express) their needs. In these traditional systems the customer's tentative requirements are then independently translated and fixed by developers, invariably compromising business-value, which is only discovered late in the project.

XP aims to produce better customer relations to facilitate a clearer understanding of requirements. Improved customer communications provide confidence to produce better quality code within predictable time frameworks, while remaining flexible to change.

If code fails or doesn't come up to customer expectations and is rejected short iterations mean that there is only low impact on code production.

Distribution of knowledge throughout the team via stand-up meetings, pair programming practices and collective code ownership, mean that code is considerably less vulnerable if key personnel leave. This also reduces the threat from legacy code which is no longer being manufactured and is stopped from entering the system to threaten future maintenance and development costs.

XP testing practices mean that there is more confidence in the quality of code entering the system, and with continuous integration, usually several times a day, this leads to an almost immediate integration of source code modifications. Each version of the system passes all completed acceptance tests and all implemented unit tests providing a useable system immediately with reproducible tests. Traditional systems often have very lengthy integration periods sometimes weeks or months apart leading to unstable systems and requiring costly fixes.

5.3 Challenges When Using XP

XP attempts to bridge the gap between customers and developers, but there is conflict between those who support predictive, heavily documented, plan-driven practices and the low documentation practices of XP. In addition between traditionalists who support the waterfall development family of paradigms and those who support the 'Agile' approach of XP with its iterative, evolutionary and incremental software development practices.

Research conducted at the University of Wales, Bangor [11] used several research methods and data collection techniques to investigate the adoption of Agile methods in commercial software development projects. The choice of data collection methods was determined by the needs of a given aspect of the research and, in particular, by the research questions confronted. Broadly, data collection techniques can be classified as either quantitative or qualitative. These methods are seen as complementary rather than mutually exclusive approaches. Each set of techniques has their particular strengths and weaknesses [12]. Quantitative survey techniques were used for example to obtain a frequency count of developer preferences for the adoption of XP practices. Tape-recorded one-to-one interviews and questionnaires were conducted to elicit the experiences and views, of developers and their managers. Open-ended interviews were adopted to understand the world as seen by the respondents [12].

Companies A, B and C presented three distinct approaches to managing their software development environment, dictated to a large degree by their size, resources and their adopted approaches to software development. Questions targeted the software development process, XP practices, and both managerial and behavioural effectiveness. Company A were a small traditional company not following any distinct development process methodology. Company B were a medium sized rural based software development company of 9 developers which included a developer manager who reported to senior management. Company C were a large city based software development company of 36 developers, one developer manager and two active senior managers.

Several important considerations for companies starting from scratch with Agile methods adoption were revealed by the studies of small and medium sized companies [13]; One company software development manager provides an insight into the dilemmas facing skilled practitioners, when reflecting upon attempts at implementing XP in his early projects:

“One of the key ‘discoveries’ has been the relative ease to which XP has been employed on an all-new projects, and the difficulty in applying XP retrospectively on established systems.”

The University of Wales, Bangor research shows that the XP approach engenders strongly expressed positive and negative reactions.

...everyone who has been involved with it (XP) has said – ‘yeah this is good!’

Peter, Developer Coach, Company B, Interview, April 01.

Well to me it means you learn more , you get more experience. ...

Susan, Developer, Company C, Interview, March 01.

Mary was an experienced developer who had been with Company C for four years, and regarded the adoption of XP as an unwelcome change

I was very happy before XP – but now I’m not sure!

Mary, Developer, Company C, Interview, March 01.

The developer manager at company B provided an insight into the difficulties facing skilled developers that oppose the fundamental 'collective' philosophy of XP.

As one guy told me, 'I'm a system architect and for my next job I want to be a system architect on a bigger project and earn more money. How do I explain to my future employers that for the past 6 years I've merely been a developer in an XP team?'

Denise, Developer Manager, Company B, Interview, January 01.

Much depends on the size and disposition of the company – distributed offices are going to find it difficult to implement some of the practices. Large development teams of more than 25 developers are going to experience communication and coordination difficulties. Integration with heritage or legacy code deeply concerned companies studied. Extreme Programming was regarded as ideal for new projects but was not easily adapted, or integrated into existing projects – and there was strong support for completely re-writing smaller projects.

Given below are factors that have been identified of particular interest in respect of adopting Extreme Programming software development practices:

1. The changing relationship between customer and developer has benefited both parties. Customers are more confident that developers will produce more of what they want, when they want it and within budget. Developers are more confident that they are working on the right features and that they have negotiated realistic time framework to achieve their targets.
2. With only short stories to guide them, developers commence code seeking regular clarification and negotiated changes with the customer on a daily basis.
3. Short release cycles keep the customer realistically informed of progress as well as providing opportunity-capture, and disaster-impact cushioning.
4. Developers did not see the point of simplicity when they could quickly and easily add enhanced features, or hooks that would help future-proof the project.
5. Legacy code dominate concerns of both managers and developers – Developers were reluctant to mess with existing code if they thought it would be better both in the long and short term to start from scratch. Traditional developers in small companies ignored problems until they became a priority that surprised them, at which point they scrambled to put out the fires.
6. Developers were found to fall into different categories - those who bought-in to XP or were willing to buy-in, and those who were waiting to see how it imposed upon their position. The later category were unwilling or unhappy, to relinquish their status and benefits.

6. Comparison of DSDM, RUP and XP

The RUP complements DSDM by providing exceedingly detailed guidance on the activities that go into making a successful project - under the umbrella of the DSDM principles.

XP focuses on the core disciplines needed to deliver software:- requirements, coding, design, test and project/team management issues. XP provides excellent principles and practices to follow in these areas but not surprisingly doesn't neatly match all types of projects. For larger project demands, such as business modeling, configuration management and application deployment, RUP can be combined with XP to supplement the thin parts of XP to facilitate a better solution.

Methodology	Key Points	Special Features	Identified Shortcomings
XP	Customer driven development, small teams, daily builds	Refactoring – the ongoing redesign of the system to improve its performance and responsiveness to change.	While individual practices are suitable for many situations, overall view and management practices are given less attention.
DSDM	Application of controls to Rapid Application Development, use of timeboxing, empowered DSDM teams, active consortium to steer the method development.	First truly agile software development method, use of prototyping, several user roles: “ambassador” “visionary” and “advisor”.	While method is available, only consortium members have access to whitepapers dealing with the actual use of the method.
RUP	Complete Software Development model including tool support. Activity driven role assignment	Business Modeling. Tool family support	RUP has no limitations in the scope of use. A description of how to tailor to specific changing needs is missing

Table 2: Comparison of XP, DSDM and RUP (adapted from [14] pp.89-90)

7. Adopting Agile Methods in Large Enterprises

We have identified three main strategies for adopting Agile methods in large enterprises: Proof of Concept, Hybrid and Strategic. These approaches are discussed below.

7.1 Proof of Concept

A cautious approach to the adoption of Agile methods is to use XP practices such as planning games, stories, pair programming, simple design and upfront testing in order to develop a proof of concept implementation of the system under development. This is a simple approach to the adoption of Agile methods for several reasons:

- Risks are lower since the resulting product is not intended for deployment,
- Proof of concept teams tend to be smaller and self contained,
- Reduced project scope (in proof of concept phase) makes full-time involvement of customer more feasible

7.2 Hybrid Methodologies

Most large enterprises have an established internal project management process or methodology. Many Business Process Management or Customer Relationship Management projects involve both IT practitioners and other enterprise stakeholders. It is not always practical for a project team to adopt an Agile method at an enterprise-wide level. A Hybrid approach has been used at several major financial services institutions. In this approach stakeholders outside the IT team continue to use the Enterprise methodology. Within the IT team, however, activities and deliverables are selected from both the Enterprise methodology and from applicable Agile methods to suite the needs of the project. For example, there are benefits to adopting an incremental approach to design, construction and unit testing, as shown in Figure 3. The benefits from this hybrid approach can include risk reduction, learning and refactoring between increments and early delivery of core functionality.

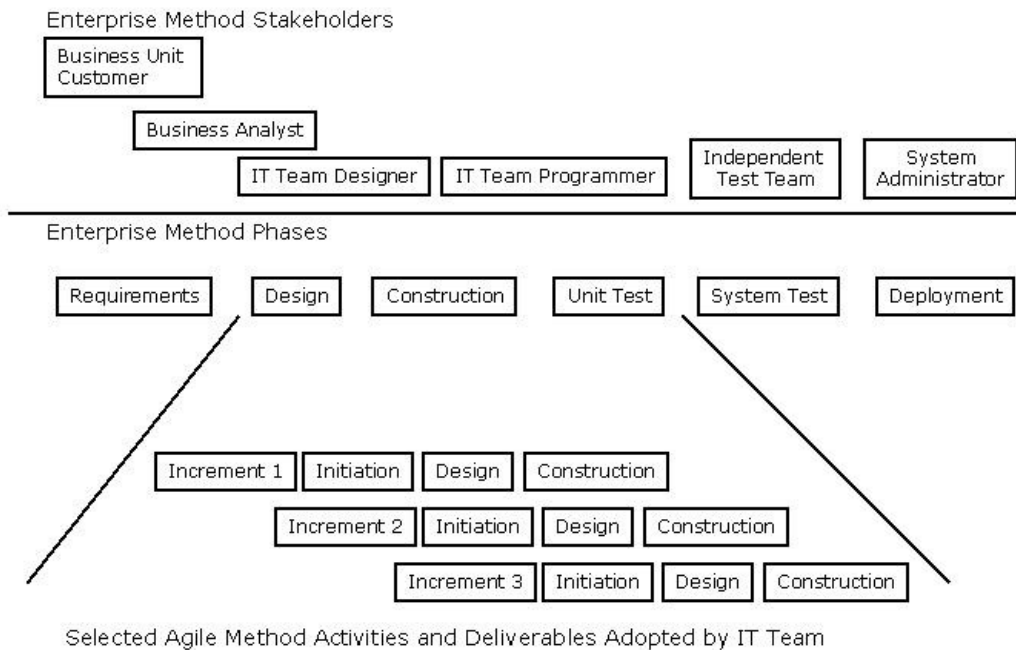


Figure 3: Hybrid Development Method

Staff from business units in other parts of the enterprise can still conduct early Requirements Analysis and System Testing. It may not be possible or even necessary for these other business units to fully adopt Agile methods.

7.3 Strategic Agile Adoption

There are large enterprises that can benefit from adoption of Agile methods on a project-by-project basis. There may be several factors that permit strategic adoption of Agile methods:

- The enterprise does not have (or does not mandate use of) a waterfall-based enterprise methodology,
- The existing enterprise methodology is sufficiently flexible to allow adoption of Agile practices while maintaining compliance, or
- Executive sponsorship of Agile methods is sufficiently senior to mandate adoption on all the stakeholder groups.

Software development projects operating in such a climate have the opportunity to embrace Agile methods according to the needs of the project.

8. Conclusions

Developers have expressed enthusiasm for the socially rewarding benefits of several of the XP practices. Pair programming brought fun back into software development for some. However, the collaborative nature of agile methods, such as XP, can make it difficult for experienced developers to gain leadership positions in projects. This lack of a distinctive architectural role could be seen as undermining the professional development required for career progression. Pair-programming appears most effective when both developers are contributing a broadly similar level of expertise. Pair programming can be advocated for mentoring. A careful approach to coaching is required to avoid this being reduced to a tedious observation of an expert in performance of their art.

XP can be helpful because it offers the chance of support to otherwise isolated developers. It improves communications through pair programming and raises the morale of some developers but concerned some developers who felt XP undermined their status. XP holds the prospect of better customer relations, which is appreciated by developers. Not everyone accepts the idea of simple design and this concept may need re-evaluation.

Wholesale adoption of Agile methodologies in large enterprises may be difficult. The organisational constraints on adoption of Agile methodologies in large organisations seem considerable. Hybrid methodologies integrating agile and more traditional approaches can be successful. This allows developers in such an organisational context to benefit from Agile concepts like pair-programming, short-release cycles and continuous integration.

9. References

- [1] B. W. Boehm, "A Spiral Model for Software Development and Enhancement", IEEE Computer, vol. 21, pp. 61-72, May 1988.
- [2] Manifesto of the Agile Alliance, <http://www.agilemanifesto.org/> 2001.
- [3] P. Abrahamsson, J Warsta, M T Siponen and J Ronkainen, "New Directions on Agile Methods: A Comparative Analysis", Proc. of the 25th Int. Conf. On Software Engineering, (ICSE '03), Portland, Oregon, May 2003, pp. 244-54.
- [4] J. Stapleton, "Dynamic Systems Development Method – The Method in Practice" Addison-Wesley 1997.
- [5] I. Jacobson, G. Booch and J. Rumbaugh, "The Unified Software Development Process", Addison-Wesley Longman, 1999.
- [6] M.Fowler, "The New Methodology", <http://martinfowler.com/articles/newMethodology.html>.
- [7] K. Beck, "Extreme Programming Explained : Embrace Change", Addison-Wesley, 1999.
- [8] K. Beck & M. Fowler "Planning Extreme Programming", Addison-Wesley, 2001.
- [9] M. Fowler, & K. Beck. "Refactoring : Improving the Design of Existing Code", Addison-Wesley, 1999.
- [10] R. Jeffries, A. Anderson & C. Hendrickson "Extreme Programming Installed", Addison-Wesley, The XP series, 2001.
- [11] R. Gittins, "Strategies, Methods and Algorithms for Novel Software Tools, to Support the Analysis and Design Process in Software Development". PhD Thesis, University of Wales, Bangor – In preparation, 2004.
- [12] M. Q. Patton, Qualitative Research & Evaluation Methods. 2002. SAGE Publications.
- [13] R. Gittins, J M Bass and S Hope, "A Comparison of Software Development Process Experiences", 5th Int. Conf. on Extreme Programming and Agile Processes in Software Engineering, Garmisch-Partenkirchen, Germany, June 2004, to appear.
- [14] P. Abrahamsson, O. Salo, J. Ronkainen and J Warsta, "Agile Software Development Methods: Review and Analysis", VTT Technical Research Centre of Finland, VTT Publications, Vol. 478, 2002.

About The Authors

Robert Gittins is a research assistant and PhD candidate at the School of Informatics, University of Wales Bangor UK. He is a leading member of the Software Engineering and Systems Integration (SESI) group. His research interests include qualitative research methods in Software Engineering and the development of Agile Methods. He has consulted with several companies on the adoption of agile methods for software development and contributed to a book and international conference on the subject. Robert has extensive commercial and industrial experience prior to obtaining his BSc degree in Computer Systems with Business Studies at University of Wales, Bangor.

Robert is a member of the IEEE Computer Society.

Dr Julian Bass obtained his PhD in hard real-time fault-tolerant software systems at the Department of Automatic Control and Systems Engineering in 1995 and an MEd in Teaching and Learning in Higher Education from the Division of Education in January 2003, both at the University of Sheffield. He has co-authored nearly 50 research publications in hard real time software development and fault-tolerant voting algorithms. He was formerly a Lecturer at the University of Wales, Bangor, UK where he was Computer Science course director. Previously he was course director of the MSc in Control and Systems Engineering at the University of Sheffield. Dr Bass is a Chartered Engineer and member of the British Computer Society, ACM and IEEE.

Dr Bass is Chordiant Training Product Manager at Business Agility, where he provides consulting and develops and delivers tailored training programmes in the design and construction of object-oriented enterprise integration software for Fortune 500 clients. Dr Bass currently has interests in n-tier enterprise architectures, component technologies and agile software development methodologies.