

Extreme Programming

Embracing Change

Tim Bacon, ThoughtWorks Ltd
tbacon@thoughtworks.com

XP in a nutshell

- Identifies 4 team roles
- Defines 12 practices, and 4 values to guide the application of the practices
- Applicability boundaries are well mapped out
- Continual change is both an expected input to and an outcome of the process
- Warning for the weak of heart:
 - Extreme Programming **is** extreme
 - It takes discipline to make it work
 - It can be an all-or-nothing proposition
 - There are organisational requirements

Why has XP gathered so much attention?

- Because practitioners deliberately court controversy?
 - Kent Beck and Ron Jeffries especially...
- Because the XP philosophy is provocative?
 - If it's worth doing, it's worth doing all the time
 - Do the simplest thing that could possibly work
 - Have you got a test for that?
 - No big design up front (BDUF)
 - You aren't going to need it (YAGNI)
- Because of the accolades from XP groupies?
 - "The best project I ever worked on"
- Because it directly addresses the frustrations of programmers and customers?

XP roles

- Programmers write code
 - Programmers make all the technical decisions
 - XP is unashamedly focussed on the activities of programmers, and the interactions between programmers and others
- Customers make all the business decisions
 - E.g. release dates, cost-value tradeoffs, work prioritisation
- Trackers monitor progress
 - Tracker could be a part-time role
- Coaches maintain the discipline required to make XP work
 - Act as a teacher, guide, change agent, and conscience

Test driven development

- All code is exercised by a unit test
 - The unit test is written before the code it exercises
- Progress is demonstrated to the customer with suites of functional tests
 - The test is specified by the customer before the code is written
- These tests:
 - Verify correctness
 - Detect changes made in the future
 - Drive out a design that fits the context of the application
 - Tell programmers when to stop
- xUnit (e.g. JUnit) and FIT are invaluable tools

Continuous integration

- A dedicated machine rebuilds the application from the source code repository and reruns the test suites many times every day
- All unit tests must pass before new code can be added to the system
- New code is integrated with existing code regularly
- Check-out / check-in cycle is minutes not days
- Tools for CI such as CruiseControl can be helpful

Working in pairs

- Code is written by two people working together at a single keyboard
- The person who is not typing is performing a real-time code review and thinking ahead
 - Not a passive role!
- Pairs are fluid and change regularly
 - Spreads knowledge about the workings of the system among the team
- Pair programming looks expensive but is not
 - “Why are two people doing one person’s work?”
 - Economic case is made by payoff through higher quality

On site customer

- The customer sits with the team
 - Always available to answer questions about functionality,
 - Can easily monitor progress
 - Can change the scope of the programmers' work at any time
 - To be the business ambassador for the application they may have to give up some of their 'day-to-day' activities
- This is often an organisational sticking point

Frequent small releases

- The team delivers working software to the customer in iterations lasting 1-4 weeks
 - This software is of releasable quality and is typically deployed to a staging environment in each iteration
- Iterations are combined into 2-3 month releases
 - The output of the last iteration in a release is deployed to the production environment
 - There is no need for a final testing or acceptance phase

Priority driven planning

- At the beginning of each iteration a joint planning session called the 'planning game' is held
 - The development team provides estimates for new pieces of functionality
 - The customer prioritises the outstanding work to be done
 - The highest priority work that can be achieved in the next iteration is broken down in greater detail and the rest is left for future iterations

Sidebar: XP Stories

- Functionality is expressed in ‘stories’ that are written on index cards
 - A story describes in natural language what the system must do and is accompanied (often on the reverse of the card) by an acceptance test that will validate the delivery of the story
 - Overly long or complex stories are split into smaller stories, until both the customer and the developers are happy that each story has business value and can be estimated
 - Each story card has a unique reference code
- Each story card has an estimate in ‘story points’
 - These reflect the difficulty of the programming tasks required to complete the story (i.e. effort, not duration)

Sidebar: XP Tracking

- This iteration's stories are tracked on a board or wall
 - Arranged in priority order from left to right
 - Broken down in to tasks and tests from top to bottom
 - Each card is marked off with a big tick when completed
- Story completion is binary: it is all done or not yet done
 - Avoids 80% completeness syndrome
 - Brings out the completer-finishers in the team
- Progress is very visible to all interested parties
- A daily standup meeting around the story board keeps attention on the most important tasks for the day

Maintainable pace

- The number of story points completed per iteration is tracked
- The actual work rate of the previous iteration is used to adjust the number of stories in this iteration
 - Capacity reflects the likely pace of the team
- Overtime is not regularly used to meet release deadlines
 - The effects of overwork are counter-productive in anything but the short term
 - Overtime is loosely defined as periods of time spent working when you feel you should have gone home already

Collective ownership

- The team as a whole owns all the code
- No part of the system is the responsibility of a single person
- Anyone can alter existing code at any time
 - So long as the tests for that code continue to pass
- However, ego-less programming is hard in practice
 - Requires mutual respect and willingness to give-and-take

Relentless refactoring

- Existing code is constantly being changed to accommodate new functionality
- Refactoring is used to maintain the design quality of the resulting code base e.g. simplifying classes by removing duplication
 - This allows the code base to evolve over time in a maintainable fashion
 - A refactoring is a set of behaviour-preserving code changes
 - Refactoring is to programming what weeding is to gardening
- The “red-green-refactor” cycle is used to ensure that tests pass and design quality is maintained

Simple design

- In order for the code to be owned and refactored by the whole team, the design must be simple
- Simple designs can support complex functionality, but complex designs can preclude the rapid addition of new functionality
- Simple does not mean simplistic
 - Simple design has aesthetic properties, e.g. elegance, coherence, self-referentiality
 - Simplistic design is ugly

Using the language of the application domain

- The application domain can provide a vocabulary that is unambiguous to both customers and programmers
- The design of the system should reflect the workings of the application domain
- A metaphor can be used to sum up the way the system works in a statement that is easy to communicate
- It has taken a long time to clarify this aspect of XP

Having a code standard

- A coding standard ensures that all code is easily read by the whole team
 - And prevents 'religious wars' that slow the pace of development, e.g. arguments over where to place curly braces
- Having a standard is more important than which standard is chosen
- A tool can be used to regularly reformat code files to ensure that the whole of the system conforms to the chosen standard

The values tie the practices together

- Communication
 - “Problems with projects can invariably be traced back to somebody not talking to somebody else about something important”
- Feedback
 - Feedback promotes validation and facilitates improvement
- Simplicity
 - No features that ‘might be needed’
 - No code tricks just because they allow a ‘cool’ technology to be used
 - Focus on completing discrete pieces of work that have business value
- Courage
 - Doing nothing is an act too. Is it the right one? How will you know?
 - Courage provides the impetus necessary to tackle difficult problems

XP is more than the sum of its parts

- This is both a strength and a weakness
 - The parts themselves are strong (Kent said “I just watched what Ward was doing and wrote it down”)
 - However the whole can be hard to swallow organisationally
 - On site customer, pair programming and maintainable pace are often sticking points
- However if all you take away from XP is test driven development and continuous integration then your project will still benefit

Questions?

- Kent Beck and Dave Astels have both written good books covering Test Driven Development
- The Addison Wesley series of XP books are excellent
 - Kent Beck – the white book
 - Ron Jeffries – the pink book
 - Ken Auer – the purple book
 - Pete McBreen – the black book