Agile Software Development
Methods and Practices

Laurie Williams
laurie.williams@gmail.com

Agenda

• Overview of Agile Methods
  • Extreme Programming
  • Scrum
  • Feature Driven Development
Extreme Programming Practices: Primary

Sit Together

- Develop in an open space big enough for everyone.
- Communication drops off shortly if the walk to a colleague > 10 meters (Allen).
- Warroom doubles productivity (Olsen and Olsen).
- Have small, private spaces nearby → “cave and commons”.

Whole Team

- 1st class cross-functional team

Informative Workspace

Corkboard from: http://www.xprogramming.com/xpmag/BigVisibleCharts.htm
Energized Work

- Work only as many hours as you can be productive and only as many hours you can sustain.

- Tired developers make more mistakes, which slows you down more in the long run (remove value from product).

- If you mess with people’s personal lives (by taking it over), in the long run the project will pay the consequences.

Overtime: Extended vs. Sporadic

- Lack of energy
- Lack of creativity
- Defect fixing due to quality decline
Stories

• Customer-visible/customer-valued functionality

Title: Delete a Recipe
As the owner, I would like to be able to delete any of the existing coffee recipes so I can remove unpopular coffee types to make room for other selections.
Weekly cycle [Iteration]

- Highest priority stories in “time boxed” weekly increments

Quarterly Cycle [Release]

- Timeboxed
- As small as possible, but still delivering business value
- Get customer feedback early and often
Ten-Minute Build

- Automatically build the entire system and run all tests in 10 minutes
- Feedback, feedback!

Continuous Integration

- Writes up unit test cases and code for a task (part of a user story)
- All unit tests for new functionality pass
- Integrate new functionality to code base
- ALL unit test cases for code base pass
- Should happen once or twice a day.
- Prevents IntegrationHell [integration could take longer than programming]

http://cruisecontrol.sourceforge.net/index.html
Test-driven Development

Incremental Design

• No Big Design Up Front (BDUF)

• Knowledge-based design – the most effective design is in light of experience

• “Do The Simplest Thing That Could Possibly Work”
• “You Aren’t Gonna Need It” (YAGNI)

• Refactoring: Improve the design of existing code without changing functionality
  • Simplify code
  • Opportunity for abstraction
  • Remove duplicate code

• Relies on testing to ensure nothing breaks in the process of refactoring
Empirical Studies of XP Teams

The use of a specified subset of XP practices leads to an improvement in . . .

<table>
<thead>
<tr>
<th>Hypothesis</th>
<th>IBM</th>
<th>Sabre-A</th>
<th>Sabre-P</th>
<th>Tekelec</th>
</tr>
</thead>
<tbody>
<tr>
<td>pre-release quality</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>N/A</td>
</tr>
<tr>
<td>post-release quality</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>programmer productivity</td>
<td>Yes</td>
<td>Yes</td>
<td>Similar-Higher</td>
<td>No</td>
</tr>
<tr>
<td>customer satisfaction</td>
<td>Yes</td>
<td>N/A</td>
<td>N/A</td>
<td>Neutral-satisfied</td>
</tr>
<tr>
<td>team morale</td>
<td>Yes</td>
<td>N/A</td>
<td>N/A</td>
<td>N/A</td>
</tr>
</tbody>
</table>

Scrum

Feature = “sashimi”
Every slice must be complete.

Sprint Backlog: Feature(s) assigned to sprint.

New functionality is demonstrated at end of sprint.

Every 24 hours:
Sprint 15 minute daily meeting,
Teams member respond to basics:
1) What did you do since last Scrum Meeting?
2) Do you have any disturbed?
3) What will you do before next meeting?

Product Backlog:
Prioritized product features desired by the customer

Purposes of Daily Scrum

• Provide coordination mechanism
  – Once per day, hear where everyone is

• Each team mate provides verbal commitment to his/her peers

• Who attends?
  – Developers, testers, product owner, ScrumMaster, others as available: UI, doc, etc.
Daily Scrum

• ~10-15 minutes
• Same Scrum time, same Scrum location
• Chickens and pigs
  – Chickens are “involved” and only listen, stand on periphery
  – Pigs are “committed” and talk
• Only three questions, one person at a time
  – What did I do yesterday?
  – What will I do today?
  – What’s standing in my way?
The Scrum Master

- Most often the project manager
- Makes sure people are following Scrum rules
  - e.g. no new functionality mid Sprint
- Remove obstacles/barriers
- Produce necessary reports and make visible
- Make sure things are progressing

Scrum of Scrums
Scrum of Scrums Questions

- What has your team done since we last met?
- What will your team do before we meet again?
- Is anything slowing your team down or getting in their way?
- Are you about to put something in another team’s way?

... focusing especially on areas of overlap and integration
... technical person who can explain current issues; can change attendee

Scrum of Scrum of Scrums

http://www.scrumalliance.org/articles/46-advice-on-conducting-the-scrum-of-scrums-meeting
Burn-down Chart

Feature Driven Development

http://www.agilemodeling.com/images/lifecycleFDD.gif
FDD Best Practices

• Domain Object Modeling
• Developing by Feature
• Individual Class (Code) Ownership
• Feature Teams
• Inspections
• Regular Builds
• Configuration Management
• Reporting/Visibility of Results (next slide . . .)

Feature teams: http://www.methodsandtools.com/archive/FDD2.gif

Reporting/Visibility of Results

• Design by Feature
  – Domain Walkthrough (1%)
  – Design (40%)
  – Design Inspection (3%)
• Build by Feature
  – Code (45%)
  – Code Inspection (10%)
  – Promote to Build (1%)
• Binary done/not done

Parking lot: http://www.featuredrivendevelopment.com/files/images/img_1b37fa5b23d3179fae67c65c6152c9db.preview.png
Review & Check Understanding

• Draw a class diagram to illustrate the relationship between software development models, software development processes, software development practices. Be sure to clearly demonstrate any inheritance and/or aggregation. Show any multiplicity.
  – All models have similarities and differences.
  – All but “code and fix”, chaotic processes have a process model.
  – All processes involve requirements analysis, design, coding, testing, etc.
  – There are a variety of ways to do requirements, analysis, design, coding, etc. Each processes uses the way it wants to accomplish the step.