Chapter 3 – Agile Software Development

Topics covered

- ✧ Agile methods
- ✧ Plan-driven and agile development
- ✧ Extreme programming
- ✧ Agile project management
- ✧ Scaling agile methods

Need for rapid software development

- ✧ Changing businesses environment
  - New opportunities and technologies
  - Changing markets, new competitors
- ✧ Companies will trade off quality for faster deployment
- ✧ Requirements are never stable and hard to predict

- ✧ Waterfall methods are inadequate here:
  - Process is prolonged when there is too much change
  - Product is out of date when it's delivered
- ✧ 1990's: Agile processes were developed in response to these problems.

Rapid software development

- ✧ Goal: produce useful software quickly
- ✧ Form of incremental development:
  - Specification, design and implementation are inter-leaved
  - Customers evaluate versions
- ✧ Minimal process documentation
  - Minimal user requirements
  - No detailed design specifications
- ✧ Use of development tools (IDE, UI development tools)
- ✧ Very Small Increments

3.1 Agile methods

- 1980s software design methods
  - careful project planning
  - formal methods.
- Large systems vs. smaller business applications
- 1990s agile processes developed
- The aim of agile methods is to
  - Reduce overhead in the software process
  - Avoid rework when responding to change

Agile manifesto

- We have come to value:
  - Individuals and interactions over processes and tools
  - Working software over comprehensive documentation
  - Customer collaboration over contract negotiation
  - Responding to change over following a plan
- That is, while there is value in the items on the right, we value the items on the left more.

www.agilealliance.org

Some agile methods

- Extreme Programming (XP)
- Scrum
- Crystal methods
- Evo
- Adaptive Software Development
- Dynamic Solutions Delivery Model (DSDM)
- Feature Driven Development
- Agile modeling methods
- Agile instantiations of RUP

Some principles of agile methods

- Customer Involvement
  - should be closely involved in development process
  - prioritize requirements and evaluate iterations
- Incremental Delivery
  - small increments, rapid delivery
  - working software is primary measure of success
- People not process
  - value+use particular skills of dev team members
  - let them develop their own processes
- Embrace Change
  - expect change, design the process to accommodate it
- Maintain Simplicity
  - in software and process, eliminate complexity

Agile method applicability

- Small to medium product development
- Custom system development when
  - Committed customer
  - Few rules and regulations
- Difficult to scale to large systems
- Not necessarily for security- or safety-critical systems

Problems with agile methods: The principles are difficult to realize

- Customer commitment
  - Must be willing and able to spend time on project
- Suitability of development team members
  - Some team members may not like intense involvement
- Difficulty prioritizing changes for each increment.
  - Multiple stakeholders may be in conflict
- Maintaining simplicity requires extra work.
  - May require scheduling extra time for refactoring
- Large organizations and formal processes
  - Trend has been towards formal processes, not away from them

Agile methods and software maintenance

- Are systems that are developed using an agile approach maintainable?
  - issue: very little documentation
  - issue: original development team
- Can agile methods be used for evolving a system regardless of how developed?
  - Agile methods designed for managing change.
  - Customer involvement

3.2 Plan-driven and agile development

- Plan-driven development
  - Separate, sequential development stages.
  - Iteration occurs within stages.
  - Waterfall or planned increments.
- Agile development
  - Specification, design, implementation in each cycle
  - Iteration occurs across stages.
  - May have elements of formal processes.

Should your approach be plan-driven or agile?
Technical, human, organizational issues

1. Is it important to have a very detailed specification and design before moving to implementation?
2. Is an incremental delivery strategy, where you deliver the software to customers and get rapid feedback from them, realistic?
3. How large is the system that is being developed (and consequently, the development team)?
4. What type of system is being developed? Real time system with complex timing requirements? Safety-critical?
5. What is the expected system lifetime? Long-lifetime?

Should your approach be plan-driven or agile?
Technical, human, organizational issues

6. What technologies are available to support system development? Do you have good tools?
7. How is the development team organized? Distributed or outsourced?
8. Are there cultural or organizational issues that may affect the system development? Is the team old-school?
9. How good are the designers and programmers in the development team? Are they highly skilled?
10. Is the system subject to external regulation?

3.3 Extreme programming (XP)

- ✧ Best-known and most widely used agile method.
- ✧ Kent Beck, 2000

- ✧ Pushing recognized good practice to the extreme:
  - More customer involvement is good so bring customers onsite.
  - Code reviews are good, so do constant code reviews via pair programming
  - Testing is good, so write tests before writing the code.
  - Short iterations and early feedback are good, so make iterations only 1 or 2 weeks.

XP: 12 core practices

- ✧ Planning Game(s)
  - Major Release: Define scope, customer writes story cards
  - Iteration: customer picks cards, developers pick tasks
- ✧ Small, frequent releases
  - 1-3 weeks
- ✧ System metaphors
  - used to describe architecture in easily understood terms
- ✧ Simple Design
  - No speculative design, keep it easy to understand
- ✧ Testing
  - Automated, test-driven development
- ✧ Frequent Refactoring
  - Cleaning code without changing functionality

XP: 12 core practices

- Pair Programming
  - One computer, one typist, other reviews, then swap
  - Rotate partners
- Team Code ownership
  - Any programmer can improve any code,
  - Entire team is responsible for all the code,
- Continuous Integration
  - all checked in code is continually tested on a build machine
- Sustainable Pace: No overtime
- Whole Team Together
  - Developer and customer in one room
- Coding Standards
  - Adopt a common programming style

XP reflects agile principles

- Customer involvement:
  - Full-time, on-site customer.
- Incremental delivery:
  - Small, frequent releases.
- People not process:
  - Pair programming
  - Collective ownership
  - Sustainable pace
- Embrace Change
  - Quick releases to customer for feedback
- Maintaining simplicity
  - Maintaining simple code, simple designs

Requirements (The planning game)

- Story Cards
  - Customer writes brief feature request.
- Task List
  - Implementation tasks
  - Written by Developer(s)
  - After discussing story card with Customer
- Customer chooses the story cards
- Cards can be changed or discarded
- Requirements specification depends on oral communication.

Requirements: example story cards

- From the MHC-PMS case study:

> Doctor wants to prescribe medicine for a patient at a clinic.

- Or that might take too long to implement

> Doctor wants to change current prescription for a patient at a clinic.

> Doctor wants to prescribe new medicine by drug name for a patient at a clinic.

> Doctor wants to prescribe new medicine by formulary for a patient at a clinic.

Task List example

- From the story card:

> Doctor wants to prescribe new medicine by drug name for a patient at a clinic.

- List of Implementation Tasks
  - Implement drug list/database
  - Implement search for a drug by name
  - Add/modify GUI for doctor to access search
  - Implement get+check dosage
  - Implement save prescription for patient
  - etc.

XP and anticipating change

- Conventional wisdom:
  design for change by using very general designs.
  - Claim: this reduces costs later in the life cycle.
- XP maintains: this is not worthwhile
  - Changes cannot be reliably anticipated.
- XP proposes: Constant code improvement (refactoring)
  - make changes easier when they have to be implemented

Refactoring

- Restructuring an existing body of code, altering its internal structure without changing its external behavior
- Advantages:
  - Easier to understand, easier to add new functionality
- Examples:
  - Breaking up a large class into two or more classes.
  - Moving methods/functions to different classes.
  - Renaming attributes and methods to make them easier to understand.
  - Replacement of inline code with a call to a method/function.

3.3.1 Testing in XP

- No requirements document implies no external testing.
- Test-first Development
  - Tests are written before the task is implemented.
  - Forces developer to clarify the interface and the behavior of the implementation.
  - Tests are based on user stories and tasks, one test per task.
- Customer involvement.
  - Customer helps write tests, making them acceptance tests.
  - Thus acceptance testing is incremental.
- Test automation is crucial (no external testing)
  - No interaction required
  - Results checked automatically and reported.
- Regression testing is automated.
  - ensure no existing functionality got broken

Test driven development example

- Task: implement a Money class in Java to support multiple currencies, adding money, etc.
- Developer writes a Money test class:

```
public class MoneyTest extends TestCase {

  public void testSimpleAdd() {
    Money m1 = new Money(12,"usd");
    Money m2 = new Money (14, "usd");
    Money expected = new Money(26, "usd");
    Money result = m1.add(m2);
    assertEquals (expected, result);
  }
}
```


3.3.2 Pair programming

- Programmers work in pairs at one workstation.
- Pairs rotate on different tasks.
- Advantages:
  - Helps develop common ownership of code.
  - Informal review process.
  - Encourages refactoring.
- How productive is it?
  - Results vary, hard to measure full effect.

3.4 Agile project management

- What is Project Management?
  - job of ensuring software is delivered on time within the budget.
- Standard approach is plan-driven, project manager decides:
  - what should be delivered,
  - when it should be delivered and
  - who will work on the development of the project deliverables
- This approach does not work for Agile projects.
  - "what should be delivered" is not known up front

3.5 Scrum

- Emphasizes a set of project management values and practices.
- Easy to combine with other methods
- XP has adopted many Scrum practices
- Hands-off approach:
  - No project manager or team leader
  - Team is empowered to make own decisions
- Three phases:
  - Outline planning: stakeholders enter features in product backlog, choose the product owner.
  - A series of sprint cycles, each develops one increment
  - Project closure phase

The Sprint cycle

- Sprints are fixed length
- Sprint planning
  - Stakeholders select features for next sprint
  - Scrum team and product owner meet to plan work
- Scrum daily meetings
  - Stand-up meeting, 15-20 minutes
  - Each member gives progress report, future plans, and problems
  - Keeps sprint backlog up to date
- Scrum master
  - Makes sure team is not interrupted
  - Manages communication with customer and management
  - Resolves team “blocks” asap.
- Sprint review: Product Demo

Scrum in practice

- Used successfully for developing telecommunication software (see book).
- Can it be scaled to larger, even distributed teams?