CS 335 — Software Development

Agile Methodologies

Jan 22, 2014

Principles/values

Rapid feedback	Small initial investment
Assumption of simplicity	Concrete experiments; test infection
Incremental, expected change	Responsibility

Invest in the design of the system every day. Strive to make the design of the system an excellent fit for the needs of the system that day. When your understanding of the best possible design leaps forward, work gradually but persistently to bring the design back into alignment with your understanding. (Beck, pg 51–52)

Practices/activities

- Whole team
- Metaphor
- The planning game
- Small releases
- Simple design
- Testing
- Refactoring
- Pair programming
- Collective ownership
- Continuous integration
- 40 hour work week
- On-site customer

Metaphor

Plan using units of customer-visible functionality. "Handle five time the traffic with the same response time." "Provide a two-click way for users to dial frequently used numbers." As soon as a story is written, try to estimate the development effort necessary to implement it.
Software development has been steered wrong by the word "requirement," ... The word carries a connotation of absolutism and permanence, inhibitors to embracing change. [Not all requirements will be found truly to be obligatory.] (Beck, pg 44)

The planning game

Plan work a week at a time. Have a meeting at the beginning of every week. During the meeting:

1. *Review the progress to date*
2. *Have the customers pick a week's worth of stories to implement this week.*
3. *Break the stories into tasks. Team members sign up for tasks and estimate them. (Beck pg 46)*

Plan work a quarter at a time. Once a quarter reflect on the team, the project, its progress, and its alignment with larger goals.
Identify bottlenecks; plan the theme for the quarter; pick a quarter's worth of stories to address those themes. (Beck, pg 47)

Testing

Write a failing automated test before changing any code. Test-first programming addresses many problems at once:

- *Scope creep—It's easy to get carried away programming and put in code "just in case." By stated explicitly and objectively what the program is supposed to do, you give yourself a focus for your coding.*
- *Coupling and cohesion—If it's hard to write a test, it's a sign that you have a design problem, not a testing problem. Loosely coupled, highly cohesive code is easy to test.*
- *Trust—It's hard to test the author of code that doesn't work. By writing clean code that works and demonstrating your intentions with automated tests, you give your teammates a reason to trust you.*
- *Rhythm—It's easy to get lost for hours when you are coding. When programming test-first, it's clearer what to do next: either write another test or make the broken test work. (Beck pg 50–51)*

On-site customer

Make people whose lives and business are affected by your system part of the team. Visionary customers can be part of quarterly and weekly planning. . . . If this is the kind of customer who encounters problems six months before the rest of the market, making the system they want can put you ahead of your competition. (Beck, pg 61)

Scrum: The product backlog

The Product Backlog is an ordered list of everything that might be needed in the product and is the single source of requirements for any changes to be made to the product. A Product Backlog is never complete. The earliest development of it only lays out the initially known and best-understood requirements. The Product Backlog evolves as the product and the environment in which it will be used evolves. The Product Backlog is dynamic; it constantly changes to identify what the product needs to be appropriate, competitive, and useful. As long as a product exists, its Product Backlog also exists. The Product Backlog lists all features, functions, requirements, enhancements, and fixes that constitute the changes to be made to the product in future releases. Product Backlog items have the attributes of a description, order, estimate and value. (Scrum Guide, pg 12)