UNIT–V

SOFTWARE MAINTENANCE

Student-Friendly • Exam-Oriented • Relatable Examples • Quick Revision

📚 SYLLABUS COVERAGE

Management of Maintenance • Maintenance Process • Maintenance Models • Reverse Engineering • Software Re-engineering • Configuration Management • Documentation

🏫 RUNNING CASE STUDY — COLLEGE CANTEEN ORDERING SYSTEM

Our software allows students to log in, select food, place orders, pay online and lets the canteen manage orders. After deployment, the software will still need fixes, changes, improvements and documentation. That continuing work is Software Maintenance.

The Big Idea

Software development does not end when the system is delivered. Real software changes because bugs are discovered, users request new features, technology changes, and the environment around the software changes.

🧠 MEMORY AID

DEVELOP → DELIVER → USE → CHANGE → MAINTAIN

⭐ EXAM TIP

Maintenance is not simply “fixing bugs.” It also includes adapting software, improving it and preventing future problems.

1. Management of Software Maintenance

Management of Software Maintenance is the planning, organizing, controlling and monitoring of maintenance activities after software has been delivered.

💡 **RELATABLE EXAMPLE**

After the canteen app is launched, students report a payment problem, the college changes its payment provider, and the canteen asks for a new discount feature. Someone must receive requests, prioritize them, assign developers, estimate effort, test changes and release updates. That is maintenance management.

1.1 Main Activities of Maintenance Management

- Receive and record maintenance requests.
- Analyze and classify each request.
- Prioritize requests according to urgency, impact and business value.
- Estimate effort, time and resources.
- Assign work to maintenance personnel.
- Control changes and releases.
- Test modified software.
- Maintain records, versions and documentation.
- Monitor maintenance cost, quality and progress.

1.2 Maintenance Team Responsibilities

Responsibility	Simple Meaning	Canteen Example
Request Management	Receive and track change/fix requests	Student reports payment failure
Impact Analysis	Identify what may be affected	Payment change may affect Order Confirmation
Planning	Decide people, time and resources	Assign developer + tester
Change Control	Approve and control modifications	Approve new UPI integration
Testing	Verify the change and existing behavior	Test UPI and old payment methods
Release Management	Deliver the updated version	Publish app update
Documentation	Keep technical/user records updated	Update payment API and user guide

1.3 Important Factors in Maintenance Management

- Maintainability — how easily software can be understood and changed.
- Team capability — knowledge of the system and technology.
- Software complexity — difficult systems generally require more maintenance effort.
- Documentation quality — good documentation reduces understanding time.
- Configuration and version control — prevents confusion about which version is being changed.
- Prioritization — critical failures usually receive higher priority than minor enhancements.

🧠 **MEMORY AID**

Maintenance Management = REQUEST → ANALYZE → PLAN → CHANGE → TEST → RELEASE → RECORD

2. Maintenance Process

The Maintenance Process is the sequence of activities used to receive a maintenance request, understand the required change, modify the software, test it and release the updated version.

2.1 Typical Maintenance Process

2.2 Maintenance Request Analysis

Before changing code, the maintenance team should understand what is requested and what parts of the system may be affected.

💡 RELATABLE EXAMPLE

Request: “Students should be able to cancel an order within 2 minutes.” The team must check the Order module, payment handling, database records, canteen notifications and existing cancellation rules before implementing it.

2.3 Impact Analysis

Impact Analysis identifies the possible effects of a proposed change on modules, data, interfaces, tests, documentation and other system components.

- Which modules will change?
- Which modules depend on them?
- Which database/data structures are affected?
- Which interfaces or APIs may be affected?
- Which test cases need to be repeated?
- Which documents need updating?

⭐ EXAM TIP

Never think of maintenance as “change one line and finish.” A change can affect many connected parts of a system.

3. Maintenance Models

A Maintenance Model describes how maintenance activities are organized and how changes are introduced into an existing software system.

3.1 Quick View of Common Maintenance Models

Model / Approach	Main Idea	When Useful	Simple Understanding
Quick-Fix Model	Make a small immediate correction with minimal analysis	Urgent/simple fixes	Fix quickly
Iterative Enhancement Model	Improve the existing system through repeated enhancement cycles	Continuous enhancement	Improve step-by-step
Re-engineering-oriented approach	Study and transform an existing legacy system to improve maintainability	Old/legacy systems	Modernize the old system

Different textbooks classify maintenance models differently. For an exam, follow the terminology and model names used by your prescribed textbook/teacher.

3.2 Quick-Fix Model

The Quick-Fix approach makes a direct change to solve a problem, often with limited analysis and documentation.

💡 RELATABLE EXAMPLE

If the canteen app displays the wrong label “Cash Payment” after an already-fixed payment method change, a small UI correction may be made quickly.

- Fast for small, obvious problems.
- Useful when immediate correction is necessary.
- Poor choice for complex changes because deeper impacts may be missed.

3.3 Iterative Enhancement Model

The system is maintained through planned, repeated cycles in which existing functionality is modified and new functionality is added.

💡 RELATABLE EXAMPLE

Version 1: Online ordering. Version 2: UPI payment. Version 3: Order cancellation. Version 4: Discount coupons. Each version improves the existing system.

🧠 MEMORY AID

Iterative Enhancement = CHANGE → TEST → RELEASE → REPEAT

3.4 Re-engineering-Oriented Maintenance

For difficult legacy software, maintenance may involve understanding the existing system and transforming parts of it into a more maintainable form rather than repeatedly patching the old code.

💡 RELATABLE EXAMPLE

A college has a 10-year-old desktop canteen application with outdated code and poor documentation. The team studies the old system, extracts its important behavior and rebuilds it using a modern architecture.

⭐ EXAM TIP

Maintenance models organize how changes are performed; choose a model according to urgency, complexity and the condition of the existing software.

4. Reverse Engineering

Reverse Engineering is the process of analyzing an existing software system to identify its components, relationships, behavior and design information, usually starting from lower-level implementation details and moving toward higher-level understanding.

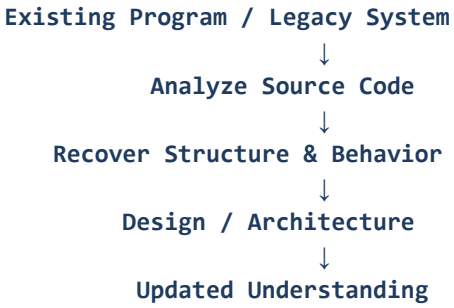
💡 RELATABLE EXAMPLE

Imagine receiving an old canteen application with source code but no proper design document. By studying the code, database and interfaces, the team reconstructs diagrams and understands how the original system works. This is reverse engineering.

4.1 Why is Reverse Engineering Needed?

- Understand legacy software.
- Recover missing or outdated design information.
- Improve documentation.
- Support maintenance and impact analysis.
- Help prepare a system for re-engineering or migration.
- Reduce dependence on undocumented knowledge.

4.2 Reverse Engineering Flow



4.3 Reverse Engineering vs Forward Engineering

Point	Reverse Engineering	Forward Engineering
Direction	Existing implementation → higher-level understanding	Requirements/design → implementation
Main Purpose	Understand/recover existing system	Build a new system
Starting Point	Existing software	Requirements/specification
Example	Recover design from old canteen code	Build canteen app from SRS/design

🧠 MEMORY AID

REVERSE = Code → Design/Understanding. FORWARD = Requirements/Design → Code.

⭐ EXAM TIP

Reverse engineering does not necessarily mean changing the system. Its primary goal is understanding/recovering information from the existing system.

5. Software Re-engineering

Software Re-engineering is the systematic examination, modification and transformation of existing software to improve its quality, maintainability or usefulness while preserving needed functionality.

💡 RELATABLE EXAMPLE

The college canteen has an old application that works but is difficult to maintain. The team may restructure its code, update the database, improve documentation and move it to a modern platform while keeping the important ordering functions.

5.1 Common Re-engineering Activities

Activity	Meaning	Example
Inventory Analysis	Identify and evaluate existing software/assets	List old canteen applications/modules
Document Restructuring	Improve poor/outdated documentation	Create current architecture and user guides
Reverse Engineering	Recover higher-level information from existing software	Recover design from source code
Code Restructuring	Improve code organization without changing required behavior	Break a large module into smaller modules
Data Restructuring	Improve data organization/storage	Redesign outdated database structures
Forward Engineering	Build the transformed system	Implement the improved architecture

5.2 Reverse Engineering vs Re-engineering

Reverse Engineering	Re-engineering
Mainly understanding/recovery	Understanding + transformation/improvement
Existing code → design/knowledge	Existing system → improved system
May not modify the system	Usually results in a modified/transformed system
Often supports re-engineering	Can use reverse engineering as one of its activities

🧠 **MEMORY AID**

Reverse Engineering = UNDERSTAND. Re-engineering = UNDERSTAND + IMPROVE.

⭐ **EXAM TIP**

A common exam trap: Reverse engineering and re-engineering are related, but they are not identical.

6. Configuration Management

Software Configuration Management (SCM) is the process of identifying, organizing, controlling and tracking changes to software artifacts throughout their life cycle.

💡 **RELATABLE EXAMPLE**

Suppose three developers work on the canteen system. Without configuration management, one may overwrite another's changes or test the wrong version. Version control and controlled releases help prevent this.

6.1 What is a Software Configuration Item (SCI)?

A Software Configuration Item is a software artifact placed under configuration control.

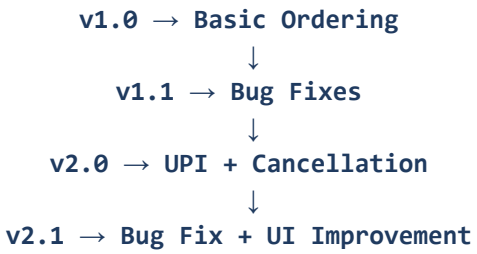
- Source code
- Requirements/SRS
- Design documents
- Test cases and test scripts
- Database scripts/schema
- Build and deployment files
- User documentation

6.2 Main SCM Activities

Activity	Meaning	Canteen Example

Configuration Identification	Identify items and versions to control	Order.java, SRS, test cases
Version Control	Maintain different versions systematically	v1.0, v1.1, v2.0
Change Control	Evaluate and approve changes	Approve new cancellation feature
Configuration Status Accounting	Record what changed and current status	Record version, change and approval
Configuration Audit	Check that approved changes are correctly implemented	Verify release contains approved files
Build/Release Management	Create and deliver controlled releases	Prepare v2.0 for deployment

6.3 Version Control — Simple Example



⭐ **EXAM TIP**

SCM controls changes to software artifacts. Version control is one important part of SCM, not the whole of SCM.

7. Documentation

Software Documentation is the collection of written or electronic information that explains software requirements, design, implementation, operation, testing, maintenance and use.

💡 **RELATABLE EXAMPLE**

When a new developer joins the canteen project, good documentation helps them understand the system without asking the original developer about every detail.

7.1 Types of Software Documentation

Type	Purpose	Example
Requirements Documentation	Describes what the system should do	SRS
Design Documentation	Explains architecture, modules, database and interfaces	Architecture/design document
Technical Documentation	Helps developers understand implementation	API notes, code comments, setup instructions
Test Documentation	Describes testing strategy, cases and results	Test plan, test cases, test report
User Documentation	Helps end users operate the software	Student user guide
Maintenance Documentation	Supports future changes and troubleshooting	Change history, known issues, maintenance notes

7.2 Good Documentation Characteristics

- Accurate — reflects the actual system.
- Clear — easy to understand.
- Complete — covers the information needed by its audience.
- Consistent — terminology and structure remain consistent.
- Up to date — changed software should have updated documentation.
- Easy to maintain — documents should be organized and version-controlled.

7.3 Why Documentation Matters in Maintenance

- Reduces the time needed to understand existing software.
- Helps developers perform impact analysis.
- Preserves knowledge when team members leave.
- Supports testing and troubleshooting.
- Helps users operate the system correctly.
- Provides evidence of requirements, decisions and changes.

⭐ **EXAM TIP**

Outdated documentation can become a maintenance problem itself. Documentation should be updated whenever significant system changes are made.

8. Four Common Types of Software Maintenance

Software maintenance is commonly classified by the reason for the change.

Type	Meaning	Canteen Example	Memory
Corrective	Fix defects discovered after delivery	Fix incorrect bill calculation	Correct a problem
Adaptive	Modify software to work in a changed environment	Update app for a new payment API/OS	Adapt to environment
Perfective	Improve performance, usability or add desired enhancements	Add coupons or improve checkout UI	Perfect the system

Preventive	Modify software to reduce future problems and improve maintainability	Refactor risky code before it causes failures	Prevent future problems

🧠 **MEMORY AID**

CAP: Corrective → Adaptive → Perfective → Preventive.

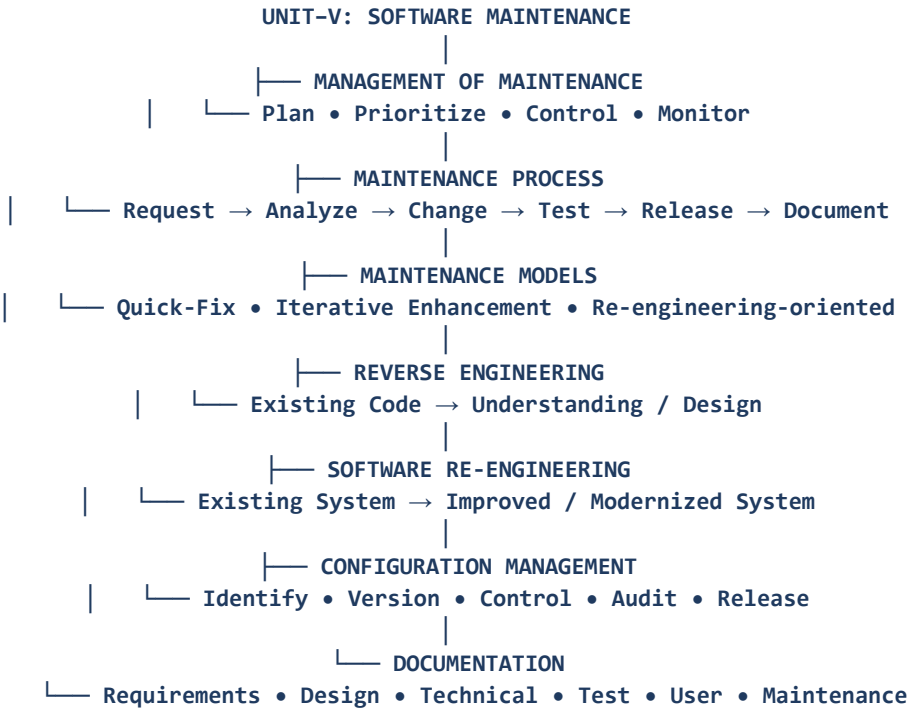
⭐ **EXAM TIP**

This classification is frequently asked in exams. Corrective fixes faults; Adaptive responds to environment changes; Perfective improves/enhances; Preventive reduces future maintenance problems.

9. Reverse Engineering, Re-engineering & Maintenance

Concept	Main Question	Main Goal	Example
Maintenance	What change/fix is needed in the existing system?	Keep/improve the operational system	Fix payment bug
Reverse Engineering	How does the existing system work?	Recover understanding/design	Recover design from legacy code
Re-engineering	How can the existing system be transformed?	Improve/modernize the system	Transform legacy canteen app

10. Unit-V at a Glance



11. Most Important One-Line Definitions

Topic	Exam-ready definition
Maintenance Management	Planning, organizing, controlling and monitoring maintenance activities after software delivery.
Maintenance Process	Sequence of activities used to analyze, modify, test and release changes to existing software.
Maintenance Model	An approach that organizes how maintenance changes are performed.

Reverse Engineering	Analyzing an existing system to recover its structure, design and behavior.
Software Re-engineering	Transforming existing software to improve maintainability, quality or usefulness while preserving needed functionality.
Configuration Management	Identifying, controlling, tracking and auditing changes to software artifacts.
Configuration Item	A software artifact placed under configuration control.
Documentation	Written/electronic information explaining software requirements, design, implementation, operation, testing and use.
Corrective Maintenance	Maintenance performed to correct defects.
Adaptive Maintenance	Maintenance performed to adapt software to a changed environment.
Perfective Maintenance	Maintenance performed to improve or enhance software.
Preventive Maintenance	Maintenance performed to reduce future problems and improve maintainability.

12. Last-Minute Memory Sheet

MAINTENANCE MANAGEMENT

Request → Analyze → Prioritize → Plan → Change → Test → Release → Record

FOUR MAINTENANCE TYPES

Corrective = Fix • Adaptive = Environment • Perfective = Improve • Preventive = Prevent future problems

REVERSE vs RE-ENGINEERING

Reverse = Understand/Recover • Re-engineering = Understand + Transform/Improve

SCM

Identify → Version Control → Change Control → Status Accounting → Audit → Release

DOCUMENTATION

Requirements + Design + Technical + Test + User + Maintenance

🏆 GOLDEN RULE

GOOD MAINTENANCE = CONTROLLED CHANGE + TESTING + UPDATED DOCUMENTATION

13. Practice Questions

1. Define Software Maintenance. Why is maintenance required after software delivery?
2. Explain Management of Maintenance and its major activities.
3. Explain the Software Maintenance Process with a suitable diagram.
4. What is impact analysis? Why is it important during maintenance?
5. Explain Maintenance Models with suitable examples.
6. Define Reverse Engineering. Explain why it is needed for legacy systems.
7. Differentiate Reverse Engineering and Forward Engineering.
8. Define Software Re-engineering. Explain its major activities.
9. Differentiate Reverse Engineering and Software Re-engineering.
10. What is Software Configuration Management? Why is it needed?
11. Explain Software Configuration Items and major SCM activities.
12. What is Version Control? Explain with an example.

13. What is Software Documentation? Explain its types.
14. Explain the importance of documentation in software maintenance.
15. Explain Corrective, Adaptive, Perfective and Preventive Maintenance.
16. Write short notes on Maintenance Management, Reverse Engineering and Configuration Management.

⭐ **EXAM TIP**

For a 5–10 mark answer: Definition → Explanation → Steps/Types → Relatable example → Diagram/table → Importance → Short conclusion.