Software Life-Cycle

• Series of steps through which software product progresses
  – From requirements through retirement
• A life-cycle is selected during requirement Phase

Build-And-Fix Model

• Product is implemented without any specification or design
• May work for projects with few hundred line of code

Build-And-Fix Model

Build First version

Modify until client is satisfied

Maintenance Phase

Retirement

Waterfall Model

Requirements Phase

Verify

Specification Phase

Verify

Design Phase

Verify

Implementation phase

Test

Integration phase

Test

Maintenance phase

Retirement
Rapid Prototyping Model

- Develop product with reduced capability
- Present to client for approval
- Develop specification with better understanding
- Prototype could give designers
  - Insight to the product
  - What not to do

Incremental Model

- The product is divided into builds
- Each build contains an operational quality subsystem
- With each additional build a new subsystem is integrated with the previous build
- Other types of Incremental Models
  - Extreme Programming Model
  - Synchronize-And-Stabilize Model
Spiral Model

- Concept is to minimize risk via the use of prototypes and other means
- Well suited for:
  - Internal development
  - Large scale

Spiral Model

- Set of important requirements are selected for each prototype
- Risk analysis is performed at each step of development
  - Specification
  - Design
  - Implementation
  - Integration

Object-Oriented Life-Cycle Models

- Fountain model
- Recursive/parallel life cycle
- Unified software development process
Object-Oriented Life-Cycle Models

- All Object oriented life cycles are:
  - Iterative
  - Incorporate some form of parallelism (Overlap of Activities)
  - Support incremental development

Object-Oriented Life-Cycle Models

- Fountain model
  - Each circle represents various phases overlap
  - Overlap between activities
  - Arrows within each phase represent iteration within that phase
  - The maintenance circle is smaller
    - Symbolizes reduced maintenance effort when Object Oriented paradigm is used

Comparison of Life-Cycle Models

<table>
<thead>
<tr>
<th>Life-Cycle Model</th>
<th>Strengths</th>
<th>Weaknesses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Build-and-fix model</td>
<td>Fine for short programs that will not require any maintenance.</td>
<td>Totally unsatisfactory for nontrivial programs.</td>
</tr>
<tr>
<td>Rapid prototyping model</td>
<td>Ensures that delivered product meets client's needs.</td>
<td>May degenerate into build-and-fix.</td>
</tr>
<tr>
<td>Incremental model</td>
<td>Ensures that delivered product meets client's needs.</td>
<td>Requires open architecture.</td>
</tr>
<tr>
<td>Extreme model</td>
<td>Promises early return on investment.</td>
<td>Has not yet been widely used.</td>
</tr>
<tr>
<td>Synchronize-and-stabilize model</td>
<td>Ensures early return on investment.</td>
<td>Has not been widely used other than at Microsoft.</td>
</tr>
<tr>
<td>Spiral model</td>
<td>Incorporates features of all the other models.</td>
<td>Can be used only for large-scale, in-house products.</td>
</tr>
<tr>
<td>Object-Oriented model</td>
<td>Support iteration within phases, overlapping between phases.</td>
<td>May degenerate into CAB/FAB.</td>
</tr>
</tbody>
</table>

Figure 3.10