# Software Process Models

## Table of Contents
- Waterfall Model
- Waterfall Model Phases
- Waterfall Model Phases (cont)
- Spiral Model
- Spiral Model (cont)

## Waterfall Model

### Phases
- **Requirements**
- **System Analysis**
  - **Problem Description**
- **Specification**
  - **Problem Definition**
- **High Level Design**
  - **Logical Design**
    - **Modular Composition**
- **Low Level Design**
  - **Dependent Design**
    - **Prog. Language**
    - **Operating Sys.**
- **Physical Design**
  - **Step-Wise Refinement**
  - **Top-Down Design**
- **Coding**
- **Integration**
- **Testing**
- **Deployment**
- **Maintenance**

### Advantages / Disadvantages
- Most-widely used process model
- Controls schedules, budgets & documentation
- Tends to favor well-understood system aspects over poorly understood system components
- Does not detect development areas behind schedule early in the lifecycle stages.

### Document-driven process
- Deliverables: documents produced at the end of each phase, usually in accordance to contract deadlines
Waterfall Model: Phases

Requirements
- A statement of the functions and behavior of the system required by its users & operators
- General Requirements
  † Defines broad & detailed objectives of the system
  † e.g., reliable, correct, efficient, user-friendly, expandable
- Gives relationship of Qualitative & Quantitative System Goals

Specification
- Listing of specific, measurable behavioral system constraints that satisfy system requirements
- Clearly communicates system operations with end user(s)
  † complete, unambiguous, minimal, understandable, testable
- Cross-reference indexed to requirement items
- Defines the design validation & final system testing criteria
- Provides chief mechanism for estimating the project’s progress

Design: Representation or model of a system

Waterfall Model: Phases (cont)

Coding and Debugging (implementation)
- Translation of design into a programming language
- Indispensable Programmer Phenomena
- Program Unit Notebooks
  1. Documents programmer’s work activities
  2. Maintains current unit (module) documentation
  3. Passed from programmer to programmer during development

<table>
<thead>
<tr>
<th>Unit Name:</th>
<th>Programmer:</th>
<th>Routines Included:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SECTION</td>
<td>CONTENTS</td>
<td>DUE DATE</td>
</tr>
<tr>
<td>1.</td>
<td>RIGHTS</td>
<td></td>
</tr>
<tr>
<td>2.</td>
<td>ARCH DESIGN</td>
<td></td>
</tr>
<tr>
<td>3.</td>
<td>UFTAIN DESIGN</td>
<td></td>
</tr>
<tr>
<td>4.</td>
<td>TEST PLAN</td>
<td></td>
</tr>
<tr>
<td>5.</td>
<td>TEST RESULTS</td>
<td></td>
</tr>
<tr>
<td>6.</td>
<td>CHANGE REQUESTS</td>
<td></td>
</tr>
<tr>
<td>7.</td>
<td>SOURCE CODE</td>
<td></td>
</tr>
<tr>
<td>8.</td>
<td>NOTES</td>
<td></td>
</tr>
</tbody>
</table>

RELEASE APPROVAL: __________________________ DATE: __________

High Level Design
- Structure Charts, Data-flow Diagrams
- User-interface design

Low Level Design
- Decision Tables
- PDLs (Prog Design Langs)
- HIPO (Hierarchy Input Processing Output)

Code Level
- Programming Language

Details Internal Code Processing:
  algorithms, data structures, control flow

less Detail
more Detail
Coding and Debugging (implementation)
- Data Dictionary: records information and physical format details of all structures, variables, files . . .

  - Data Dictionary performs documentation mapping
    - Data Objects $\rightarrow$ System Structures
    - " " $\rightarrow$ Parent Objects
    - " " $\rightarrow$ Module Routines

- Data Dictionary Entry
  - Name: from the data-flow diagram or structure chart
  - Routine Usage: routines that access the object
  - Purpose: explanation
  - Derivation: where the data that the items holds comes from ex. files, user, other entries . . .
  - Subitems: Record components
  - Notes: comments

Integration and Testing
- Unit testing: individual modules (functions) are tested separate from other modules
- Integration testing: system modules are tested together

Deployment & Maintenance
- Requires previous phases to be repeated
- Makes up 70%-90% of total system cost
- Majority of maintenance time (50%) spent on system understanding $\rightarrow$ system documentation
- Maintenance Tasks
  - collection, analysis and prioritization of user trouble reports
  - new system release installations
  - documentation (user’s manuals) changes
  - configuration control issues
The Spiral Software Process Model Diagram

- Development phases reiterates through four cycles:
  † Set goals and determine constraints for the phase
  † Evaluate and resolve risks for the phase
  † Develop the prototype for the phase
  † Plan the next stage activities
- Step 2 involves a Risk Analysis that identifies:
  - less understood system areas
  - systems areas that pose the greatest jeopardy to development

Prototype Based
- Prototype: a limited, semi-functional, task restricted, partially operational system
  † Analogous to a model or mockup that allows evaluation of development alternatives before commitment
- Rapid Prototyping Systems
  † Authoring/scripting (multimedia) systems used to quickly develop multiple interfaces for user evaluation, cannot serve as a kernel for future iterative system prototype development
  † Users tend to view prototypes as final versions of the system

Mimic
- Risk analysis produces a risk-resolution strategy
  † Feasibility Study: determination of a strategy achieving set goals and requirements within stated constraints.
    ‡ Address development factors of expertise, experience, resources and motivation
  † Extension of cost/benefit analysis
    ‡ cost & benefits are estimated for best & worst case outcomes which are multiplied by their probability of occurrence giving an expected value.
    ‡ Decisions on strategies are made to minimize cost and maximize benefits
- Cycles are modified to concentrate on different areas of system development driven by the risk-resolution plan
- Spiral model tends to behave like other process models due to differing cycles