Process Models

Why Process Models Important?

- Improving time to market and reducing production cost
- Process influences the quality of products
- By controlling processes, achieved better control of the required qualities of products.

Source: J. Hong

Basic Concepts - Architecture

Basic Concepts - Processes

Source: J. Hong
A task set defines the actual work to be done to accomplish the objectives of a software engineering action.

- A list of the tasks to be accomplished
- A list of the work products to be produced
- A list of the quality assurance filters to be applied

A process pattern
- describes a process-related problem that is encountered during software engineering work,
- identifies the environment in which the problem has been encountered, and
- suggests one or more proven solutions to the problem.

Stated in more general terms, a process pattern provides you with a template - a consistent method for describing problem solutions within the context of the software process.
Process Pattern Types

- Phase patterns
  - define the sequence of framework activities in a process, even when the overall flow of activities is iterative in nature.
- Spiral Model or Prototyping pattern
- Stage patterns
  - defines a problem associated with a framework activity
- Task patterns
  - defines a problem associated with an action or work task and relevant to successful software engineering practice

Key to success: Process + Practice!

Major Output Documents

<table>
<thead>
<tr>
<th>Phase</th>
<th>Documents</th>
<th>Components</th>
<th>Remarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Project planning</td>
<td>Project Management Plan</td>
<td>- Management Issues (cost, schedule, resources, etc) - Quality Management Plan - Configuration management Plan</td>
<td></td>
</tr>
<tr>
<td>Requirements Gathering</td>
<td>RDD (Requirements Description Doc.)</td>
<td>- System description - Functional requirements - Non-functional Requirements</td>
<td></td>
</tr>
<tr>
<td>Requirements Analysis</td>
<td>SRS (Software Req. Specification)</td>
<td>- Functional Analysis Model - Data Model</td>
<td></td>
</tr>
<tr>
<td>Design</td>
<td>SDD (Software Design Description)</td>
<td>- Preliminary Design - Detailed Design (data, interface,..) - Deployment</td>
<td>Separated in some case</td>
</tr>
<tr>
<td>Implementation</td>
<td>Source Code List</td>
<td>- Code</td>
<td></td>
</tr>
</tbody>
</table>

ISO/IEC 12207

- Purpose
  - To establish a common framework for the life cycle of software:
    - To acquire, supply, develop, operate & maintain software
    - To manage, control, and improve the framework
- History
  - Proposed in June 1988; Published 1 August 1995
  - Over 6 years and 17000 person-hours expended
- Participants
  - Australia, Canada, Denmark, Finland, France, Germany, Ireland, Italy, Japan, Korea, Netherlands, Spain, Sweden, UK, USA

Source: J. Hong
Standards for Software Process

Prescriptive Models
- Prescriptive process models advocate an **orderly** approach to software engineering
  - Provides a road map for software teams
- Example prescriptive process model
  - Waterfall model
  - Incremental process model
  - Evolutionary process model
  - Spiral model
  - Concurrent model

Build-and-Fix Model
- Mainly a single-person task
- Weak or no specification and testing
- Unsatisfactory for products of reasonable size, but served many early projects
- Not suitable for today's environments where
  - Developed for people with no computer background
  - More stringent reliability requirement
  - Group activity
The Waterfall Model

- Linear flow, document-driven
- Useful if requirements are well defined (or fixed) and stable
- A variation: V-model
  - Maps relationship between actions of framework activities and QA actions

Problems of Waterfall Model

- Linear flow, in reality?
- Can elicit all requirements at the beginning?
- Patient customers?
- Leads to blocking states due to task dependencies

The V-Model

The Incremental Model

- Each linear flow produces deliverable increments
  - First, core product, then supplementary features
  - Useful when staffing is not enough
Evolutionary Models: Prototyping

- Iterative
  - Focus on aspects visible to end users
  - Consider software evolution and requirement changes
    - When requirements are fuzzy
  - Can be implemented within the context of any other process model
- After requirement elicitation,
  - Throwaway, or
  - Evolve into actual system
- Customers and developers must
  - Resist temptation to expend prototype

Evolutionary Models: The Spiral

- Risk-driven, Evolutionary
  - Couples the nature of prototyping and waterfall model
  - Cyclic (incrementally growing)
  - Anchor point milestones – need review by stakeholders
  - Realistic for large-scale development
- Adaptable for the entire life cycle
  - Early iterations for conceptualization
  - Later, for maintenance
- Success relies on risk assessment cost and expertise

Evolutionary Models: The Spiral

- To represent concurrent elements of a process model
  - State transition diagrams can exist for process elements in any granularity
  - Trigger transitions between diagrams
    - example: at requirement changes, state changes from "under development" to "Awaiting changes"
- Captures a snapshot of project execution status
  - Process network, instead of sequence

One element of the concurrent process model
The Unified Process (UP)

- History
  - The beginning (1990s):
    - Object-Oriented software development
    - UML (Unified Modeling Language) by James Rumbaugh, Grady Booch, and Ivar Jacobson -- de facto industry standard for OOSD
  - A framework for OOSD using UML
    - Iterative, incremental process flow
    - Use-case, requirements driven
    - Architecture centric

- UP Phases
  - Inception
    - Requirement identification
    - Preliminary use cases – features, class functions
    - Architecture: minimal
    - Project planning – resources, risks
  - Elaboration
    - Refine use cases
    - Expand architectural representation and show viability
      - Use case model, requirements model, design model, implementation model, and deployment model.
      - May create a prototype system
  - Construction
    - Develop or acquire components
    - Make use cases operational
    - Complete requirements and design models
    - Conduct unit and integration testing
    - Derive acceptance tests based on use cases
  - Transition
    - Beta testing and feedback reports
    - Support documents
  - Production
    - Deploy and monitor
### UP Phases

<table>
<thead>
<tr>
<th>Phases</th>
<th>Inception</th>
<th>Elaboration</th>
<th>Construction</th>
<th>Transition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activities</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Requirements</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Implement</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Integrate</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Multiple iterations

### UP Work Products

**Inception phase**
- Vision document
- Initial use case model
- Initial project glossary
- Initial business case
- Initial risk assessment
- Project plan
- Phases and iterations
- Business model, if necessary
- One or more prototypes

**Elaboration phase**
- Use case model
- Supplementary requirements including non-functional
- Analysis model
- Software architecture description
- Evaluable architectural prototype
- Preliminary design model
- Revised risk list
- Project plan including iteration plan
- Advanced workframes
- Advanced work products
- Preliminary user manual

**Construction phase**
- Design model
- Software components
- Integrated software component
- Test plan and procedure
- Test cases
- Support documentation
- User manuals
- Installable manuals
- Description of current increment

**Transition phase**
- Delivered software increment
- Beta test reports
- General user feedback

### CBSE Process

- Component-Based Software Engineering Process
- First prominent with Douglas McIlroy, 1968
  - Mass produced software components
- Modern concept of a software component by Brad Cox, 1986
  - Software ICs
- Development with CBSE process

Source: J. Hong
Personal Software Process (PSP)

- Ineffective personal process makes the operation of team or organizational process ineffective

- PSP framework activities
  - Planning
  - High-level design
  - High-level design review
  - Development
  - Postmortem

- Metrics-based, document-oriented

- But... Not popular. Why?

Team Software Process (TSP)

- To build "self-directed" project team that organizes itself to produce high-quality software

- TSP framework activities
  - Project launch
  - High-level design
  - Implementation,
  - Integration and test
  - Postmortem

- Use of forms, scripts, and standards to guide team members.

Dealing with Legacy Software

- Motivation
  - Not possible to develop new software from scratch
    - Huge investment in developing existing software
  - Legacy: asset to preserve carefully before closing down

- Reengineering
  - Process through which an existing system undergoes an alternation, to be reconstituted in a new form

Other Process Models

- and MORE ... ?!

Source: J. Hong