

Software Methodologies

Lecture #18
Software Engineering and
Project Management

Instructed by Steven Choy on Mar 12, 2007



1

Recap: Project Manager's Decision

- Consider you are the project manager and start a software project, what are the key decisions?
 - Project Goal & Scope
 - Schedule
 - Budget
 - Team Members
 - Project Organization
 - Software Tools
 - Software Life Cycle Model/Methodology



2

What's a methodology?

- A software engineering methodology is **a collection of methods, best practices, guidelines, templates and tools** for developing and managing a software system to achieve a specific goal in a given environment
- A methodology specifies:
 - When methods or tools should be used
 - What to do to achieve a specific task (e.g. how to perform unit testing) and when unexpected events occur
- So, how does it differ from the term "process"?
 - Methodology → more focus on how
 - Process → more focus on what
- Examples of software methodology:
 - Royce's Methodology
 - RUP
 - Extreme Programming
 - SSADM
 - Prince2



3

Methodology Structure

- Guidelines
 - Contain advices and recommendation to guide through a successful development
- Techniques
 - Detailed process descriptions that support activities throughout the entire software development life cycle
- Tools
 - Project management tools are integrated with the methodology
 - Main development tools
- Templates
 - Reusable documents and checklists



4

Royce's Methodology



- Based on Rational Unified Process
- Inherit all features of RUP
- Focus:
 - Planning
 - Use Constructive Cost Model for estimation
- Not a popular methodology as RUP

5

SSADM



- Short for “Structured Systems Analysis and Design Method”
- A methodology used in the analysis and design stages of system development
- Also, based on Waterfall Model (so, inherit its deficiencies)
- Commonly used in government software projects in UK and HK

6

How is SSADM structured?



- SSADM consists of five main modules:
 - Feasibility Study
 - Conduct a high-level analysis of a business idea to determine whether the system can support the business requirement in a cost-effective way
 - Requirement Analysis
 - Investigate the current environment in terms of processes and data structure
 - Produce business system options and choose from them
 - Requirement Specification
 - Identify the detailed functional & non-functional requirement
 - Further refine the selection business system
 - Logical System Specification
 - Develop technical system options in terms of software/hardware platform
 - Design program logic
 - Physical Design
 - Produce database design and a set of program specifications

7

PRINCE2



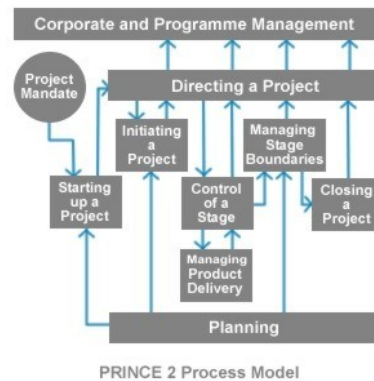
- Stands for “Project in Controlled Environments”
- First developed by the Central Computer & Telecommunication Agency of UK as a UK Government standard for IT project management
- A **project management methodology** providing organizations with a standard approach to managing and controlling projects
- Complement with SSADM

8

How does PRINCE2 work?



- PRINCE2 outlines 8 processes that are required to carry out a successful project
- Each process is defined clearly with objectives, activities and deliverables



PRINCE 2 Process Model

9

Considering Methodology...



- Any proven methodology are available to meet the organization goal?
- What's the benefit of a methodology to the organization?
- Does the selected methodology apply to all projects or some?
- How do you select and implement the methodology?
- How do you define metric to measure the productivity of applying the methodology?
- Will all project members (including client and user) comfortable with the methodology?
- How about training?

10

Agile Software Methodology



11

Manifesto for Agile Software Development



We are uncovering better ways of developing software by doing it and helping others do it.

Through this work 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.

12

Principles behind Agile



- According to AgileManifesto.org,
 - Satisfying our customer is the highest priority
 - Welcome changing requirements
 - Deliver working software frequently
 - Work with business people throughout the project
 - Build project around motivated people
 - Favor face-to-face communication
 - Continuous attention to technical excellence and good design
 - Simplicity

13

Agile Methodology



- Traditional methodologies are heavy and process-oriented
- Agile methodologies evolved in late 90's
- Heavy-weight vs Light-weight
 - A heavy-weight methodology has many rules, practices and documents
 - A light-weight methodology has only a few rules, practices to follow. And it does not stress on documentation
- Core values of Agile development:
 - **Individuals and interactions** over processes and tools
 - **Working software** over comprehensive documentation
 - **Customer collaboration** over contract negotiation
 - **Responding to change** over following a plan
- Best known agile methodology: **XP (Extreme Programming) and Scrum**

14

Extreme Programming (XP)



- Extreme Programming is a discipline of software development based on values of simplicity, communication, feedback, and courage. It works by bringing the whole team together in the presence of simple practices, with enough feedback to enable the team to see where they are and to tune the practices to their unique situation
- Target: Small to mid-sized teams (typically from 2 to 12 members) building software under vague or rapidly changing requirements

15

Core Values & Principles of XP



- XP has four core values:
 - **Communication**
 - Continual communication with users and within team
 - **Simplicity**
 - Choose the simplest design to just make thing work
 - **Feedback**
 - Feedback early from team, customers and end-users lets you easy to steer your direction if it goes wrong
 - Frequent feedback keeps your project on track
 - **Courage**
 - The courage to deal with problems proactively

16

Core Values & Principles of XP...



- The core values gives rise to five principles:
 - Provide rapid feedback
 - Assume simplicity
 - Make incremental changes
 - Embrace changes
 - Do quality work

17

XP Practices



- Planning game
 - Determine the scope, priorities and date of next iteration release
 - Customer writes user stories, a lightweight form of use case
 - Developer provides technical estimates
- Small releases
 - Base on the idea of iterative development
 - Put a simple system into production as quick as possible
 - Release a new version in a short cycle
 - Keeping a short release cycle can let customer feedback early and frequently

18

XP Practices...



- Metaphor
 - Enhance communication within team and between customers
 - Speak in customers' language, not techno-speak
- Simple Design
 - Focus on the current problem and design simple
 - Do not try to build a complex design to solve future problems
- Testing
 - Test-driven development
 - Automate the test (e.g. using JUnit)
 - Automated testing forms the basis for refactoring

19

XP Practices...



- Refactoring
 - Changing the code to improve readability but without modifying the functionality of the system
- Pair Programming
 - Developers work in pair to do development, including design decision, coding, testing
 - Facilitates communication, training of new developers, code review
 - Read how pair programming work:
 - <http://www.objectmentor.com/resources/articles/xpepisode.htm>
- Collective ownership
 - Anyone in the team can modify any source code of the system
 - No one owns a particular part of the system

20

XP Practices...



- Continuous integration
 - Integrate and build the system as soon as you finish up a change in a component, module or class
- 40-hours week
 - Work no more than 40 hours per week whenever possible
- Onsite customer
 - Have a customer onsite with the team to answer questions
 - Better & rapid response to clarify requirement

21

XP Practices...



- Coding standard
 - Produce code in consistent style (independent of author) that improves readability
 - Standard should be documented and formalized with team or even organization
 - Working in pair can help developers conforming to the coding standard

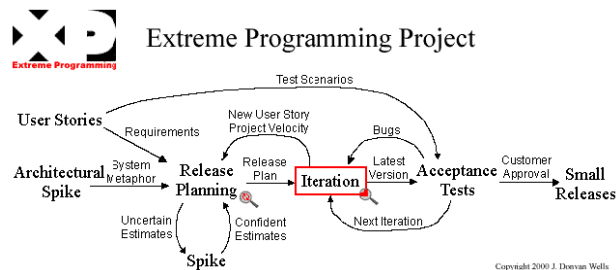
22

XP Roadmap



<http://www.extremeprogramming.org/>

<http://www.extremeprogramming.org/map/project.html>



23

Source: <http://www.extremeprogramming.org/map/project.html>

Readings...



Five Lessons You Should Learn from Extreme Programming

1. Code for Maintainability
2. Know Your Status
3. Communicate Early and Often
4. Do Things That Matter
5. Fix Your Most Important Problem First

<http://www.onlamp.com/pub/a/onlamp/2003/07/31/extremeprogramming.html>

Comics from VisualPatterns.com: An 8-Week Project Using Extreme Programming and Agile Model Driven Development

<http://www.visualpatterns.com/comics.jsp>

24

Supplementary Lecture

Refactoring



25

Refactoring



- Refactoring is a transformation of the source code that improves its readability or modifiability without changing the behaviour of the system
- Refactoring is a controlled technique for improving the design of an existing code base. [Fowler]

26

Refactoring Example

- Take a look, how can it be improved?

```
public class Product {  
  
    public void printProductDetails() {  
        System.out.println("-----");  
        System.out.println("Product");  
        System.out.println("-----");  
  
        System.out.println("Name: " + this.name);  
        System.out.println("Product ID: " + this.id);  
        System.out.println("Price: " + this.price);  
        System.out.println("Description: " + this.description);  
        System.out.println("Manufacturer: " + this.manufacturer);  
    }  
  
    public void printProductBriefInfo() {  
        System.out.println("-----");  
        System.out.println("Product");  
        System.out.println("-----");  
  
        System.out.println("Name: " + this.name);  
        System.out.println("Product ID: " + this.id);  
        System.out.println("Price: " + this.price);  
    }  
}
```

27

After Refactoring

- Extract common code and put it into a method

```
public class Product {  
    public void printProductDetails() {  
        printBanner();  
  
        System.out.println("Name: " + this.name);  
        System.out.println("Product ID: " + this.id);  
        System.out.println("Price: " + this.price);  
        System.out.println("Description: " + this.description);  
        System.out.println("Manufacturer: " + this.manufacturer);  
    }  
  
    public void printProductBriefInfo() {  
        printBanner();  
  
        System.out.println("Name: " + this.name);  
        System.out.println("Product ID: " + this.id);  
        System.out.println("Price: " + this.price);  
    }  
  
    public void printBanner() {  
        System.out.println("-----");  
        System.out.println("Product");  
        System.out.println("-----");  
    }  
}
```

28



Refactoring Catalog



- Common refactoring techniques are grouped into a refactoring catalog
- Like the technique we use in the previous example is known as “**Extract Method**”
- Check out the catalog of refactoring at:
<http://www.refactoring.com/catalog/index.html>

29

Refactoring Catalog Sample #1



- Consolidate Conditional Expression
 - You have a sequence of conditional test with the same result → Combine them into a single conditional expression and extract it

```
double disabilityAmount() {
    if (_seniority < 2) return 0;
    if (_monthsDisabled > 12) return 0;
    if (_isPartTime) return 0;
    // compute the disability amount
}
```



```
double disabilityAmount() {
    if (isNotEligibleForDisability())
        return 0;
    // compute the disability amount
}
```

30

Refactoring Catalog Sample #2



- Replace error code with Exception
 - A method returns a special code to indicate an error → Throw an exception instead

```
int withdraw(int amount) {
    if (amount > _balance)
        return -1;
    else {
        _balance -= amount; return 0;
    }
}
```



```
void withdraw(int amount)
    throws BalanceException {
    if (amount > _balance)
        throw new BalanceException();
    _balance -= amount;
}
```

31

Refactoring Catalog Sample #3



- Hide method
 - A method is not used by any other class → Make the method **private**

32



Refactoring Catalog Sample #4

- Introduce Explaining Variable

- You have a complicated expression → Put the result of the expression in a temporary variable with a name that explains the purpose

```
if ( (platform.toUpperCase().indexOf("MAC") > -1)
    && (browser.toUpperCase().indexOf("IE") > -1)
    && wasInitialized() && resize > 0 ) {
    // do something
}
```



```
final boolean isMacOs = platform.toUpperCase().indexOf("MAC") > -1;
final boolean isIEBrowser = browser.toUpperCase().indexOf("IE") > -1;
final boolean wasResized = resize > 0;

if (isMacOs && isIEBrowser && wasInitialized() && wasResized) {
    // do something
}
```

33



Acknowledgement

The slides were originally authored by **Simon Ng**.

34