Agile Software Development

Chapter 3

Agile Software Development
in the textbook

- 3.1 Agile methods
- 3.2 Plan-driven and agile development
- 3.3 Extreme programming (XP)
  - A well known agile method
- 3.4 Agile project management (Scrum)
- 3.5 Scaling agile methods
  - We'll read and discuss the paper by Lindvall, et.al. instead of going over this section.

The need for rapid software development

- Changing business environments
  - 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  - Product is out of date when it's delivered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 interleaved
  - Customers evaluate versions
  - Very small increments (2-3 weeks)
- Minimal process documentation
  - Minimal user requirements documents
  - Lack of detailed design specifications
- Favor use of development tools:
  - IDEs, UI development tools, etc.

3.1 Agile methods

- 1980s software design methods:
  - careful project planning
  - controlled and rigorous development processes
- Large systems vs. smaller business applications
  - traditional methods had too much overhead for smaller apps
- 1990s: agile processes were developed
- The aim of agile methods is to
  - Reduce overhead in the software process
  - Avoid rework when responding to change
- Best suited to rapidly changing requirements

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.
- Website: 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
(derived from the manifesto)

- 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 pros and cons

- Good for small to medium product development
- Good for custom system development when
  - Committed customer
  - Few rules and regulations
- Difficult to scale agile methods to large systems
  - Agile methods emphasize small teams
- Not necessarily for security- or safety-critical systems
  - These depend on thorough analysis, documentation

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 like 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: continuity of original development team
- Can agile methods be used for evolving a system developed using another method?
  - Agile methods designed for managing change.
  - Customer involvement may be difficult
  - May need to refactor original code base

3.2 Plan-driven and agile development

- Plan-driven development
  - Separate, sequential development stages.
  - Output from one stage is used to plan the next stage
  - Can be incremental: each increment is planned up front.
- Agile development
  - Specification, design, implementation in each cycle
  - The primary output is the code itself.
  - May still have some elements of more 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? If so ...
2. Is an incremental delivery strategy, where you deliver the software to customers and get rapid feedback from them, realistic? If so ...
3. How large is the system that is being developed (and consequently, the development team)? If large ...
4. What type of system is being developed? Real time system with complex timing requirements? Safety-critical? If so ...
5. What is the expected system lifetime? If 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 dev tools? If so ...
7. How is the development team organized? Distributed or outsourced? If so ...
8. Are there cultural or organizational issues that may affect the system development? Is the team old-school? If so ...
9. How good are the designers and programmers in the development team? Are they highly skilled?
10. Is the system subject to external regulation? If so ...

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

1. Planning Game(s)
   - Major Release: Define scope, customer writes story cards
   - Iteration: customer picks cards, developers pick tasks
2. Small, frequent releases
   - 1-3 weeks
3. System metaphors
   - used to describe architecture in easily understood terms
4. Simple Design
   - No speculative design, keep it easy to understand

XP: 12 core practices

5. Testing
   - Automated, test-driven (test-first) development
6. Frequent Refactoring
   - Cleaning code without changing functionality
   - Keep the structure from degrading
7. Pair Programming
   - One computer, one typist, other reviews, then swap
   - Rotate (change) partners
8. Team Code ownership
   - Any programmer can improve any code,
   - Entire team is responsible for all the code.

XP: 12 core practices

9. Continuous Integration
   - all checked in code is continually tested on a build machine
10. Sustainable Pace:
   - No overtime, developers not overworked
11. Whole Team Together
   - Developers and customer in one room, accessible
12. 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 to implement next
- Cards can be changed or discarded
- Requirements specification depends on oral communication.

Requirements: example story cards

- From a flight-booking website

  User needs to Find Lowest Fares

- Or if the scope of that is too large for an iteration, break it down into several stories:

  User needs to find lowest fares for a one-way trip

  User needs to find lowest fares for a round-trip

  User needs to find lowest fares offered by a given airline

Task List example

- From the story card:

  User needs to find lowest fares for a round-trip

- List of Implementation Tasks
  - Implement/modify fare schedule database
  - Implement search for a flights/legs by date
  - Implement search for multi-leg flight
  - Add/modify GUI for user to access search
  - Implement save itinerary for user
  - 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

- 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, throughout development process.
  - (traditionally customer testing occurs at the end of the project.)
- Test automation is crucial
  - Testing is developer's responsibility (no external test team)
  - No interaction required: results checked automatically and reported.
  - Automatic regression testing ensures no existing functionality gets broken by a new increment or refactoring

Test driven development example

- Task: implement a Money class in Java to support multiple currencies, adding money, etc.
- Developer writes a Money test class:
  - Assumes: Money(int,string) constr, Money add(Money) method

```
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.
  - One has control of the computer
  - Other is "looking over their shoulder"
  - take turns in each role
- Pairs change partners for 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
  - change is the norm
  - But agile projects still need to make good use of resources

Scrum

- A set of project management values and practices.
  - Easy to combine with other agile methods
- Hands-off approach:
  - No project manager or team leader (only a scrum master)
  - Team is empowered to make own decisions
- Three phases:
  - **Outline planning**: where stakeholders
    - enter features/requirements in product backlog
    - choose the product owner (usually a customer)
  - A series of **sprint cycles**, each develops one increment
  - **Project closure** phase (deployment)

product backlog:
features to be implemented

The Sprint cycle

- Sprints are fixed length (often 30 calendar days)
- 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, with estimates
- Scrum master
  - Makes sure team is not interrupted
  - Manages communication with customer and management
  - Resolves team "blocks" asap.
- Sprint review: Product Demo for customers

sprint backlog:
tasks to be done

Scrum in practice

- Used successfully for developing telecommunication software (see book for details).
- The following book lists three projects that struggled for months (or years), then adopted scrum and had success within a year (often less).

Larman, Craig (2003) *Agile & Iterative Development: A Manager's Guide*. Boston, MA. Addison-Wesley

- Can scrum be scaled to larger, even distributed teams?