Agile Software Development

Chapter 3

I. The problem with traditional development processes

- Lengthy development times (one to five years)
  - Product may be out of date before it is completed
- Lack of flexibility regarding requirements:
  - Unable to cope with changing requirements
  - Requirements must be completely understood upfront
- Too much reliance on heroic developer effort
  - lots of overtime to finish on time
- Too much overhead
  - complex methodology requires detailed specifications of activities, detailed design documents, etc.
  - Much information is maintained in multiple forms

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
- Traditional methods are inadequate in this context
- 1990's: Agile processes were developed in response to these problems.

II. What are agile processes?

- Form of incremental development:
  - Very small increments (2-3 weeks)
  - Customers evaluate versions
- Minimal process documentation
  - Minimal user requirements documents
  - Lack of detailed design specifications
- Focus on human and team aspects of software development.
- Favor use of development tools:
  - IDEs, UI development tools, etc.

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 principles of agile processes

- Incremental Delivery
  - small increments, rapid delivery
  - working software is primary measure of success
- Customer Involvement, constant feedback
- People not process
  - focus on informal communication
- Embrace Change
  - expect change, design the process to accommodate it
  - incremental design: delay design decisions as much as possible
- Maintain Simplicity
  - minimal documentation, source code is the documentation
  - in software and process, eliminate complexity

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

III. 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
   - Each 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

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

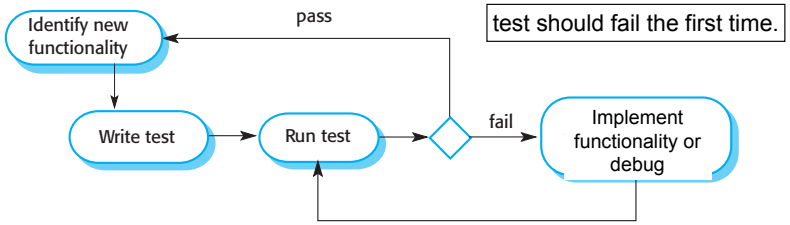
Refactoring

- 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.

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



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);
  }
}
```


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.

Project management

- What is Project Management?
  - job of ensuring software is delivered on time within the budget.
- In traditional processes the 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

IV. Agile versions of UP

- Unified Process is a hybrid process, and can be instantiated in different ways
- How to make a more agile instantiation of UP:
  - restrict the required work products (artifacts)
  - eliminate/merge some of the roles
  - add more customer involvement in the iterations
- The following paper discusses this approach:

Michael Hirsch. 2002. Making RUP agile. In *OOPSLA 2002 Practitioners Reports* (OOPSLA '02). ACM, New York, NY, USA, 1-ff. DOI=10.1145/604251.604254 http://doi.acm.org/10.1145/604251.604254

V. 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
- Consists of roles, events, and artifacts
- Iterations are called **sprints**
  - one month or less
  - **time-boxed:** duration is constant, features are dropped to meet the deadline.

Scrum: roles

- Product owner
  - represents the voice of the customer
- Development team
  - 3 to 10 developers who produce the software
- Scrum master
  - keeps team on track, makes sure Scrum is followed
  - Makes sure team is not interrupted, resolves blocks
  - intermediary between developers and management/stakeholders
- Stakeholders and Managers
  - **Stakeholders:** customers/users/etc.
  - **Managers** development organization administrators

Scrum: events

- Sprint planning meeting
  - Scrum team and product owner meet to decide what will be implemented in the next sprint
- Daily scrum
  - Stand-up meeting, 15-20 minutes
  - Each member gives progress report, future plans, and problems
- Sprint review
  - held at end of sprint
  - product demo by developers, answer questions
  - entire team decides what to do next
- Sprint retrospective (after sprint review)
  - team members discuss what they learned from sprint review

Scrum: artifacts

- Product backlog
  - ordered list of all remaining requirements
  - prioritized by product owner
- Sprint backlog
  - ordered list of tasks that need to be done for current sprint
  - short (4-16 hours), chosen by developer
- Increment
  - sum of all requirements implemented so far (the release)
- Burn down chart
  - frequently updated, publicly displayed chart
  - shows remaining work from sprint backlog

VI. Choosing a process

- No process fits all projects.
- Must adjust the process to
  - the project
  - the organizational culture
  - the people participating in it
- Requires being familiar with
  - the characteristics of the project (size, stability of requirements, criticality of requirements).
  - the characteristics of the development organization.

Risks/disadvantages of agile processes

- Difficult to scale agile methods to large systems
  - Agile methods better suited to small teams
- Heavy reliance on teamwork
  - Not all people are able to work well in teams
- Reliance on frequent access to customer
  - May be too expensive to have customer onsite (travel)
  - Large project may require too many customer representatives
- Cultural clash
  - Many XP practices clash with formal processes and management techniques.
- Not well-suited for security- or safety-critical systems
  - These depend on thorough analysis and documentation

Advantages of agile processes

- Efficient handling of changes to requirements
- Low process complexity
  - (relatively) easy to implement
- Low cost and overhead
  - Most activities directly produce quality software
- Fast results (rapid development)
  - Short iterations, core system produced up front
  - Produce final results faster
- Usable systems
  - Final system is more likely to be **Acceptable**, due to customer involvement and quick response to changes.