

UNIT 5

Implementation and Maintenance

(4 Hours) — System Analysis and Design (CACS203)

1. System Implementation

1.1 Introduction and System Implementation

Implementation is the phase in which the designed system is **built, tested, installed** and put into operation. It has six major activities: **coding, testing, installation, documentation, training and support**. The phase ends when the system is handed over to users and the project is formally reviewed.

Activity	Purpose	Deliverable
Coding	Convert design into working programs	Source code, database
Testing	Find and fix errors; verify requirements	Test plans, test results
Installation	Replace the old system with the new one	Operational system
Documentation	Explain how to use and maintain the system	System and user documents
Training and support	Prepare users and provide help	Trained users, help desk

1.2 Software Application Testing

Testing is the process of executing software to find errors and to confirm that it meets requirements. **Verification** asks “are we building the product right?” and **validation** asks “are we building the right product?”. A successful test is one that finds an error.

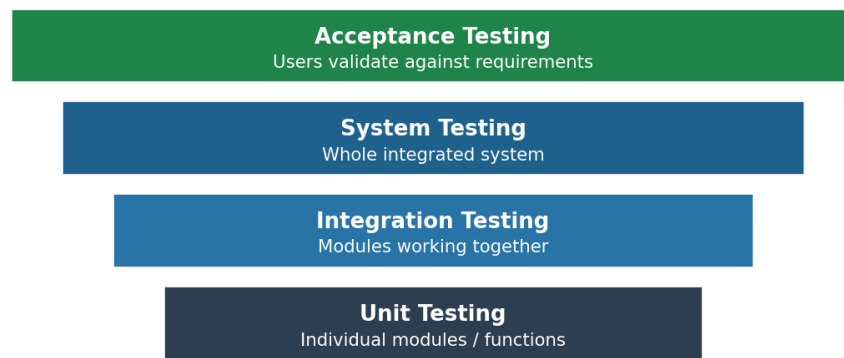


Figure 5.1 — Levels of testing (from unit testing up to acceptance testing)

Test level	Description	Performed by
Unit (module) testing	Test each program, function or class in isolation	Programmers
Integration testing	Test that combined modules work together (top-down, bottom-up, big-bang)	Programmers / testers
System testing	Test the complete, integrated system against specifications (functional, performance, security, recovery, stress)	Test team
Acceptance testing	Users verify the system meets business needs: alpha (in-house) and beta (real users)	Users / customers

Testing Approaches

- **Black-box testing** — tests inputs and outputs without seeing the code (equivalence partitioning, boundary-value analysis).
- **White-box testing** — tests internal logic (statement, branch and path coverage).
- **Static testing** — inspections, walkthroughs, desk checking without running the program.
- **Regression testing** — re-run tests after a change to confirm nothing else has broken.
- **Stub and driver** — a stub simulates a called module; a driver simulates a calling module.

Example 5.1 — Boundary-value analysis

A field accepts marks from 0 to 100. Give test values.

Solution

Valid boundaries: **0, 1, 99, 100**. Invalid values just outside the range: **-1, 101**. A typical middle value such as **50** is also tested. Errors most often occur at the boundaries.

1.3 Installation

Installation is the organizational process of changing from the old system to the new one. Four approaches exist:

Approach	Description	Risk	Cost
Direct (cut-over)	Old system stops, new starts immediately	High	Low
Parallel	Both run together until the new one is trusted	Low	High
Single-location (pilot)	New system is installed at one site first, then others	Medium	Medium
Phased	New system introduced module by module or in stages	Medium	Medium

Installation also includes **data conversion** (moving data from old files to the new database), hardware and software set-up, and site preparation.

1.4 Documenting the System

- **System documentation** — for analysts and programmers: requirements, DFD/ERD, design specifications, source code comments, test cases, data dictionary. Essential for maintenance.
- **User documentation** — for end users: user manuals, quick-reference guides, tutorials, online help, FAQs, release notes.

1.5 Training and Supporting Users

Users must be trained for the system to succeed. Training covers using the system, its business context, and problem solving. Methods: **instructor-led classes, one-to-one coaching, computer-based/e-learning, video tutorials, documentation and vendor training**. Support is provided through a **help desk**, on-line help, chat, bulletin boards, **power users** in each department and **information centres**.

1.6 Organizational Issues in Systems Implementation

Technology alone does not ensure success. Common issues include **resistance to change**, lack of management support, poor communication, insufficient user involvement, unrealistic expectations, and data-quality and security concerns. Success is measured by use, user satisfaction, meeting objectives, and cost–benefit results. Managers reduce risk by involving users early, communicating openly, providing training, using change-management methods and **post-implementation reviews**.

2. System Maintenance

2.1 Introduction and Maintaining Information Systems

Maintenance is the modification of a system after delivery to correct faults, improve performance or adapt to a changing environment. It usually lasts longest and costs the most in the life cycle — often 60–80% of total cost. Maintenance falls into four types:

Type	Purpose	Example
Corrective	Fix errors and bugs found in use	Correct wrong tax calculation
Adaptive	Change to match new environments, laws, platforms	Support a new operating system or tax law
Perfective	Add features or improve performance/usability	Add a search filter, faster report
Preventive	Restructure to prevent future problems and ease later change	Refactor code, update documentation

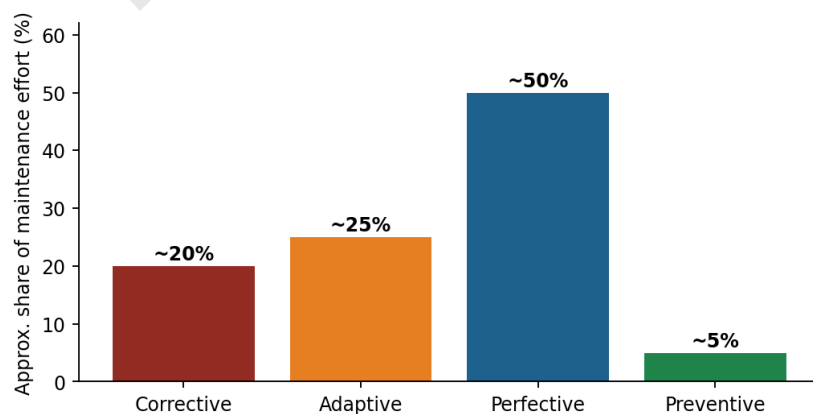


Figure 5.2 — Typical (approximate) distribution of maintenance effort

Note: The percentages are indicative; surveys show perfective and adaptive maintenance together take most of the effort, so most “maintenance” is actually enhancement, not bug fixing.

2.2 Conducting Systems Maintenance

Maintenance follows a mini-lifecycle, often managed as a formal **change-control** process:

- **1. Obtain a maintenance request** — a system change request (SCR) is logged.
- **2. Analyse and approve** — evaluate impact, priority, cost and benefit.
- **3. Design the change** — plan modifications to code, data and documents.
- **4. Implement** — code the change.
- **5. Test** — unit, integration and regression tests.
- **6. Release** — install the new version, update documentation and notify users.

Factors affecting maintenance cost: the number of latent defects, number of customers, quality of documentation, staff turnover, system age, code structure and complexity, and the use of standards and CASE tools. **Managing maintenance** uses **configuration and version management**, a **help desk**, **metrics** (mean time between failures, number of requests, backlog), and periodic decisions to **re-engineer or retire** the system.

Reverse engineering analyses code to recover design, and **re-engineering** restructures the system to improve maintainability. When maintenance costs exceed the benefit, a new SDLC cycle begins.

Example 5.2 — Classify the maintenance

Classify: (a) a login bug fix; (b) upgrade to support the new VAT rate; (c) adding a dashboard requested by the manager; (d) rewriting messy code for readability.

Solution

(a) **Corrective** (b) **Adaptive** (c) **Perfective** (d) **Preventive**.

Key Takeaways

- ✓ Implementation includes coding, testing, installation, documentation, training and user support.
- ✓ Testing proceeds from unit to integration, system and acceptance levels; use black-box, white-box and regression testing.
- ✓ Installation strategies are direct, parallel, single-location and phased — trading risk against cost.
- ✓ Maintenance is corrective, adaptive, perfective or preventive and often consumes most of the system's lifetime cost.
- ✓ Organizational issues, user training and change control are as important as technical work.

— *End of Unit 5* —

bcalink.com