eXtreme Programming in Context

Introduction

In this briefing we provide an objective overview of a new lightweight development process called eXtreme Programming (XP). Specifically we study the process in the following context:

- Requirements Change: It is the norm for customer requirements to be vague, incomplete and subject to change. These characteristics are more prevalent today than ever before with the Web continuously reshaping underlying business models. To try and pin the user down to a fixed set of well-defined requirements can be dangerous and counter productive. The process should allow the developer and customer to explore the problem domain and solution space, learning from each other as they do so.

- Expressive Environments: With modern Object-Oriented (OO) languages such as Java and Eiffel and their associated class libraries and environments, we now are able to translate requirements into code quicker than ever before. If we also adopt a disciplined OO development approach to design, we can easily implement new functionality without impacting on the existing design quality and correctness of an application.

- Rapidly Changing Technology: Modern e-business focused technologies such as the Java 2 platform, Corba, and Microsoft's COM and .NET technologies are creating a phenomenon known as the disappearing programmer. Application designers and architects need to program in these technologies, in order to be able to design meaningful solutions instead of just bubble diagrams.

eXtreme Programming (XP) is a method that attempts to tackle the above problems head on. With XP the application is built in small "one to three week" iterations. During each iteration the team implements a set of features arrived at and refined through conversation between the developers and the on-site customer. It is the customer's responsibility to write the functional tests. It is the developer's responsibilities to estimate the time and risk associated with each feature implementation, and to implement the features in such a way that the simplicity and correctness of the design is not compromised.

Lets take a black box view of the method from the point of view of the Customer before we look at how the developer meets the customer's requirements.

© 2000 Centre for Software Engineering

The XP Customer

Kent Beck, one of the creators of XP, has put forward Conversation as a new paradigm for software development. Features, called user stories, are scribbled onto an index card and put into “cold storage” until a decision is made to implement them in an iteration. An iteration should last between one and three weeks. The stories are reanimated though the *conversation* between the customer and developer. A feature’s development can then be viewed as a conversation between the developer and the customer, that starts with an index card and lasts until the functional tests have completed successfully for the feature.

The primary role of the customer is to be on hand to guide the development process. Ideally the customer should be on-site. The unavailability of a customer provides a difficulty for XP. The lack of daily face to face communication between developer and customer makes the balance shift back towards the capturing of user requirements at a particular point of time, i.e. development becomes an asynchronous message passing process. This is fine if the models are complete, correct, and easily understood and developers do not have to refer back to the customer on any points. When the developer has to stop in mid development because they do not have sufficient information, the development clock does not stop ticking. Under time pressure the developer may be tempted to proceed based on unsafe assumptions.

The customer also has very specific duties to fulfil, namely:

- Selecting the features that will be implemented in an iteration. This selection will be based on their own needs, taking into account the time estimates and technical risks identified by the developers.
- Providing the developers with all the domain information they need to implement a feature.
- Writing the functional tests to ensure that the feature performs as expected.

eXtreme Discipline

Complexity is the enemy of software quality. The XP developer’s energy is concentrated on producing the simplest solution for each user requirement. XP’s touchstone for software quality is duplicate logic. Other warning signs include long function bodies, long parameter lists, and verbose code comments. Writing about quality, Beck says, "The only possible values are 'excellent' and 'insanely excellent', depending on whether lives are at stake or not".

It seems a lot to ask of developers, especially novice developers, to adhere to this model of pure simplicity. This is where XP provides a set of interdependent practises and techniques, that makes the goal of consistently correct and elegant software achievable, even for teams with their fair share of inexperienced programmers.

Refactoring

Refactoring has been defined by Martin Fowler as “the process of changing a software system in such a way that it does not alter the external behaviour of the code yet improves its internal structure. You learn from building the system how to improve the design. The resulting interaction leads to a program with a design that stays good as development continues.”

With refactoring you look on each change as an opportunity to improve the existing design of the system. This is the opposite of the natural software product life cycle which gradually sees changes over time chipping away at the quality and cohesion of the product, until the product can no longer tolerate any changes and it has to be dumped.

The Refactoring based approach has grown out of the experience of software framework developers. A framework is a skeletal piece of software that provides core functionality for a particular application domain. Frameworks offer the biggest reuse potential, but the downside is they are very difficult to develop, and almost impossible to develop from scratch. Frameworks tend to evolve over time, and one of the key techniques for allowing them to evolve is constant refactoring. If we are to build solutions with the maximum reuse potential then the industry lesson is that you have to adopt a refactoring based approach to development.

How do you do refactoring and how does it impact on the development process? Basically, a refactoring stage needs to be built into each small system change. Instead of the developer rushing in and making the change, they adopt the following strategy. "When you find you have to add a feature to a program, and the program's code is not structured in a convenient way to add the feature, first refactor the program to make it easy to add the feature, then add the feature".

Unit Testing

Kent Beck and Erich Gamma have nicely summed up why programmers find testing painful - "Every programmer knows they should write tests for their code. Few do. The universal response to "Why not?" is "I'm in too much of a hurry." This quickly becomes a vicious cycle - the more pressure you feel, the fewer tests you write, the less productive you are and the less stable your code becomes. The less productive and accurate you are, the more pressure you feel."

With refactoring you are constantly changing things that are already working. You therefore need to be continuously running regression tests to check you have not broken any of the code with a refactoring. XP advocates a test-first approach to software development. Before you write a piece of code you must first write the test, see the test fail and then write the code that makes the test work. This is a variation on the theme of specifying what you want to achieve before you consider how you will implement it.

What you need to make strong unit testing a reality is tool support that allows you to quickly write and run your own tests. The tests should be self-checking. Basically you want a scenario where you can push a button, your tests will run in a fraction of a minute, and you will be told they all succeeded, or if there was a failure which test failed. The XP camp has come up with an object oriented framework called XUnit that allows you to do this. The Java variant of this framework is known as JUnit. It is a simple and effective tool, with the added benefits that it is free and open source, so you can easily customise it to suit your particular testing approach.

XP has a lot to say about the importance of testing, but precious little on how to ensure your testing is comprehensive and complete or how to track the dependencies between different tests. For adding rigour and completeness to your tests you may wish to tap into the existing extensive testing literature. Also the unit tests provide an intuitive description of the expected behaviour of a function but not necessarily a complete picture. A complementary technique known as Design By Contract, allows you to specify system behaviour more precisely and completely.

Collective Ownership and Continuous Integration

A refactoring based approach implies that team members can change each others code at any time. This has the benefit that complex code does not hang around for very long as it soon gets factored into simpler code. It also drives out the need for continuous integration. XP development teams integrate at least once a day. Ideally all unit tests should be run every time someone checks back in a piece of code, which could mean that system integration is done every few minutes.

Eight-Hour Burn

XP view software development as a team centred process, if all the members are not contributing to the best of their abilities then disaffection with each other and the game plan will soon set in. The ideal is for people to work only while they are fresh and enjoying their work. The latest XP catch phrase for this concept is the eight-hour burn.

Pair Programming

With the goal of development being a lean, elegant application that can flex in anyway the customer wants it to go, how does the process accommodate novice programmers? The XP technique here is for programmers to pair off with each other on tasks. An XP task should not take longer than a day to complete. Pair Programming allows new programmers to learn from more experienced developers at their own pace while making an ever increasing number of contributions, until they are ready to accept responsibilities for specific tasks.

> Pair Programming has many other benefits apart from helping to train in new team members.
>
> - It encourages team interaction and knowledge dissemination
> - It speeds up development, people working in pairs tend to avoid going down blind alleys

XP Concerns

Before we jump off our whistle stop tour of XP, lets run through some of the concerns that people have with the approach.

What about taking a big picture view?

With XP you do the simplest thing to solve today's problem and rely on refactoring to grow a quality design over time. The XP contention is that once you start building for the future then you are adding unnecessary complexity and reducing the flexibility of the system. "If you believe that the future is uncertain, and you believe that you can cheaply change your mind, then putting in functionality on speculation is crazy, put in what you need when you need it", writes Beck.

Where are the models and documentation?

XP puts emphasis on code expressiveness and flexibility. With XP we might use models to help us sketch a solution. This is in line with the current popularity of round trip engineering tools that allow the model and code to be kept in sync with each other. Also tools such as JavaDOC allow us to generate documentation straight from program comments.

Where is the architect?

With XP the developer is empowered to contribute to the architecture as development progresses. This is a natural consequence of working with an OO platform such as Java, where the associated class libraries such as Java Servlets, Remote Method Invocation (RMI), and the utility classes can only be fully mastered by people who spend time developing with them.

What about distributed development?

Ideally developers and customers should be at the same location. If they are physically separate then you are impeding one of the basic XP tools: conversation.

Summary

In conclusion, whether you agree with XP's status as a full-fledged software engineering process or not, is not really the point. The point is that XP highlights the core practises that your team *must get right* if you wish to develop quality software on time while having to deal with changing requirements. From this point of view it is of key consideration to nearly everybody engaged in software development today.

Further Information:

WWW:

http://www.extremeprogramming.org
http://www.xprogramming.com
http://www.junit.org
http://www.eiffel.com – for information on Design By Contract

Books:

eXtreme Programming eXplained
Kent Beck, Addison Wesley '99

eXtreme Programming Installed
Ron Jeffries, et al, Addison Wesley 2000

Planning eXtreme Programming
Kent Beck and Martin Fowler, Addison Wesley 2000

Refactoring : Improving the Design of Existing Code
Martin Fowler, Kent Beck (Contributor), John Brant (Contributor), William Opdyke, Don Roberts
Addison-Wesley '99

Design Patterns, Elements of Reusable Object-Oriented Software
Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides
Addison Wesley 1995

Patterns are often the endgame of a sequence of refactorings. Design patterns capture proven effective design solutions and are defined as *"descriptions of communicating objects and classes that are customised to solve a general design problem in a particular context."*

Technical Briefing Notes are issued on a range of software engineering topics as an aid to software developers, project leaders and managers. The intention is to provide a 'status report' on the state of the art (and/or the state of practice) in relation to particular aspects of software engineering. In addition they aim to highlight, where appropriate, a likely roadmap on a time horizon for future developments of the technology.

Centre for Software Engineering Ltd, Dublin City University Campus, Dublin 9, Ireland.
Telephone: +353 1 7005750 Fax: +353 1 7005605 Email: admin@cse.dcu.ie