Chapter 7
Implementation

Implementation Approaches

- Big bang
  - Code entire system and test in an unstructured manner
- Top-down
  - Start by implementing modules on top of hierarchy and test by coding stubs
- Bottom-up
  - Start by implementing modules on bottom of hierarchy and code drivers to test
- Threads
  - Implement and test a minimal set of modules implementing a function

Implementation Approaches
Advantages and Disadvantages

- **Big bang advantages**
  - None
- **Big bang disadvantages**
  - Difficult to debug
  - Much throwaway code
  - Critical and peripheral modules not distinguished
  - User does not see product until very late in the development cycle

- **Top-down advantages**
  - Separately debugged modules
  - System test by integrating previously debugged modules
  - Stubs are easier to code than drivers
  - User interfaces are top-level modules
- **Top-down disadvantages**
  - Stubs must be written
  - Low-level, critical modules built last
  - Testing upper-level modules is difficult

- **Bottom-up advantages**
  - Separately debugged modules
  - System test by integrating previously debugged modules
  - Testing upper-level modules is easier
- **Bottom-up disadvantages**
  - Drivers must be written
  - Upper-level, critical modules are built last
  - Drivers are more difficult to write than stubs
  - User interfaces are top-level modules

Implementation Plan

- Specifies which implementation strategy will be employed
- Divides the project into phases
- Specifies a schedule for each phase including the following steps for each phase:
  - Coding
  - Unit testing
  - Integration of units
  - Integration testing
**Sample Implementation Plan**

<table>
<thead>
<tr>
<th>Week number</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Phase I coding</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Phase I unit testing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Phase I integration</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Phase I integration testing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Phase II coding</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Phase II unit testing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Phase II integration</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Phase II integration testing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Implementation in the LMS Case Study**

- Phase I
  - CheckOutResource
  - CheckInResource
- Phase II
  - ManageResource
- Phase III
  - ManagePatron
- Phase IV
  - BrowseResource
  - RequestResource
  - ReserveResource
  - GenFormLetter

(See deliverable 7.3 for a sample schedule)

**Programming Style**

- Programmers should strive for simple and readable code
- Shorter code is usually simpler code
- Simpler code typically contains fewer decisions
- Excessively nested logic should be avoided

**Comments and Internal Documentation**

- Most successful software systems exist for several years and require modification during that time
- Comments are the chief source of information about the source code, and thus are extremely important
- A header comment block summarizes the purpose of a module (class or method)
- Line comments are inserted into the source code to explain the objectives of a subset of the instructions comprising a module

**Programming Standards for the LMS Case Study**

- **General Rules:**
  - Structured programming techniques should be used in all methods
  - Comments should be used to identify the purpose of each method and to clarify obscure processing
  - Each method must be prefaced with a comment box
  - All major blocks of code should also be prefaced with comment boxes
  - Each major control structure (loop, selection) should be clarified through a line comment
  - All code should be simple and easy to read

- **Modularization Rules:**
  - Each method must perform only one task
  - Global variables should not be used. Data should be shared only through the parameter list
  - If, during implementation, a need arises for a class not specified in the design, the design must be reexamined before the new class is implemented
Programming Standards for the LMS Case Study

- Declarations:
  - All variables must be declared and explicit types used
  - Name of variables and methods should describe their content or usage. An exception can be made for naming subscripts and counters
  - Declarations of any files should precede variable declarations
  - Declarations of instance variables in a class should precede method definitions

Executable instructions:
- Align and indent parallel constructs. Indent subsequent levels at least three spaces. Each line should contain at most one instruction. Use blank lines to set off major control constructs
- Do not use goto
- Use priming read/looping read technique for all sequential files
- Open and close files explicitly
- Avoid negative conditions

Complete your own Implementation Plan:
- Gather into your development groups, and work out your plan: impPlan.doc