

Project Management Course Book

T++ Technical Skills Training Program

CUNY Institute for Software Design & Development (CISDD)

New York Software Industry Association (NYSIA)

Aron Trauring

Class 9 November 2nd, 2004

Class Agenda

1:30-3:00 Extreme Programming Part I (Presentation)

3:00-3:15 Break

3:15-4:45 Extreme Programming Part II (Presentation)

4:45-5:00 Break

5:00-5:30 Exercise 4 — Compare and Contrast: RUP vs. XP

Extreme Programming

Historical Background

- Kent Beck and Ward Cunningham (Tektronix research)
- CRC cards
- Design Patterns (Smalltalk)
- Cunningham invented the Wiki
- C3 payroll project at Chrysler (Martin Fowler, Ron Jeffries)

Key Practices

- On-site customer as part of team
- Extreme avoidance of up-front design work
- Automated acceptance tests / test-driven development
- Pair programming

Classification

- Low on ceremony
- “Extreme” — hard to have too much of a good thing
- Explicit methods for programmers
- From C1 to E20 — small project teams with delivery dates under one year

Characteristics of XP

- IID method with very short iterations (one-two weeks)
- Stresses customer satisfaction
- Reducing risk through discipline and rapid code production
- Highly communication- and team-oriented
- Detailed work-products only for code — stress on oral communication (minimize documentation overhead)
- Many inter-related practices that need to be used as a whole in order to work properly

Core XP Practices — Requirements

1. Release Planning Game
2. Short Releases
3. On-site Customer / Whole team Together
4. Acceptance Testing

Release Planning Game — Description

- Define the scope of the next operational release with maximum value to the customer
- Half-day to one-day session
- Customers write story cards to describe features
- Developers estimate them
- May be left-over story cards
- Customer chooses cards for next release

Release Planning Game — Story Cards

- Paper index cards
- User Stories — Brief feature requests, fixes or non-functional requirements
- *Not* use cases
- Just a “promise to talk” with customer for details (goes together with “whole team together”)
- Minimalist approach — just a few words
- Granularity 2-10 days

Release Planning Game — Release Length

- Choose date and choose cards to fit or
- Choose cards and set date to fit
- Choice and decision is made by customer
- Programmers give time input

Short Releases — Description

- Evolutionary delivery
- Not the same as iterations — each release has multiple iterations
- Gives customer ability to kill project early — reduces risks

On-site Customer / Whole Team Together — Description

- Customer involvement is good
- Be extreme and have one or more customers sit more or less full-time with the team
- Subject matter experts
- Empowered to make decisions about requirements
- Common project room

On-site Customer — Customer Contribution

- Detailed explanation of features
- Planning Game participation
- Clarifications as programming progresses
- Writing acceptance tests in collaboration with programmers

Whole Team Together — Purpose

- Communication is good
- Be extreme so have everyone sit together in one room

Acceptance Testing — Description

- Testing is good
- Be extreme and test everything always
- All features *must have* automated acceptance tests
- All tests (unit and acceptance) must run with a binary pass/fail result
- No human inspection of individual tests should be required
- Customer tests — acceptance test written in collaboration with customer
- Customer tests — define a testable statement of what acceptance means

Core XP Practices — Design

1. Iteration Planning Game
2. Simple Design
3. System Metaphors
4. Frequent Refactoring

Iteration Planning Game — Description

- Customer choose stories to implement in up-coming iteration
- Determine associated tasks in task list
- Allocate tasks

Iteration Planning Game — Task List

- A list of tasks for each story in the iteration
- On whiteboard or cards
- Programmers volunteer for tasks

Iteration Planning Game — Time Allocation

- Programmer gives estimate for task in Ideal Engineering Hours (IEH)
- IEH — uninterrupted, dedicated, focused time to complete a task
- Short iteration is good
- Be extreme and make iterations one to two weeks
- Tasks taking more than two days are re-factored

Iteration Planning Game — Accepted Responsibility (“Only by Volunteering”)

- Tasks are not assigned
- People choose or volunteer for tasks
- Higher degree of commitment
- Greater satisfaction
- Self-organizing

Simple Design — Description

- Avoid speculative design for possible future changes
- Avoid creating generalized components that are not immediately required
- Avoid duplicate code
- Have a minimal set of classes methods
- Be easily comprehensible

Simple Design — Very Light Modeling

- Start programming very early
- Extreme avoidance of up-front design
- More than 10-20 minutes is considered excessive

Simple Design — CRC Cards

- Class — Responsibilities — Collaboration
- Class — nouns, objects — written on top of card
- Responsibilities — verbs, messages — written on left side
- Collaboration — interacting objects — written on right side next to responsibilities
- CRC session — simulating the system by talking about which objects send messages to other objects

System Metaphors — Description

- Use memorable metaphors to capture the overall system or subsystem
- Describe key architectural themes

Frequent Re-Factoring — Description

- Simplify large design elements
- Simplify fine-grained code
- Clean design and code without changing functionality
- Be extreme about re-factoring
- Goal: minimal, simple, comprehensible code

Frequent Re-Factoring — Process

- Small change steps
- Verify through testing
- IDE re-factoring tools where available

Core XP Practices — Implementation

1. Pair Programming
2. Team Code Ownership
3. Test-Driven Development
4. Continuous Integration
5. Frequent Re-Factoring
6. Coding Standards

Pair Programming — Description

- Code reviews are good
- Be extreme — do code reviews in real-time
- All production code is done by two programmers at one computer
- Take turns at using keyboard
- Observer does real-time code review
- Pairs may change frequently

Pair Programming — Benefits

- Team productivity is not a function of number of people at individual workstations
- Cross-learning
- Peer pressure to be more disciplined
- Greater output from mutual encouragement
- Defect reduction from real-time code-review
- Stamina and insight to carry on when one gets stuck

Team Code Ownership — Description

- Any pair of programmers can improve any code
- Entire team collectively responsible for all code
- Whoever spots problem fixes it

Team Code Ownership — Benefits and Pitfalls

- Elevate bottleneck of change requests — whoever is available does it
- Pairing partner helps brings another set of eyes to problem
- Common adherence to coding standards necessary

Test-Driven Development — Description

- Unit tests written for almost all code
- Test written before the code to be tested
- Ties into continuous integration
- Customer input on developing tests

Continuous Integration — Description

- All checked-in code is continuously re-integrated and tested on a separate build machine
- Automated 24/7 process loop of compiling, running all unit tests and all or most acceptance tests
- Something breaks added to issue list and open to being fixed by anyone on team

Coding Standards — Description

- Everyone needs to follow same coding style
- Necessary for other practices to work properly
- Core XP Practices — Miscellaneous

Other XP Practices

- Embrace change — respond quickly at any point in the process
- Sustainable Pace — no overtime, “family-friendly” work-days
- “Just enough” documentation — time better spent on programming
- “Fail early, fail often” — start with riskiest parts first to allow killing project early with minimal overhead
- Metrics — simplest ones that work — number of completed stories, tasks, bugs
- Visible wall graphs — collected metrics updated daily on wall
- Tracker — human collector of completion metrics through a walk-about
- Incremental Infrastructure — only built the infrastructure that is necessary when it is needed
- Daily Stand-Up Meeting

Core XP Work-Products

- Story cards
- CRC cards
- Task lists
- Visible graphs
- Sketches

Core XP Values

- Communication — Whole Team, On-site Customer, Planning Games, Pair Programming
- Simplicity — “Do the simplest thing that could possibly work” — Simple Design, Refactoring, Incremental Infrastructure
- Feedback — Test-First Development, Continuous Integration, Daily Tracker, Short Iterations, Frequent Releases
- Courage — Extreme and radical change possible because of combination of all practices

XP Lifecycle

1. Exploration
2. Planning
3. Iterations to Release
4. Deployment / Product
5. Repeat Game

Exploration — Purpose

- Enough well-estimated story cards for first release (in rough draft format)
- Feasibility ensured

Exploration — Activities

- Exploratory proof of concept
- Exploratory proof of technology
- Story card writing and estimates

Planning — Purpose

- Agree on date and stories for first/next release

Planning — Activities

- Release Planning Game — completion of rough drafts
- Define and schedule next release

Iteration to Release — Purpose

- Implement a tested system ready for release

Iteration to Release — Activities

- Iteration Planning Game — stories picked and broken into tasks based on current status and priorities
- Review of total estimated time may adjust stories implemented
- Programmers may not be overworked
- Repeat time-boxed iterations until ready for release
- Continuous integration

Deployment / Product — Purpose

- Operational deployment or packaging

Deployment / Product — Activities

- Documentation
- Training
- Marketing

Repeat Game — Purpose

- Enhance or Fix
- Build major release

Repeat Game — Activities

- Go back to beginning of life-cycle

XP Roles

1. Customer
2. Development
3. Management
4. Other

XP Roles — Customer

- Writes stories and acceptance tests
- Picks stories for release and iteration

XP Roles — Development

- Programmer — writes, tests and designs code, re-factors, identifies tasks and provides estimates
- Tester — helps customer define and develop tests

XP Roles — Management

- Coach (PM) — process conscience, process customizing, intervention and teaching
- Tracker — collects metrics, tells progress and provides feedback on poor estimates

XP Roles — Other

- Technical consultants
- Coaches
- Administrative support

Throughput Accounting

- Inventory is story points — can be standardized and measured over time in an organization
- Throughput is dollar value of implemented story points
- Inventory can be tracked in both an iteration and a release
- Highly effective in capping inventory because of frequent releases
- Minimal up-front investment in inventory — investment made in course of production, capping risk
- Initial investment low also means cost of abandonment low
- Constant testing allows for pipelining
- Velocity — Lead Time — emphasis on feedback loops rapidly leads to convergence
- XP closely follows throughput model and can easily be modified to add true metrics and tracking

Criticisms of XP

- Near impossibility of getting on-site customers — often resort to “customer proxies” or “customer on call”
- Resistance to implementing the “whole package”
- Too little up front design may be dangerous
- No standard post-project documentation

In Defense of XP

- Frequent releases mean benefits come early and come often
- Little design does not mean no design — design is a continuous process
- Practical, high-impact, easily sustainable techniques
- Focus on measurement
- High level of customer involvement and communication
- High level of inspection and testing reduces defects

Wrong way to use XP

- Pick and choose practices — “do all of XP before trying to customize”
- Develop use cases instead of oral communication with customer
- Not writing test firsts — tests influence design
- Not letting customer decide — customer *must* design acceptance test and choose stories in planning games
- QA separate – QA part of whole team and works with customer on acceptance test
- Pair programming done wrong — change frequently, mix experienced with novice

Exercise 9 — Compare and Contrast RUP vs. XP

- a. Define each in one sentence.
- b. List what they have in common.
- c. List how they differ.

2. Discuss with group and make an agreed upon definition for a and list of five-ten key points raised for b and c.

3. Someone different presents to the class.