Note 10: Software Process

Yan Shi
Lecture Notes for SE 3330
UW-Platteville

Based on Pressman Chapter 2 & 3
Software Process

- Definition: a framework for the tasks that are required to build high-quality software.
Generic Process Framework

- **Communication**
  - communication/collaboration with customers & other stakeholders
  - goal is to figure out requirements

- **Planning**
  - Describing tasks, risks, resources, work products and a work schedule

- **Modeling**
  - creation of models to clarify requirements and outline the design

- **Construction**
  - Coding and testing

- **Deployment**
  - delivering the product including customer evaluation
Umbrella Activities

- Activities applicable across all framework activities:
  - project tracking and control
  - risk management
  - quality assurance
  - formal technical reviews
  - measurement: process, project, product
  - software configuration management
  - reusability management
  - construction methods

- Although sharing same elements, process models do differ fundamentally!
Process Assessment

- **CMMI (Capability Maturity Model Integration)**
  - 5 maturity levels: performed >> managed >> defined >> quantitatively managed >> optimized
  - CMMI appraisal published results
Prescriptive process models define a distinct set of activities, actions, tasks, milestones and work products that are required to engineer high quality software.

- They are not perfect.
- Need to be customized based on different projects.
The Waterfall Model

Communication

Planning

Modeling

Construction

Deployment

“classic” life cycle

Sequential approach
Waterfall Model Advantages

- **Simple** and easy to follow.
- Documentations are produced at each phase.
- Similar to other engineering process models.
  - Often used when developing software which is part of a larger system engineering project.
Waterfall Model Problems

- The main drawback of the waterfall model is the difficulty of accommodating change after the process is underway.  
  *Why?*

- In the waterfall model, one phase has to be complete before moving on to the next phase.

- Inflexible partitioning of the project into distinct stages makes it difficult to respond to changing customer requirements.

- Therefore, this model is only appropriate when the requirements are well-understood and changes will be fairly limited during the design process.
Incremental Model

apply waterfall model in iterative fashion
Incremental Model

- General pattern:
  - identify features; analyze, design, implement those features; repeat with new set of features

- Typically start with core functionality
- Often do analysis, design, implementation in parallel (second team can start based on results of first, etc.)
- Each iteration must provide useful features to customer
Incremental Model Advantages

- Give customer chances to provide feedback on product before final delivery
- Help convince customer progress is being made
- Great when full team might not be available early
- Handle scheduling problems such as certain types of hardware not being available until very late in the project
The RAD process model

- **communication:**
  - establish business problem, requirements

- **planning:**
  - must allow for multiple teams working in parallel!

- **modeling:**
  - business modeling, data modeling, process modeling

- **construction:**
  - make **heavy reuse** of existing components
  - if there are lots of reuse, testing assumes existing components having been tested already
  - data/process modeling, construction should be doable in the **60-90 day period**

- Deployment (cutover)
The RAD Model

- The Rapid Application Development model:
  - Is an incremental model
  - Emphasizes a short development cycle
  - High-speed adaption of waterfall model
  - Often uses pre-existing components and automatic code generation

- Use RAD model when:
  - Well-understood requirements, project scope constrained
  - Goal: fully functional system within 60 to 90 days
RAD Advantages

- Quick response to new business needs
- High level of satisfaction for developers: they get things to work quickly
- Can be applied to larger projects by breaking the project into smaller pieces each of which is done using the RAD model by different teams
  - it must be possible to identify sub modules
  - there must be sufficient human resources
RAD Problems

- Human resources have to be in place at *start* of project
- Both developers and customers must be committed to rapid-fire activities
  - need quick turnaround on decisions
- Not appropriate when technical risks are high
  - not when new technology plays a major role
  - not when must interface extensively with existing software
  - that is, problem must have tight scope
- If tuning needs to be done between modules to obtain appropriate performance, not clear when this would be done
Evolutionary (Iterative) Models

- For products that evolve all the time
- Iterative models

- Two major Evolutionary models:
  - Prototyping
  - Spiral
Throwaway Prototyping

- Often, it is necessary to experiment with a system to determine what exactly is needed or what is the best way to implement it.

- Prototypes are quick solutions that are intended to be thrown away when done.

- It is important not to let a prototype be released into production.
  - They were not designed or implemented with the same care that we would exercise for production code.
The Prototyping Model

Communication

Deployment: Delivery and Feedback

Quick Plan

Construction of Prototype

Modeling: Quick Design

Even after iterations, the final prototype is still the “first system” to be thrown away!
When to Use Prototyping

- Prototyping is useful when:
  - Customers don’t know exactly what they want
  - We want to analyze the usability of an HCI
  - We are not sure if a particular approach to solving a problem will work
  - We have multiple approaches to solving a problem and we want to experiment to see which is better.
Possible Problems with Prototyping

- Manager and/or customer thinks development is almost done and expects developer to produce full system by making just a few tweaks
  - *Must have* a "contract" with management acknowledging that system *must* be rewritten

- Developer reuses stuff that shouldn't be
  - developer makes expedient choices on tools, algorithms;
  - developer becomes comfortable with those choices, does not re-examine them
  - must record such decisions so know to re-examine them!

- Turning prototype into product erodes quality and must be resisted
Variations
Spiral Model

- It is a risk-driven process model.
- Each spiral is divided into task regions.

- Key element of each spiral: evaluate what to do for the next spiral
  - establish scope for work done in next spiral, or
  - decide to revisit previous work, or
  - cancel project
- Canceling early is better than canceling late!
Spiral Model Advantages

- Can be applied throughout the life of the software:
  - first circuit: “concept development project”
  - later circuits: “product enhancement project”

- Realistic model for large-scale problems
  - use prototyping as a risk reduction mechanism

- Build-in plan for revisiting work, etc.
  - reduce risks
Spiral Model Problems

- Doesn't work well for fixed-budget problems!
  - each spiral is supposed to work off the one before it; can't know the cost of new spiral until old completed

- Requires risk assessment expertise to identify what to work on first

- Must convince customers evolutionary approach is manageable
  - less of a track record
  - note this is an issue for all evolutionary models!
Unified Process

- History:

  - Unifies Process is a framework for object-oriented software engineering using UML.
  - The process is *use-case driven*.
  - The process is *incremental and iterative*. 

- Unified Process (UP)

- Unified Modeling Language (UML)

- Object Oriented programming methods (OO)
Four Phases of Unified Process

- Inception
- Elaboration
- Construction
- Transition

Communication
Planning
Modeling
Construction
Deployment
Disciplines Involved in Each Phase

Iterative Development
Business value is delivered incrementally in time-boxed cross-discipline iterations.

<table>
<thead>
<tr>
<th>Inception</th>
<th>Elaboration</th>
<th>Construction</th>
<th>Transition</th>
</tr>
</thead>
<tbody>
<tr>
<td>I1</td>
<td>E1</td>
<td>C1</td>
<td>T1</td>
</tr>
<tr>
<td></td>
<td>E2</td>
<td>C2</td>
<td>T2</td>
</tr>
<tr>
<td>Business Modeling</td>
<td></td>
<td>C3</td>
<td></td>
</tr>
<tr>
<td>Requirements</td>
<td></td>
<td>C4</td>
<td></td>
</tr>
<tr>
<td>Analysis &amp; Design</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implementation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deployment</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Time
Inception

- Identify preliminary use cases
- Develop preliminary architecture
- Outline development

- Primary goal: establish goals for project and each iteration (with inputs from all stakeholders)
- Key issue: address business and requirements risks
Disciplines Involved in Each Phase

Iterative Development
Business value is delivered incrementally in time-boxed cross-discipline iterations.

<table>
<thead>
<tr>
<th>Inception</th>
<th>Elaboration</th>
<th>Construction</th>
<th>Transition</th>
</tr>
</thead>
<tbody>
<tr>
<td>I1</td>
<td>E1</td>
<td>C1</td>
<td>T1</td>
</tr>
<tr>
<td>E2</td>
<td></td>
<td>C2</td>
<td>T2</td>
</tr>
<tr>
<td>Business Modeling</td>
<td></td>
<td>C3</td>
<td></td>
</tr>
<tr>
<td>Requirements</td>
<td></td>
<td>C4</td>
<td></td>
</tr>
<tr>
<td>Analysis &amp; Design</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implementation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deployment</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Elaboration

- **Use cases**: 80% level of final detail
- **Non-functional requirements** completed
- **Design and architecture**: high-level sequence, class, state diagrams
- **Establish baseline architecture**: package diagram
- **Executable architecture/prototypes**
- **Components**: identify existing components to incorporate into design
- **Planning**
- **Risks** reviewed, updated
- **Testing** - develop test cases
- **Evaluate** - continue or abort?
Disciplines Involved in Each Phase

Iterative Development
Business value is delivered incrementally in time-boxed cross-discipline iterations.

<table>
<thead>
<tr>
<th>Inception</th>
<th>Elaboration</th>
<th>Construction</th>
<th>Transition</th>
</tr>
</thead>
<tbody>
<tr>
<td>I1</td>
<td>E1</td>
<td>C1, C2, C3, C4</td>
<td>T1, T2</td>
</tr>
</tbody>
</table>

- **Business Modeling**
- **Requirements**
- **Analysis & Design**
- **Implementation**
- **Test**
- **Deployment**

**Time**
Construction

- Make use cases operational for end users
  - i.e., implementation work
- Clarify remaining requirements/refine use cases
- **Key:** optimize costs, schedule, and quality
- **System features are implemented in a series of short, time boxed iterations:**
  - Each iteration results in an executable release.
  - Each iteration is either an increment or an improvement.
- **Documentation:** user's manual, training classes, sales literature, etc.
- **Alpha level testing**
Disciplines Involved in Each Phase

Iterative Development
Business value is delivered incrementally in time-boxed cross-discipline iterations.

<table>
<thead>
<tr>
<th>Disciplines</th>
<th>Inception</th>
<th>Elaboration</th>
<th>Construction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business Modeling</td>
<td>I1</td>
<td>E1</td>
<td>E2</td>
</tr>
<tr>
<td>Requirements</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Analysis &amp; Design</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implementation</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deployment</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Time
Transition

- Beta/acceptance testing by end users, deployment, user training
- **End result**: available for customers/end users
- May make minor adjustments based on user feedback
- **All major defects found before** entering this phase!
Relative Sizes of the Four Phases

- Inception
- Elaboration
- Construction
- Transition
Specialized Process Models

- Tend to be applied when a narrowly defined software engineering approach is chosen.

- Component-Based Development (covered in SE 2730)

- Aspect-Oriented Development

- Formal Methods
Aspect-Oriented Software Development (AOSD) is a new approach to software design that addresses modularity problems that are not handled well by other approaches.

Typical enterprise and internet applications today have to address "concerns" like security, transactional behavior, logging, etc..

**Concern**: a particular set of information that has an effect on the code.
- Modularity is achieved by addressing separate concerns by separate modules
Crosscutting Concerns

- Crosscutting concerns
  - not the primary job of the worker classes
  - independent of the worker’s primary job
  - cross-cut the system: affect other concerns.

- Example: application for handling medical records.
  - core concern: bookkeeping and indexing of records
  - cross-cutting concerns: logging change history, security

- Problem:
  - often cannot be cleanly decomposed.
  - violation of DIY (Don’t Repeat Yourself) principle.
  - can result in either scattering, tangling, or both.
    - Scattering: code duplication
    - Tangling: significant dependencies between systems
Crosscutting Concern Example

Aspect-Oriented Development

- Aspect: a module that encapsulates a concern.
- *aspect orientation = quantification + obliviousness*
- **Quantification**: the ability of aspects to affect multiple points in the program.
- **Obliviousness**: a program has no knowledge of *which* aspects modify it *where* and *when*.
- Aspect-Oriented Programming: break down program logic into distinct aspects

*Still a relatively new topic!*
Example: Using Aspect-Oriented Design

Aspect-Oriented Programming

- Example programming languages that have implemented AOP, within the language or as an external library:
  - .NET Framework (C#, VB.NET)
  - C/C++
  - COBOL
  - Delphi
  - Java (AspectJ)
  - PHP
  - Perl
  - etc.
Formal Method

- **Mathematically-based techniques** for the **specification**, **development** and **verification** of software systems.

- **Specification:**
  - BNF (Backus Normal Form)
  - Petri nets
  - Z-notation

- **Development:**
  - Assertions (based on pre and post conditions of the specification)

- **Verification:**
  - Model Checking

- **Cleanroom Software Engineering Process:**
  - Original developed by IBM.
  - Focus: defect prevention, rather than defect removal.
  - **Z-notation**: a specification language (pronounced “zed”) to decompose a system into schemas
Summary

- Five generic process framework activities:
  - Communication, Planning, Modeling, Construction, Deployment

- Prescriptive Process Models
  - Waterfall model (classic)
  - Incremental Model: incremental and RAD
  - Evolutionary (Iterative) Model: Prototyping and Spiral
  - Unified Process
  - Special Models:
    - Component-Based
    - Aspect-Oriented
    - Formal Methods