eXtreme Programming

Agenda

- Introduction
- Motivation
- Foundations
- XP Practices
- Managing & Adopting XP

Introduction

XP evolved from an unusually successful project at Daimler-Chrysler (C3 Project). It is a lightweight, effective, code-centric, efficient and enjoyable discipline for teams of 2 - 10 developers.

But beware -- it is really extreme...

Going to Extremes

- Code reviews ==> Pair Programming
- testing ==> Automated tests, run may times per day
- design ==> Refactor Mercilessly
- integration ==> Continuous Integration
- simplicity ==> TheSimplestThingThatCouldPossiblyWork (YoureNotGonnaNeedIt)

The Promises of XP

- For Developers:
  - work on things that really matter
  - have a say in scheduling
  - don’t have to work alone
- For Managers:
  - get maximum value out of each dev. Week
  - see regular, concrete, observable progress
  - can change requirements affordably

The Claims of XP

- Reduces Risk
- Increases Project Responsiveness
- Improves Productivity
- Adds Fun to a Project

(Quit Laughing!)

The Features of XP

- Early, concrete, frequent feedback
- Evolutionary Project Planning
- Flexible Scheduling of Tasks
- Automated Testing
- Communication
  - orally or in code
- Collaborative Programming
  - supermen not required!

Motivation

Project Risk

- Schedule Slips
- Project Canceled
- High Defect Rate
- System Misses the Mark
- Business Requirements Change
- Staff Turnover

How XP Mitigates Risk
Schedule Slips

- Short Release Cycles
- Reduces scope of slip
- Higher priority tasks done first
  - “worst things first”

How XP Mitigates Risk
Project Canceled

- Plan smallest release that makes business sense
- ***Principle:*** "A complex system that works evolves from a simple system that works" (Grady Booch)

How XP Mitigates Risk
High Defect Rate

- Daily Automated Tests
- Catch defects early

How XP Mitigates Risk
System Misses the Mark

- Customer is part of the development team
- Requirements are continually refined and brought into focus

How XP Mitigates Risk
Business Requirements Change

- Iterative Development
- Do a little bit at a time
- Tasks easily shuffled in priority

How XP Mitigates Risk
Staff Turnover

- Programmers less frustrated because they do meaningful work, together

The XP Way

```
do
{
      perform an engineering task
      unit test
      integrate
      functional test
} while (!done);
```

The XP Way
continued

- Developers work in pairs
  - tied at the hip!
- Write unit tests first, then code
- Refactor as needed
- Integrate immediately

A Development Episode

Foundations

The Four Variables
Can't all vary independently!

- Scope
  - the most fluid variable - can be the difference between success and failure
- Resources
  - but too much is as bad as not enough
- Time
  - too much time can be a liability too
- Quality
  - not much of a variable
  - excellent vs. insanely excellent

The Four Values

- Communication
- Simplicity
- Feedback
- Courage

Communication

- Verbal or in Code
  - very few written documents
  - teams work within spitting distance
- Pair Programming
  - continuous communication!
- User Stories
  - on index cards!

Story Card

101 Union Dues

Bargaining Unit EEs have union dues withheld from the first pay period of the month. The amount varies by union, local, and in some cases varies by the individual.

If dues cannot be taken in the first pay period, they should not be taken until a UD30 transaction is received.

Priority High Risk Low

Jan

Simplicity

- Do just today’s work today
  - TheSimplestThingThatCouldPossiblyWork
  - YoureNotGonnaNeedIt
  - say “no” to Analysis Paralysis
- Don’t fear tomorrow
  - you’ll be just as successful as you are today
  - the process gives you resilience
  - tests give you confidence

Feedback

- Tasks take hours and minutes
  - not days and weeks
- Ask the System
  - i.e., run tests
- Do hard stuff first

Courage

- Don’t be afraid to redesign, refactor, throw away code
- Tests will keep you on track

Remember: “All (big) methodologies are based on fear.”

Basic XP Principles

- Rapid Feedback
- Assume Simplicity
- Incremental Change
- Embrace Change
- Quality Work

Rapid Feedback

- Get going!
  - So we don’t forget the question
  - Momentum is crucial
- For Developers:
  - **minutes**, not months
- For Managers:
  - **weeks**, not years

Assume Simplicity

- 98/2 Rule
  - the time you save on 98% of the tasks more than compensates for the 2% where raw simplicity doesn't work
- "Sufficient to the day are the tasks thereof"

Incremental Change

- Translate big changes into a series of small changes
- Design & plan as you go
- This applies to adopting XP too
  - start with what you feel comfortable with
  - but have a little courage!

Embrace Change

- Possible because we Assume Simplicity
- Solve the most pressing problem first
- Trust your tests

Quality Work

- Quality really isn't a variable
- Otherwise software development is no fun
  - Assume developers are quality-minded

The Basic Activities

- Coding
- Testing
- Listening
- Designing

Coding

- Use code for as many Software engineering purposes as possible
- Code isn’t impressed by degrees, salaries, or fancy talk
- Eschew Documentation Bulk

Testing

- Unit tests flesh-out/refine requirements
  - "I think this is what you said."
- The Untested does not exist
- "Programming + Testing" is faster than just "Programming"
- Unit tests make programmers happy
- Functional tests make users happy
- Both must be happy!

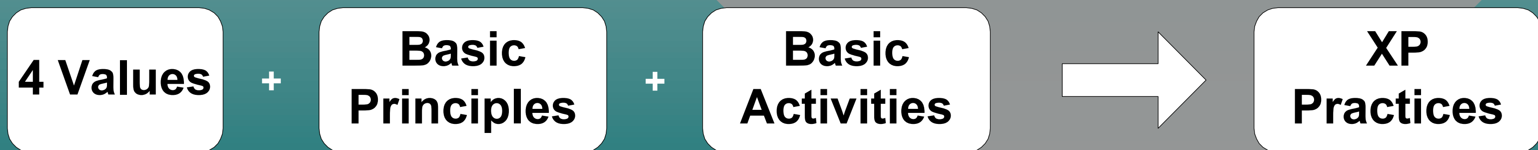
Listening

- Programmers don’t know anything
  - about the business, that is
- Ask questions
  - then listen to the answers
  - give feedback (what is hard, easy,…)

Designing

- Creating a clean internal structure
  - modular (cohesion/coupling)
    - maintainable, extensible
  - one-definition rule
- A daily activity
- Refactoring fights Entropy

XP Practices



Practices

- The Planning Game
- Small releases
- Metaphor
- Simple design
- Testing
- Refactoring
- Pair Programming
- Collective Ownership
- Continuous Integration
- 40-hour week
- On-site customer
- Coding standards

Before we Continue…

- Keep in mind that the Practices work together
- The weakness of one is covered by the strengths of others

The Planning Game

- "Quickly determine the scope of the next release by combining business priorities and technical estimates. As reality overtakes the plan, update the plan."
- "Neither business considerations nor technical considerations should be paramount. Software development is always an evolving dialog between the possible and the desirable."
- Customer owns the plan

The Planning Game
~ continued ~

- Business people decide about:
  - scope
  - priority
  - releases
- Technical people decide about:
  - estimates
  - process
  - detailed scheduling

Short Releases

- "Put a simple system into production quickly, then release new versions on a very short cycle."
- Plan a month or two at a time
- Focus only on the current release
- Implement the highest priority features first

Metaphor

- A simple, shared story that guides development
  - architecture from 30,000 feet
- "The system is…
  - a spreadsheet"
  - like a desktop"
  - like a pension calculation system"

Simple Design

- The system should be as simple as possible at any given moment.
- Extraneous complexity is removed as soon as it is discovered
  - fewest possible classes, methods
  - no duplication of logic
  - *attitude:* when you can't delete any more, the design is right

Testing

- The key to confidence
- Programmers continually write and run automated unit tests
  - for things that could possibly break
- They must pass 100% for work to continue
- Customers write functional tests to validate each user story

Refactoring

- Restructuring a system without changing its behavior
  - to remove duplication (e.g., combine methods, objects)
  - simplify design, etc.
  - promotes reliability, reuse
  - the "long term" takes care of itself
- Difficult without automated tests
- Do it when you're rested and fresh
  - which should be often!

Pair Programming

- All production code is written with two programmers at one machine
  - discourages interruptions
- Take turns driving
  - the driver implements (thinks tactically)
  - the observer thinks strategically (i.e., does the worrying:-)
- Teams can change often

Collective Ownership

- "Whoever finds a snake, kills it"
- Anyone can change code anywhere, anytime
  - to add value (not to change curly braces)
- Pair programming makes this feasible
  - always 2 brains making a joint decision

Continuous Integration

- Integrate/build many times daily
  - every time a task is completed
- Consider having a separate integration machine
- Test/debug until the system is 100% correct

40-hour Week

- Work no more than 40 hours a week as a rule
  - never work overtime a second week in a row
- Overtime is a symptom of a project in serious trouble
  - and the overtime won’t fix it
- Tired programmers make bad code
- Take a 2-week vacation every year

On-site Customer

- Have a real, live user on the team, available to answer questions
  - someone who will really use the system in practice
  - they can write functional tests, and make small-scale decisions
- Isn’t the system important enough to dedicate a real user to it?
  - if not, don’t build it
  - they can still get some real work done

Coding Standards

- A tool of communicating through code
- Avoids “curly brace” wars
- Developers will give up their religious tenets for an enjoyable, successful project

Managing & Adopting XP

Management Strategy

- Metrics
- Coaching
- Tracking
- Intervention

Metrics

- Completion Ratio (% functionality)
- Big Visible Chart (updated weekly)
- Project Velocity (estimated/real time ratio)

Coaching

- Technical management
- Coach gets everyone to make good decisions
- Does very little development
  - available as a partner
  - mentors others in technical skills
  - interfaces to upper management

Tracking

- Gathers metrics
  - completion statistics
  - once or twice a week
- Enforces the Plan
  - the hard part
  - planning is always emotional

Intervention

- When an unpopular decision is needed
- Point out the need for change
  - in the team's process, for example
  - kill the project if called for
- Personnel changes
  - better done sooner than later
- Humility required:
  - "I don't know how I let it get like this, but now I have to do XXX."

XP Roles

- Programmer (knows “how”)
- Customer (knows “what”)
- Tester (verifies quality)
- Tracker (the “conscience”, “historian”)
- Coach (the calm, responsible, manager)
- Consultant (rarely needed)
- Big Boss (leader, instills courage, insists on completion)

Adopting XP

- Incrementally of course
  - 1. Pick your worst problem
  - 2. Solve it the XP way
  - 3. When it's no longer the worst, go to 1.
- Starting Places
  - Testing
  - Planning Game

Read the Book

- We’ve covered about 50%
- Kent Beck, eXtreme Programming Explained, A-W, 1999.
- http://www.XProgramming.com
- http://c2.com
  - Extreme Programming Roadmap

Developer's Summary

- Pairs of programmers work together
- Development is driven by tests
  - test first, then code
- Pairs make tests run, and evolve the system (refactor)
  - don't let sun set on bad code
- Integration immediately follows development

The Real Summary

- Best Practices are Good Things
- XP takes them to the MAX
- XP is extreme
  - but extreme measures are often called for
- Evidence in favor of XP is mounting
- Adopt what you like
- Good luck!