SOFTWARE LIFE-CYCLE PROCESSES
From Waterfall to Extreme Programming

Lesson Objectives

• Understand major activities of software projects
• Understand the place of these in different life-cycle models
• Understand the pros and cons of different life-cycle models
• Know enough about the incremental delivery and extreme programming to use relevant parts
Software life-cycle phases

- Requirements capture and analysis
- Specification
- Design
- Implementation
- Testing
- Installation
- Maintenance
- Validation and verification

Requirements Capture and Analysis

- Identify and agree general functional and non-functional requirements
- Prioritise requirements where possible
- Determine risk factors, cost analysis, development schedule
- Identify how customer might test completed system
- Identify likely changes and how to accommodate them
Specification

- Says *what* the system does not *how* it does it
- Formal version of requirements specification, written by developer
- Developer identifies errors, omissions, or impracticalities in the requirements
- Crystalises precise functional and performance requirements
- Basis for formal contract, hence should contain sufficient explanation for customer to understand
- Should include detailed plan for acceptance test

The problem with specification

- What the salesman promised
- Architecture Group’s modified design
- General release version
- What product marketing specified
- Design group’s initial design
- Pre-release version
- Solid mahogany
- What the customer actually wanted
Design

State Diagrams

Implementation Diagrams

We will focus on object-oriented design using UML

Implementation

• Process of expressing the detailed design in a programming language so that it will run on the target computer
Validation and Verification

Verification: Are we building the product right?
Validation: Are we building the right product?

Testing

- Unit tests
- Integration tests
- System and acceptance test
Installation

- Create working version on target machine
- Make available all support files and manuals
- Coordinate with hardware
- Rerun acceptance test
- Training and on-line support
- Evaluation of project

Maintenance and Support

- Corrective
- Adaptive
- Perfective
- Preventive
Software Documentation

Life-Cycle Types

- Waterfall
- Rapid prototyping
- Evolutionary/incremental development
- Formal transformations model
- Spiral model
- Unified process (UP)
- Extreme programming (XP)
Waterfall Model

Requirements Capture
Specification
Design
Implementation
Testing
Operation

How it often goes

Requirements Analysis

Vaporware
Formal Transformation Model

- Requirements Capture
- Formal specification
  - Refinement 1
  - Refinement 2
  - Refinement n
    - (source code)
  - Executable code
- Proof indicates formal process (in the mathematical sense)

Rapid Protoyping

- Requirements Capture
- Specification
  - Design
    - Implementation
    - Testing
  - Prototype Implementation
  - Testing
- Operation
Evolutionary/iterative Development

“You should use iterative development only on projects that you want to succeed”

Martin Fowler, 2000

Spiral Model (Boehm) Deals with Iteration

- Identify risks
- Assign priorities to risks
- Develop a series of prototypes for the identified risks starting with the highest risk.
- Use a waterfall model for each prototype development (“cycle”)
- If a risk has successfully been resolved, evaluate the results of the “cycle” and plan the next round
- If a certain risk cannot be resolved, terminate the project immediately
Unified Process (UP)

- Current state-of-the-art methodology
- Initially developed by designers of UML
- Structures project as a number of phases
- Each phase contains several iterations
- Different workflows (activities) are performed in each iteration
Unified Process Outline

The balance between workflows is different in different phases

UML and the Unified Process

- UML is a language
- It defines diagrams which represent aspects of software systems
- UML's diagrams can be used in conjunction with many different processes (or even in the absence of a formal process)
- Because of their history, there is a close fit between UML and the UP
Extreme Programming Principles

- Plan to release in small increments
- Test first
- Keep it simple
- Own it collectively
- Code to standards
- Integrate continuously
- Refactor
- Program in pairs

Extreme Programming: Planning and Management

- Initial brief ‘prototype’ phase
- Quickly determine scope of next release
- Put simple system into production quickly, then release new versions on a short cycle.
- Keep meetings short but frequent
- Involve the customer throughout
- Don’t burn out
- Embrace change – it will happen anyway
Extreme Programming: Testing

- Write unit tests for each method even before you start coding
- Tests provide a definition and documentation of the required behaviour
- Integrate and build the system every time a task is completed

Extreme Programming: Keeping it simple

- System should be designed as simply as possible at any given moment
- Extra complexity is removed as soon as it is discovered
- Look for well known design patterns
- Refactor
Extreme Programming: Coding

- All production code written by two programmers at one machine
- Anyone can change any code anywhere in the system at any time
- All code written to agreed standard that emphasizes communication throughout

...Real Software Life-Cycle?

<table>
<thead>
<tr>
<th>Requirements capture</th>
<th>Lone genius has bright idea</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coding</td>
<td>Friends drop by with more bright ideas</td>
</tr>
<tr>
<td>Test and trial</td>
<td>Someone in accounts wants to play with it</td>
</tr>
<tr>
<td>User documentation</td>
<td>Someone in accounts can't remember how to drive it</td>
</tr>
<tr>
<td>Architectural design</td>
<td>You must be joking</td>
</tr>
<tr>
<td>Detailed design</td>
<td>&quot; &quot;</td>
</tr>
<tr>
<td>Functional specification</td>
<td>'Manager wants to know what you've been doing for the past year'</td>
</tr>
<tr>
<td>Requirements definition</td>
<td>'Manager wants to know what xxxxx use it is'</td>
</tr>
<tr>
<td>Maintenance</td>
<td>Impossible</td>
</tr>
</tbody>
</table>
..Real Software Life-Cycle?

- Enthusiasm
- Disillusionment
- Panic
- Collapse
- Search for the guilty
- Punishment of the innocent

Lesson Summary

- Some kind of defined life-cycle needed
- Classic waterfall is dead
- Different life-cycle models contain common key activities.
- Prototyping and evolutionary development approaches allow more user feedback
- Extreme programming is suitable for group project