Agile: Extreme Programming

Dinesh Anantwar

What is Extreme Programming?

- An agile development methodology
- Created by Kent Beck in the mid 1990's
- A set of 12 key practices taken to their "extremes"
- A mindset for developers and customers
- A Religion/Cult? ☺ ##

Emergence

- XP provides values and principles to guide team behavior
- Team expected to self-organize
- XP provides specific core practices
- Each practice is simple and self-complete ##

Key Assumption of XP

- The cost of change curve can be flattened
- How?
  - Better program organization (object-oriented)
  - Simple designs kept simple with refactoring
  - Automated testing to maintain integrity
  - Better tools and faster machines
- Because of this we can:
  - Make decisions later, defer costs, keep options open
  - Reduce the time-to-ROI with partial solutions
  - Learn from feedback, adapt to change ##

Why Is It Called “Extreme?”

- Selected the minimal set of effective practices
- “Turned the knob to full/max” on each practice
  - Very short cycles (planning game)
  - Continuous code reviews (pair programming)
  - Extensive testing (unit testing, acceptance testing)
  - Continuous integration
  - Constant design improvement (refactoring)
  - Etc… ##

The 12 Practices

- The Planning Game
- Small Releases
- Metaphor
- Simple Design
- Testing
- Refactoring
- Pair Programming
- Collective Ownership
- Continuous Integration
- 40-Hour Workweek
- On-site Customer
- Coding Standards

1 - The Planning Game

- Planning for the upcoming iteration
- Uses stories provided by the customer
- Technical persons determine schedules, estimates, costs, etc
- A result of collaboration between the customer and the developers ##

Planning Game (Release & Iteration Planning)

- Facilitates incremental project planning as more and better information learned
- Develop rough plan first, refine incrementally
- Release planning sets longer-term goals
  - Releases are typically from 1 to 6 months.
  - Customer as well as Developer involved.
- Iteration planning sets short-term time-box
  - Iterations are typically from 1 week to 1 month.
  - Customer is not involved. ##

The Planning Game – Advantages

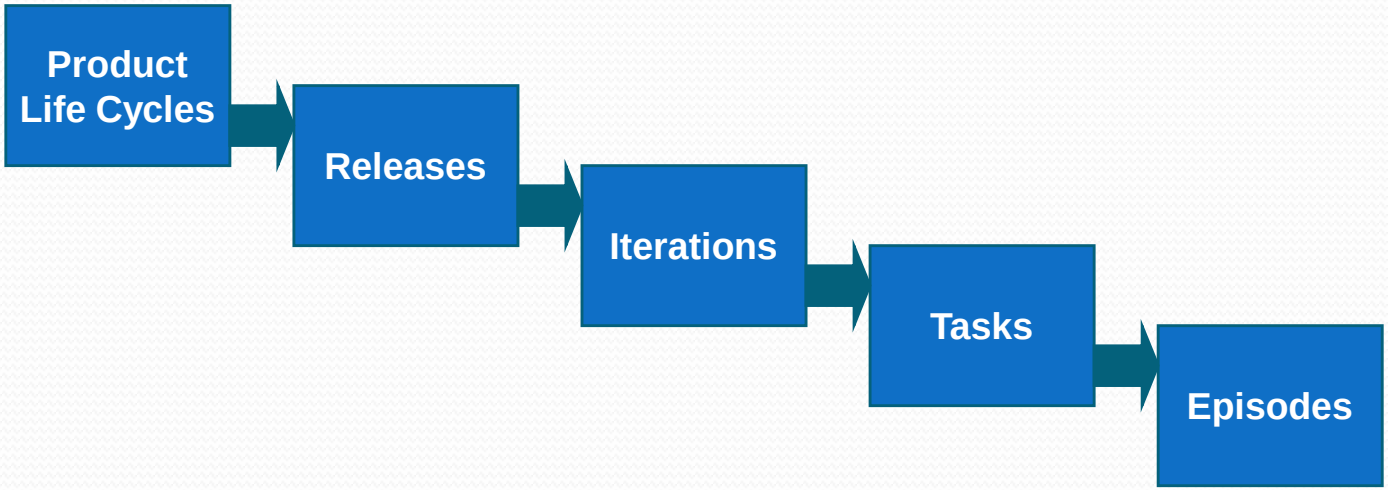
- Reduction in time wasted on useless features
- Greater customer appreciation of the cost of a feature. i.e. transparency in planning.
- Less guesswork in planning ##

The Planning Game – Disadvantages

- Customer availability
- Is planning this often necessary?

XP Process Cycle

- XP is iterative and incremental
- XP is driven by time-boxed cycles
- The rhythm of the XP process is crucial



2 - Small Releases

- Releases small as possible while still delivering enough value to be worthwhile
- Release early to begin providing business value early (maximize ROI over time)
- Release early to obtain feedback and facilitate steering
- Small releases minimize early commitment, leaving open options longer
- Support the planning game ##

Small Releases – Advantages

- Frequent feedback
- Tracking
- Reduce chance of overall project slippage

Small Releases – Disadvantages

- Not easy for all projects
- Not needed for all projects
- Versioning issues

3 – Metaphor

- The oral architecture of the system
- A common set of terminology

Metaphor

- For the team to communicate, it needs:
  - A common mental model of the system
  - A common language to talk about the system
- The ideal model and language is that of the actual problem domain (Domain-Driven Design)
- The ideal model evolves, and can take time
- A metaphor can provide a short-cut to reach the common model and language goals
  - “The payroll system is like an assembly line, where each deduction is bolted on to the paycheck as it moves down the line.”

Metaphor – Advantages

- Encourages a common set of terms for the system
- Reduction of buzz words and jargon
- A quick and easy way to explain the system

Metaphor – Disadvantages

- Not Easy to define common terminology

4 – Simple Design

- K.I.S.S.
  - Keep It Simple Stupid
- YAGNI
  - You Aren't Going to Need It
  - Do as little as needed, nothing more
- DRY
  - Don't Repeat Yourself
  - Opp. Of this is WET (Write Everything Twice) ##

Simple Design – Advantages

- Time is not wasted adding superfluous functionality
- Easier to understand what is going on
- Refactoring and collective ownership is made possible
- Helps keeps programmers on track

Simple Design – Disadvantages

- What is “simple?”
- Simple isn’t always best

6 – Testing

- Unit testing
- Test-first design
- All automated

Test-Driven Development

- Likely the most innovative XP practice
- Developer writes a unit test prior to writing code
- Developer writes just enough code to make the unit test pass
- Developer refactors to ensure code is simple and clean (standards met)
- Developer continues until acceptance test passes
- Test-driven-development tends to:
  - Result in a remarkably complete set of tests
  - Drive the code to be the most simple, minimal
  - Leave clear examples of intent and use ##

Testing – Advantages

- Unit testing promote testing completeness
- Test-first gives developers a goal
- Automation gives a suite of regression test

Testing – Disadvantages

- Automated unit testing isn't for everything
- Reliance on unit testing isn't a good idea
- A test result is only as good as the test itself

6 – Refactoring

- Changing how the system does something but not what is done
- Improves the quality of the system in some way ##

Refactoring (Design Improvement)

- Improve the design of existing code without changing its functionality
- Allows design to incrementally evolve
- Refactoring not random, driven by learning from new implementations
- Refactoring can occur just prior or just after writing new code
- Refactoring drives code towards higher-level design patterns
- ***Not*** a substitute for hacking first, thinking later
- Contributes to avoidance of design debt

Refactoring – Advantages

- Prompts developers to proactively improve the product as a whole
- Increases developer knowledge of the system

Refactoring – Disadvantages

- Not everyone is capable of refactoring
- Refactoring may not always be appropriate
- Would upfront design eliminate refactoring?

7 – Pair Programming

- Two Developers, One Computer
- One “drives” and the other thinks
- Switch roles as needed ##

Pair Programming

- All production code written in pairs
- Programming as a collaborative conversation
- Pairing is ***not*** one person looking over the other's shoulder
- Pairing provides (partial list):
  - All design decisions made by at least two
  - Continuous code reviews
  - Cross-training and mentoring spreads with pairs
  - Eases specialization dependencies & bottlenecks
  - Motivation as well as sanity checks, builds teamwork
- Research shows pairing can be more effective than solo programming
- Short-term costs made up by long-term benefits
- Contributes to avoidance of knowledge debt

Pair Programming – Advantages

- Two heads are better than one
- Focus
- Two people are more likely to answer the following questions:
  - Is this whole approach going to work?
  - What are some test cases that may not work yet?
  - Is there a way to simplify this? ##

Pair Programming – Disadvantages

- Many tasks really don't require two programmers
- A hard sell to the customers
- Not for everyone

8 – Collective Ownership

- The idea that all developers own all of the code
- Enables refactoring
- Any Developer can make changes to any part of the code as needed for their tasks
- Eliminates queuing bottlenecks
- All Developers responsible for integrity of the code base
- “You break it, you fix it” encourages collective responsibility as well as freedom to experiment ##

Collective Ownership – Advantages

- Helps mitigate the loss of a team member leaving
- Promotes developers to take responsibility for the system as a whole rather then parts of the system

Collective Ownership - Disadvantages

- Loss of accountability (Every one's responsibility is No one's responsibility ☺)
- Limitation to how much of a large system that an individual can practically "own" ##

9 – Continuous Integration

- New features and changes are worked into the system immediately
- Code is not worked on without being integrated for more than a day ##

Continuous Integration

- Avoidance of “big bang” integrations
- Integration for each pair occurs several times each day
- All tests run prior to commitment to code base
- Makes the cause of failures more obvious
- Minimizes merge pain
- Facilitates the code base evolving steadily
- Forces bug fixing to occur immediately
- Often supplemented by daily builds
- Contributes to the avoidance of quality & integration debt

Continuous Integration - Advantages

- High Predictability
- Robust system
- Enables the Small Releases practice

Continuous Integration – Disadvantages

- The one day limit is not always practical
- Reduces the importance of a well-thought-out architecture

10 – 40-Hour Week

- The work week should be limited to 40 hours
- Regular overtime is a symptom of a problem and not a long term solution

40-Hour Week – Advantage

- Most developers lose effectiveness past 40-Hours
- Value is placed on the developers well-being
- Management is forced to find real solutions

40-Hour Week - Disadvantages

- The underlying principle is flawed
- 40-Hours is a magic number
- Some may like to work more than 40-Hours

11 – On-Site Customer

- Just like the title says!
- Acts to “steer” the project
- Gives quick and continuous feedback to the development team

On-Site Customer – Advantages

- Can give quick and knowledgeable answers to real development questions
- Makes sure that what is developed is what is needed
- Functionality is prioritized correctly

On-Site Customer – Disadvantages

- Loss of work to the customer's company (Customer needs to invest)
- Difficult to get an On-Site Customer
- The On-Site customer that is given may not be fully knowledgeable about what the company
- May not have authority to make many decisions
- Micromanagement might be possible ##

12 – Coding Standards

- All code should look the same
- It should not possible to determine who coded what based on the code itself
- Consensus of coding style and practices
- Facilitates moving about the code base
- Contributes to definition of clean code and “doneness”
- Removes distraction of endless arguments
- Standards evolve over time ##

Coding Standards – Advantages

- Reduces the amount of time developers spend reformatting other peoples' code
- Reduces the need for internal commenting
- Call for clear, unambiguous code

Coding Standards – Disadvantages

- Might degrade the quality of inline documentation

Additional Practices

- Stand-Up Meetings
- Tracking & Metrics
- Retrospectives
- Big Visible Charts
- Team Culture
- Consensus
- Skunk Works, War Room
- Version & Configuration Management, Automated Builds, Build Promotion

Manager and Customer Bill of Rights

- *You have the right to an overall plan, to know what can be accomplished, when, and at what cost.*
- *You have the right to get the most value out of every programming week.*
- *You have the right to see progress in a running system, proven to work by passing repeatable tests that you specify.*
- *You have the right to change your mind, to substitute functionality, and to change priorities without paying exorbitant costs.*
- *You have the right to be informed of schedule changes, in time to choose how to reduce scope to restore the original date. You can cancel at any time and be left with a useful working system, reflecting investment to date.*

Programmer Bill Of Rights

- *You have the right to know what is needed, with clear declarations of priority.*
- *You have the right to produce quality work at all times.*
- *You have the right to ask for and receive help from peers, superiors, and customers.*
- *You have the right to make and update your own estimates.*
- *You have the right to accept your responsibilities instead of having them assigned to you.*

XP – Advantages

- Built-In Quality
- Overall Simplicity
- Programmer Power
- Customer Power
- Synergy Between Practices

XP – Disadvantages

- Informal, little, or no documentation
- Scalability
- Contract Issues
- Misconception on the cost of change
- Tailoring

When is XP Advantageous

- Highly uncertain environments
- Internal projects
- Joint ventures

When is XP Disadvantageous

- Large, complex environments
- Well understood requirements
- Distant or unavailable customer

XP References

Online references to XP at

- http://www.extremeprogramming.org/
- http://c2.com/cgi/wiki?ExtremeProgrammingRoadmap
- http://www.xprogramming.com/

Final Thoughts

- Why XP succeeds
- Why XP is popular
- Why XP fails
- Why XP is not the “silver bullet”

End of Session 4.1