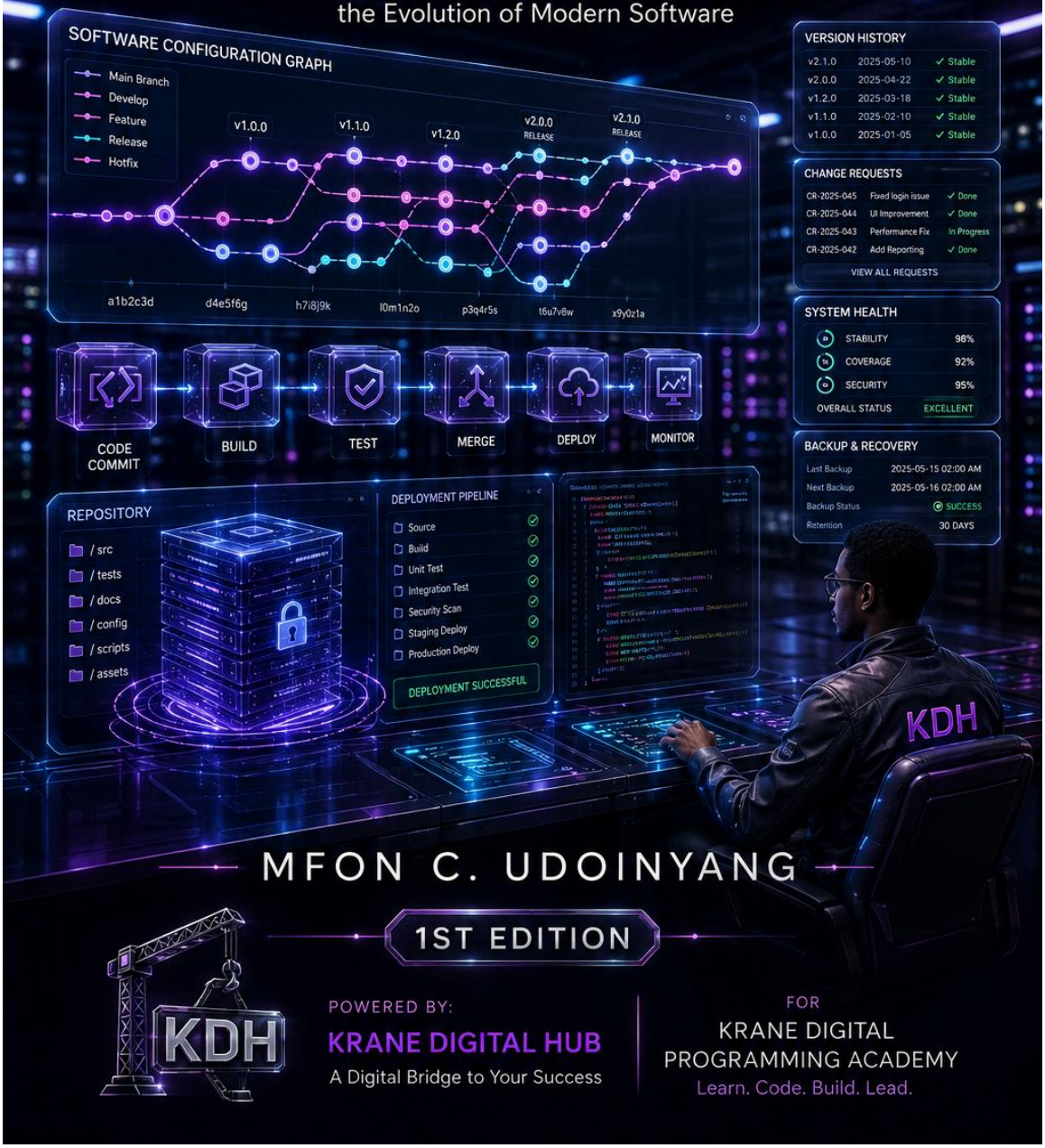
SOFTWARE CONFIGURATION AND MAINTENANCE

Managing Change, Versions, Reliability and the Evolution of Modern Software



MFON C. UDOINYANG

1ST EDITION

POWERED BY:
KRANE DIGITAL HUB
A Digital Bridge to Your Success

FOR
KRANE DIGITAL PROGRAMMING ACADEMY
Learn. Code. Build. Lead.

CSD 421 EXAM CRACKER

Software Configuration Management and Maintenance

in Computer Education

Author:

Mfon C. Udoinyang

Created for:

Krane Digital Programming Academy

Powered by:

Krane Digital Hub

Contact:

Email: kranedigitalystemz@gmail.com

Website: https://kranedigitalhub.netlify.app

WhatsApp: +234 810 472 7681

CSD 421: SOFTWARE CONFIGURATION MANAGEMENT AND MAINTENANCE IN COMPUTER EDUCATION

EXAM CRACKER: 48-HOUR MASTER NOTES

COMPACT 40-PAGE STUDY EDITION

Course Code: CSD 421
Course Title: Software Configuration Management and Maintenance in Computer Education
Level: 400 Level
Department: Computer and Robotics Education

HOW TO USE THIS BOOK

This is an exam-focused handbook, not a conventional textbook. It is designed to help a student understand, recall and reproduce major CSD 421 concepts under examination pressure.

The complete study package will contain:

1. Core lessons
2. Important definitions
3. Explanations and examples
4. Process diagrams
5. Assignment topics
6. Exam traps and distinctions
7. Memory Boxes
8. RAIDs/Rapid Recall
9. Chapter quizzes
10. 100 examination questions and answers
11. Flashcards
12. Final revision sheet

Core principle:

```
UNDERSTAND → CONNECT → RECALL → PRACTICE → ANSWER
```

MASTER COURSE MAP

```
SOFTWARE → INSTALLATION → OPERATING SYSTEM → APPLICATIONS → DRIVERS →
UTILITIES → MAINTENANCE → CONFIGURATION → CHANGE CONTROL → VERSION CONTROL →
AUDIT → RELEASE → QUALITY → RELIABILITY
```

PART I: SOFTWARE FOUNDATIONS AND INSTALLATION

CHAPTER 1: INTRODUCTION TO SOFTWARE

1.1 Meaning of Software

Software is a collection of programs, instructions, data, procedures and associated documentation that directs a computer system to perform specific tasks.

Examples include Windows, Linux, Android, Microsoft Word, web browsers, database systems, antivirus applications and educational software.

Simple Definition

Software is the set of logical instructions that tells computer hardware what to do.

Hardware vs Software

Hardware: Physical components that can be touched.

Examples:

- CPU
- RAM
- Hard disk/SSD
- Keyboard
- Monitor
- Motherboard
- Printer

Software: Programs and instructions executed by hardware.

Examples:

- Operating systems
- Applications
- Utilities
- Device drivers

Relationship

```
USER → SOFTWARE → OPERATING SYSTEM → HARDWARE
```

The operating system acts as an important intermediary between application software, users and hardware.

1.2 Major Types of Software

Software can broadly be classified into:

1. **System software**
2. **Application software**
3. **Utility software**
4. **Programming/development software**
5. **Firmware**

1. System Software

System software manages computer resources and provides a platform on which application software operates.

Examples:

- Windows
- Linux
- macOS
- Android
- Device drivers
- System utilities

The most important system software is the **Operating System (OS).**

2. Application Software

Application software performs tasks directly useful to the user.

Examples:

- Microsoft Word
- Excel
- PowerPoint
- Google Chrome
- Photoshop
- VLC
- Educational CBT applications
- Accounting software
- Hospital management systems

3. Utility Software

Utility software performs maintenance, protection, optimization and management tasks.

Examples:

- Antivirus
- Backup software
- Disk cleanup tools
- File compression tools
- Disk management tools
- System diagnostic tools

4. Programming/Development Software

These are tools used to create other software.

Examples:

- Compilers
- Interpreters
- Assemblers
- Debuggers
- Integrated Development Environments (IDEs)
- Linkers

Examples of IDEs:

- Visual Studio Code
- Visual Studio
- IntelliJ IDEA

- Eclipse

5. Firmware

Firmware is software embedded in hardware that provides low-level control of the device.

Examples:

- BIOS/UEFI firmware
- Router firmware
- Printer firmware
- Smartphone firmware

1.3 Software Components

A complete software product may contain:

```
SOURCE CODE + EXECUTABLE FILES + CONFIGURATION FILES + DATA + DOCUMENTATION +
DEPENDENCIES
```

This becomes extremely important in SCM.

For example, a web application may consist of:

- HTML files
- CSS files
- JavaScript files
- Images
- JSON files
- Database
- Environment configuration
- Third-party libraries
- Documentation

If one important component is changed without controlling the others, the system may stop working.

This is one reason Software Configuration Management is necessary.

1.4 Software Product vs Software Configuration

A **software product** is the complete software delivered to users.

A **software configuration** is the specific collection of software components, versions, settings and related items that make up a particular state of the product.

Example:

```
SchoolApp v1.0
```

may contain:

- Frontend v1.0
- Backend v1.0
- Database schema v1.0
- Configuration file v1.0
- Documentation v1.0

Together, these form a particular configuration.

MEMORY BOX 1

Software = instructions + associated data/documentation.

System software = manages the computer.

Application software = performs user tasks.

Utility software = maintains/protects/optimizes the computer.

Firmware = low-level software embedded in hardware.

RAPID RECALL 1

If asked to list five types of software:

System → Application → Utility → Programming → Firmware

Mnemonic:

SAUPF

CHAPTER 1 QUIZ

1. What is software?
2. State four types of software.
3. Differentiate system software from application software.
4. Give four examples of utility software.
5. What is firmware?
6. Why is software documentation important?
7. How does software configuration differ from a software product?

CHAPTER 2: OPERATING SYSTEMS

2.1 Meaning of an Operating System

An Operating System (OS) is system software that manages computer hardware and software resources and provides services for application programs and users.

Examples:

- Microsoft Windows
- Linux
- macOS
- Android
- iOS
- Unix

Simple Definition

An operating system is the manager of the computer system.

It controls resources and provides an environment in which applications can run.

2.2 Major Functions of an Operating System

An operating system performs several important functions.

1. Process Management

The OS manages running programs and allocates CPU time to processes.

2. Memory Management

It manages RAM and determines how memory is allocated to programs.

3. File Management

It creates, organizes, stores, retrieves and protects files and directories.

4. Device Management

It manages hardware devices through device drivers.

5. Security Management

It controls users, permissions, authentication and access to system resources.

6. User Interface

The OS provides interfaces through which users interact with the computer.

Two major interfaces are:

- **CLI:** Command Line Interface
- **GUI:** Graphical User Interface

7. Resource Management

The OS manages:

- CPU
- RAM
- Storage
- Input/output devices
- Network resources

8. Error Management

The OS detects and reports certain hardware and software errors.

9. Networking

Modern operating systems provide network communication and resource-sharing facilities.

10. Application Support

The OS provides services and system interfaces that applications require to execute.

2.3 Components of an Operating System

Important components include:

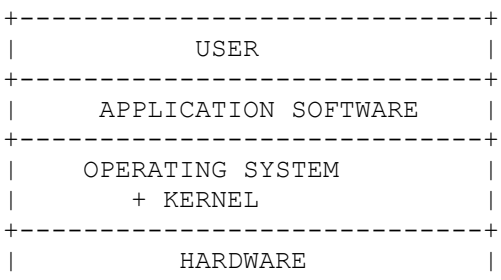
- Kernel
- Process manager
- Memory manager
- File system
- Device manager
- Security subsystem
- User interface
- Networking subsystem
- System utilities

Kernel

The **kernel** is the central part of an operating system.

It communicates closely with hardware and manages critical resources.

Simplified Architecture



+----------------------------+

2.4 Why Operating System Installation Is Important

Installing an operating system provides the fundamental software environment required for a computer to operate.

Importance includes:

1. Provides a platform for application software.
2. Manages hardware resources.
3. Provides file-management facilities.
4. Provides security mechanisms.
5. Enables user interaction.
6. Supports networking.
7. Provides device management.
8. Enables software installation.
9. Provides system services.
10. Allows the computer to become usable for productive tasks.

Without an appropriate operating system, a general-purpose computer cannot conveniently provide its normal user environment.

2.5 Types of Operating System Installation

Common approaches include:

Clean Installation

A new OS is installed on a blank or reformatted system partition.

Advantages:

- Fresh environment
- Removes accumulated software clutter
- Useful when the existing installation is severely damaged

Disadvantages:

- Existing applications may be removed
- Data can be lost if not backed up

Upgrade Installation

A newer version replaces or updates an existing version while attempting to preserve compatible applications and data.

Dual-Boot Installation

Two operating systems are installed on the same computer and the user chooses which one to start.

Network/Remote Installation

The operating system is installed across a network using centralized deployment mechanisms.

Virtual Machine Installation

An OS is installed inside a virtual machine managed by virtualization software.

CHAPTER 3: OPERATING SYSTEM INSTALLATION PROCESS

3.1 General OS Installation Workflow

```
CHECK REQUIREMENTS
        ↓
BACK UP IMPORTANT DATA
        ↓
PREPARE INSTALLATION MEDIA
        ↓
CONFIGURE BIOS/UEFI BOOT
        ↓
START INSTALLER
        ↓
SELECT LANGUAGE/REGION
        ↓
ACCEPT LICENSE
        ↓
SELECT INSTALLATION TYPE
        ↓
PARTITION/SELECT STORAGE
        ↓
COPY SYSTEM FILES
```

```
        ↓
INSTALL OS COMPONENTS
        ↓
RESTART COMPUTER
        ↓
COMPLETE INITIAL SETUP
        ↓
INSTALL DRIVERS/UPDATES
        ↓
TEST SYSTEM
```

3.2 Step 1: Check System Requirements

Before installation, verify:

- Processor requirements
- RAM requirements
- Storage capacity
- Graphics requirements
- Firmware compatibility
- Network requirements
- Supported architecture
- Application compatibility

Failure to check requirements can result in unsuccessful installation or poor performance.

3.3 Step 2: Back Up Data

Important files should be copied to:

- External storage
- Another computer
- Network storage
- Cloud storage

Backup is especially important before a clean installation or disk repartitioning.

Exam Point

Never assume installation is automatically safe for existing data.

3.4 Step 3: Prepare Installation Media

Installation media may include:

- Bootable USB drive
- DVD
- Network installation source
- Recovery partition
- Virtual installation image

A bootable USB is commonly used for modern computers.

3.5 Step 4: BIOS/UEFI Configuration

The system firmware initializes hardware and helps start the boot process.

The technician may need to:

- Set boot priority
- Select USB/DVD/network boot
- Configure secure boot where applicable
- Confirm storage settings
- Save configuration

BIOS vs UEFI

BIOS: Older firmware interface.

UEFI: Modern firmware environment with improved capabilities and commonly used with modern operating systems.

3.6 Step 5: Boot From Installation Media

The computer starts from the selected installation source.

The installer loads the required installation environment.

3.7 Step 6: Select Language and Regional Settings

The installer may request:

- Language
- Keyboard layout
- Time zone
- Region

3.8 Step 7: Select Installation Type

Depending on the OS, options may include:

- Upgrade
- Custom/Clean installation
- Repair/recovery

The correct choice depends on the user's requirements.

3.9 Step 8: Partition the Storage Device

Storage can be divided into logical sections called **partitions**.

A partition may contain:

- Operating system files
- User files
- Recovery files
- Other operating systems

Important Terms

Partition: A logical division of a storage device.

File system: A structure used to organize and manage files on storage.

Examples:

- NTFS
- FAT32
- exFAT
- ext4

3.10 Step 9: Copy and Install System Files

The installer copies necessary files to the storage device and configures the operating system.

This may involve:

- Extracting files
- Installing system components
- Configuring boot files
- Installing services
- Creating system directories

3.11 Step 10: Restart

After the main installation phase, the computer usually restarts.

The system should then boot from the newly installed OS rather than restarting the installation process.

3.12 Step 11: Initial Configuration

The user may configure:

- Username
- Password
- Computer name
- Network
- Privacy options
- Region
- Time settings
- Security settings

3.13 Step 12: Activate the Operating System

Where applicable, the OS is activated using:

- Product key
- Digital license
- Organization's licensing system

Activation requirements depend on the operating system and license.

3.14 Step 13: Install Updates

Immediately after installation, check for relevant:

- Security updates
- Feature updates
- Bug fixes
- Driver updates

An unpatched operating system may expose the computer to known vulnerabilities.

CHAPTER 4: POST-INSTALLATION PROCESS

Installing the operating system is **not the end of installation**.

The post-installation phase prepares the computer for reliable use.

Major Post-Installation Activities

```
OS INSTALLED
     ↓
VERIFY BOOT
     ↓
INSTALL/VERIFY DRIVERS
     ↓
INSTALL SYSTEM UPDATES
```

```
     ↓
CONFIGURE SECURITY
     ↓
CONFIGURE NETWORK
     ↓
INSTALL REQUIRED APPLICATIONS
     ↓
RESTORE USER DATA
     ↓
CONFIGURE BACKUP
     ↓
TEST HARDWARE/SOFTWARE
     ↓
DOCUMENT CONFIGURATION
     ↓
CREATE RECOVERY POINT/IMAGE
```

4.1 Install Device Drivers

Drivers allow the operating system to communicate with hardware.

Examples:

- Display driver
- Audio driver
- Network driver
- Printer driver
- Chipset driver
- Touchpad driver
- Bluetooth driver

4.2 Configure Security

Security configuration may include:

- User passwords
- User permissions
- Firewall
- Antivirus/endpoint protection
- Automatic updates
- Secure authentication
- Disk encryption where appropriate

4.3 Configure Network

Set up:

- Wi-Fi
- Ethernet
- IP configuration
- DNS
- Proxy where necessary
- Network authentication

4.4 Install Required Applications

Examples:

- Office software
- Browser
- PDF reader
- Development tools
- Educational software
- Communication applications
- Database software

4.5 Restore User Data

Previously backed-up files can be restored after confirming that the new environment is functioning correctly.

4.6 Configure Backup

A reliable computer should have a backup strategy.

Possible approaches:

```
LOCAL BACKUP + EXTERNAL BACKUP + CLOUD BACKUP
```

The exact approach depends on the importance and sensitivity of the data.

4.7 Test the Installation

Test:

- Booting
- Keyboard
- Mouse/touchpad
- Display
- Audio
- Network
- USB ports
- Storage
- Applications
- Printing
- Updates

4.8 Document the Configuration

Record important configuration information such as:

- OS version
- Installed drivers
- Installed applications
- Configuration settings
- License information
- Network settings
- Hardware details
- Date of installation

Why is this important to CSD 421?

Because documentation creates a known configuration that can later be compared against future changes.

That is the bridge from **installation** to **configuration management**.

MEMORY BOX 2

OS Installation = Build the environment.

Post-installation = Stabilize, secure, configure, test and document the environment.

SCM = Control and track changes to that environment and its software components.

CHAPTER 4 QUIZ

1. Why should data be backed up before OS installation?
2. State five post-installation activities.
3. What is a device driver?
4. Differentiate BIOS and UEFI.
5. What is a partition?
6. Why should an OS be updated after installation?
7. Why is documentation important after installation?
8. Explain why post-installation activities are relevant to SCM.

CHAPTER 5: APPLICATION SOFTWARE

5.1 Meaning of Application Software

Application software consists of programs designed to help users perform specific tasks.

Examples:

- Word processors
- Spreadsheets
- Presentation software
- Browsers
- Graphic design software
- Media players
- Educational applications
- Database applications
- Accounting applications

5.2 Categories of Application Software

General-Purpose Application Software

Designed for common tasks.

Examples:

- Word processors
- Spreadsheet software
- Presentation software
- Web browsers

Special-Purpose Application Software

Designed for a particular task or organization.

Examples:

- Hospital management system
- School management system
- Banking software
- Payroll system

Educational Application Software

Designed to support teaching and learning.

Examples:

- CBT software
- Learning management systems
- Simulation software
- Educational games
- Intelligent tutoring systems

Business Application Software

Examples:

- Accounting
- Inventory management
- Payroll

- Customer relationship management

Multimedia Software

Examples:

- Video editors
- Audio players
- Image editors

5.3 Steps in Installing Application Software

```
IDENTIFY SOFTWARE
      ↓
CHECK SYSTEM REQUIREMENTS
      ↓
OBTAIN TRUSTED INSTALLER
      ↓
CHECK LICENSE
      ↓
BACK UP/PROTECT EXISTING DATA
      ↓
RUN INSTALLER
      ↓
ACCEPT LICENSE TERMS
      ↓
SELECT INSTALLATION LOCATION
      ↓
SELECT COMPONENTS/OPTIONS
      ↓
INSTALL DEPENDENCIES
      ↓
COMPLETE INSTALLATION
      ↓
LAUNCH APPLICATION
      ↓
ACTIVATE IF REQUIRED
      ↓
UPDATE/PATCH
      ↓
TEST
      ↓
DOCUMENT VERSION/CONFIGURATION
```

5.4 System Requirements

Before installing software, determine whether the computer meets:

- CPU requirement
- RAM requirement
- Storage requirement
- OS requirement
- Graphics requirement
- Network requirement
- Dependency requirement

Compatibility

Software compatibility means that the application can operate correctly within the target hardware and software environment.

5.5 Software Dependencies

A dependency is another component required for software to function.

Examples include:

- Runtime libraries
- Frameworks
- Drivers
- Database engines
- APIs
- Package libraries

Example

A program may require:

```
Application → .NET Runtime → Operating System → Hardware
```

If the required runtime is missing or incompatible, the application may fail to run.

5.6 Installation Options

Some installers allow users to select:

- Typical installation
- Custom installation

- Minimal installation
- Full installation

Typical: Installs recommended components.

Custom: Allows the user to select components and locations.

Full: Installs most or all available components.

CHAPTER 6: UTILITY SOFTWARE

6.1 Meaning of Utility Software

Utility software is system-support software designed to maintain, protect, manage, diagnose or optimize computer resources.

Major Types

1. Antivirus/anti-malware
2. Backup utilities
3. Disk management utilities
4. File compression utilities
5. File recovery utilities
6. System monitoring utilities
7. Disk cleanup tools
8. Diagnostic tools
9. Encryption utilities
10. File-management tools

6.2 Importance of Utility Software

Utility software can:

- Improve system security
- Protect data
- Detect faults
- Manage storage
- Recover files
- Monitor performance
- Compress files

- Support backups
- Assist troubleshooting

CHAPTER 7: DEVICE DRIVERS

7.1 Meaning of Device Driver

A device driver is system software that enables the operating system to communicate with and control a particular hardware device.

Simplified Model

```
APPLICATION
     ↓
OPERATING SYSTEM
     ↓
DEVICE DRIVER
     ↓
HARDWARE DEVICE
```

Example:

```
Music App → OS → Audio Driver → Sound Card → Speaker
```

7.2 Examples of Drivers

- Graphics/display driver
- Printer driver
- Audio driver
- Network adapter driver
- Bluetooth driver
- Storage controller driver
- Camera driver
- Touchpad driver

7.3 Why Drivers Matter During Installation

After installing an operating system, some hardware may not function correctly until appropriate drivers are installed.

Symptoms of missing or faulty drivers include:

- No sound
- Poor display resolution
- Wi-Fi unavailable
- Bluetooth unavailable
- Printer not detected
- Touchpad malfunction
- Unknown devices in device management

7.4 Driver Maintenance

Drivers should be:

- Correct for the hardware
- Compatible with the OS
- Obtained from trusted sources
- Updated when necessary
- Documented where important

Exam Trap

A driver is not the physical hardware.

The driver is software that enables the OS to communicate with hardware.

CHAPTER 8: BOOTSTRAP LOADER

8.1 Meaning

A **bootstrap loader**, commonly called a bootloader, is a program responsible for initiating the operating-system startup process.

When a computer starts, firmware performs initial hardware initialization and locates the boot code needed to load the operating system.

Simplified Boot Sequence

```
POWER ON
   ↓
FIRMWARE (BIOS/UEFI)
   ↓
BOOT/BOOTLOADER
   ↓
OPERATING SYSTEM KERNEL
   ↓
SYSTEM SERVICES
   ↓
LOGIN/USER ENVIRONMENT
```

8.2 Functions of the Bootstrap Loader

It generally:

1. Initiates the boot process.
2. Locates the operating system.
3. Loads or transfers control to the OS loader/kernel.
4. Helps establish the environment required for the OS to start.

Exam Definition

A bootstrap loader is a small program or boot mechanism that initiates the loading of the operating system when a computer starts.

CHAPTER 9: TRANSLATORS

9.1 Meaning of Language Translator

A language translator is system software that converts instructions written in one programming language into a form that a computer can execute or process.

Major translators include:

1. Assembler
2. Compiler
3. Interpreter

9.2 Assembler

An assembler translates **assembly language** into machine code.

```
Assembly Language → Assembler → Machine Code
```

9.3 Compiler

A compiler translates an entire source program, or substantial parts of it, into another form such as machine code or intermediate code before execution.

```
High-Level Language → Compiler → Object/Executable Code
```

Examples of compiled languages include C and C++.

9.4 Interpreter

An interpreter executes or translates program instructions progressively during execution rather than producing a traditional standalone machine-code executable in one compilation step.

```
Source Code → Interpreter → Execution
```

9.5 Compiler vs Interpreter

Compiler	Interpreter
Translates program before execution in the traditional model	Processes/executes instructions during execution
Often produces object/executable output	Typically does not produce a traditional standalone executable from the whole program
Errors may be reported after compilation analysis	Errors can appear as execution reaches affected code
Compiled programs can execute efficiently after compilation	Interpretation can introduce runtime translation overhead

---

CHAPTER 10: COMMON SOFTWARE INSTALLATION PROBLEMS

Software installation can fail for technical, environmental, user, security or configuration reasons.

10.1 Insufficient Storage

Cause:

The computer does not have enough free storage.

Solution:

- Remove unnecessary files
- Increase storage
- Choose another installation location
- Free disk space

10.2 Incompatible Operating System

Cause:

The application does not support the installed OS version or architecture.

Solution:

- Install a compatible version
- Upgrade the OS if appropriate
- Use a supported alternative
- Check vendor requirements

10.3 Insufficient RAM

Cause:

The application requires more memory than is available.

Solution:

- Close unnecessary applications
- Increase RAM where possible
- Use a lighter application/version

10.4 Missing Dependencies

Cause:

Required runtime libraries, frameworks or components are missing.

Solution:

Install the required dependency from a trusted source.

10.5 Corrupted Installer

Cause:

Installation package was incomplete, damaged or improperly downloaded.

Solution:

- Download again
- Verify source
- Verify file integrity where possible
- Use an official installer

10.6 Permission Problems

Cause:

The user account does not have sufficient privileges.

Solution:

- Use an authorized administrative account

- Adjust permissions appropriately
- Follow organizational installation policies

10.7 Malware or Security Blocking

Security software may prevent suspicious or unauthorized installers from executing.

Solution:

- Verify the source
- Scan the installer
- Follow organizational security policy
- Never blindly disable security controls

10.8 Driver Conflict

A new application may conflict with an existing driver.

Solution:

- Update the driver
- Roll back a problematic driver
- Check compatibility
- Consult vendor documentation

10.9 Network Failure

Some applications require internet access during installation.

Causes:

- Poor connectivity
- DNS failure
- Firewall restrictions
- Server unavailable

Solutions:

- Check connection
- Verify network settings
- Retry later
- Use an approved offline installer where available

10.10 Version Conflict

A newer or older component may conflict with an existing version.

Solution:

- Check supported versions
- Update compatible components
- Remove obsolete versions where safe
- Use dependency/version management

10.11 Installation Problem Analysis Model

Use:

P-C-S

```
PROBLEM → CAUSE → SOLUTION
```

Example:

Problem: Application refuses to install.
Cause: Insufficient storage.
Solution: Free storage or install on another suitable drive.

10.12 Important Installation Troubleshooting Sequence

```
IDENTIFY ERROR
     ↓
READ ERROR MESSAGE
     ↓
CHECK SYSTEM REQUIREMENTS
```

```
     ↓
CHECK STORAGE/RAM
     ↓
CHECK OS COMPATIBILITY
     ↓
CHECK DEPENDENCIES
     ↓
CHECK PERMISSIONS
     ↓
CHECK SECURITY/NETWORK
     ↓
CHECK LOGS
     ↓
RETRY OR ROLLBACK
     ↓
DOCUMENT SOLUTION
```

CHAPTER 10 MEMORY BOX

The 7 major installation checks:

R-S-O-D-P-N-S

- **R** = Requirements
- **S** = Storage
- **O** = Operating system
- **D** = Dependencies
- **P** = Permissions
- **N** = Network
- **S** = Security

CHAPTER 10 QUIZ

1. State five causes of software installation failure.
2. Explain how insufficient storage can prevent installation.
3. What is a software dependency?
4. How can permission problems be solved?
5. Why should software be downloaded from trusted sources?
6. What is the importance of installation logs?
7. Explain the Problem-Cause-Solution troubleshooting model.
8. What should a technician check before reinstalling failed software?

CHAPTER 11: THE CONNECTION BETWEEN INSTALLATION AND SCM

This is an important bridge into the main course.

At first glance, software installation and Software Configuration Management may appear unrelated. They are not.

When software is installed, a particular configuration is created.

For example:

```
COMPUTER SYSTEM
│
├── Operating System vX
├── Driver vY
├── Application v3.2
├── Runtime v8
├── Database v5
└── Configuration File v1
```

If the application works correctly, this configuration can become a **known baseline**.

Later, suppose someone changes:

```
Application v3.2 → v3.3
```

and the application stops working.

SCM helps answer:

- What changed?
- Who changed it?
- When did it change?
- Which component was affected?
- What version worked previously?
- Can we restore the previous version?
- What other components depend on it?
- Was the change approved?
- Was it tested?

This is where installation, maintenance and SCM meet.

11.1 Configuration State

A configuration state describes the specific versions and settings of components at a particular point in time.

Example:

```
School Management System Configuration – 15 August
```

- OS: Windows 11
- Database: MySQL X
- Backend: Node.js X
- Frontend: React X
- Browser: Chrome X
- Driver: Version X
- Configuration file: Production-v4

This information is valuable when troubleshooting and maintaining the system.

11.2 Baseline

A **baseline** is a formally identified and controlled version of a configuration item or collection of configuration items that serves as a reference point for future work.

Simple Example

A school deploys:

```
School Portal v1.0
```

After testing and approval, it becomes the **baseline**.

Future changes are made from this controlled state.

```
APP v1.0
  ↓
TEST
  ↓
APPROVE
  ↓
BASELINE
  ↓
CHANGE REQUEST
  ↓
```

```
MODIFICATION
  ↓
TEST
  ↓
NEW BASELINE v1.1
```

11.3 Why Baselines Matter

Baselines provide:

- Reference points
- Stability
- Traceability
- Controlled change
- Easier rollback
- Reproducibility
- Better auditing

EXAM CONNECTION

If the question says:

"How does SCM support software maintenance?"

Do not answer only:

"SCM controls software changes."

Expand it:

SCM supports maintenance by identifying software components, controlling changes, recording versions, establishing baselines, maintaining configuration records, supporting audits, enabling rollback, improving traceability and ensuring that modifications are properly tested and approved.

That is a stronger 400-level answer.

CHAPTER 11 RAPID RECALL

Installation creates a configuration.

Documentation records the configuration.

Baseline establishes a controlled reference state.

SCM controls subsequent changes.

Maintenance modifies the software to preserve or improve its usefulness.

Therefore:

```
INSTALL → CONFIGURE → DOCUMENT → BASELINE → CHANGE → TEST → RELEASE →
MAINTAIN
```

ASSIGNMENT REVISION CONNECTION

Your assignment:

"How does SCM support software maintenance?"

is directly connected to this chapter and the major SCM chapters that follow.

Your other questions:

1. **Discuss the challenges associated with SCM.**
2. **Explain how SCM improves software quality and reliability.**

will be treated as dedicated examination topics later in the handbook.

CHAPTER 11 QUIZ

1. What is a software configuration?
2. What is a baseline?
3. Why is a baseline useful?
4. How does installation relate to SCM?
5. Why should software configurations be documented?
6. How does SCM help identify the cause of a failed update?
7. Explain the relationship between installation, configuration, maintenance and SCM.

MASTER MEMORY BOX: PART I

SOFTWARE FOUNDATION

Software
→ Programs/instructions/data/documentation

System Software
→ Manages computer resources

Application Software
→ Performs user tasks

Utility Software
→ Maintains/protects/optimizes

Operating System
→ Manages hardware/resources and provides platform for applications

Device Driver
→ Connects OS to hardware

Bootstrap Loader
→ Initiates OS startup

Assembler
→ Assembly → machine code

Compiler
→ Translates source program before execution in the traditional compilation model

Interpreter
→ Translates/processes during execution

Installation
→ Puts software into a usable environment

Post-installation
→ Updates + drivers + security + configuration + testing + documentation

Configuration
→ Specific collection of versions/settings/components

Baseline
→ Controlled reference configuration

SCM
→ Controls and tracks changes to software configurations

Maintenance
→ Corrects, adapts, improves and preserves software throughout its useful life

---

PART I SUPER-QUIZ

A. Definitions

1. Define software.
2. Define system software.
3. Define application software.
4. Define utility software.
5. Define operating system.
6. Define device driver.
7. Define bootstrap loader.
8. Define compiler.
9. Define interpreter.
10. Define software configuration.
11. Define baseline.
12. Define software maintenance.
13. Define SCM.

B. Short Answers

14. State five functions of an operating system.
15. List five examples of application software.
16. List five types of utility software.
17. State five post-installation activities.
18. State five common installation problems.
19. State five causes of installation failure.
20. State five solutions to installation problems.
21. Why is backup important before OS installation?
22. Why are device drivers important?
23. Why is software documentation important?
24. Why is a baseline important in SCM?

C. Essay Questions

25. Explain the steps involved in installing an operating system.
26. Discuss the importance of post-installation activities.
27. Explain the different types of application software.
28. Discuss the causes and solutions to common software installation problems.
29. Explain the role of device drivers in a computer system.
30. Explain how software installation is related to Software Configuration Management.

48-HOUR EXAM TIP

Do not memorize the entire chapter word-for-word.

Memorize the **chains**:

OS installation:

```
REQUIREMENTS → BACKUP → MEDIA → BIOS/UEFI → INSTALL → PARTITION → CONFIGURE → DRIVERS → UPDATE → TEST
```

Application installation:

```
REQUIREMENTS → INSTALLER → LICENSE → LOCATION → DEPENDENCIES → INSTALL → ACTIVATE → UPDATE → TEST
```

Troubleshooting:

```
PROBLEM → CAUSE → CHECK → SOLUTION → TEST → DOCUMENT
```

Configuration:

```
IDENTIFY → CONTROL → RECORD → BASELINE → CHANGE → TEST → AUDIT → RELEASE
```

The second half of that final chain is where **CSD 421 truly begins to bite.**

CSD 421: SOFTWARE CONFIGURATION MANAGEMENT AND MAINTENANCE IN COMPUTER EDUCATION

EXAM CRACKER: FINAL 48-HOUR MASTER NOTES

COMPACT 40-PAGE STUDY EDITION + 100 QUESTIONS + FLASHCARDS + RAIDs + MEMORY BOXES + QUIZZES

CHAPTER 12: SOFTWARE MAINTENANCE FUNDAMENTALS

12.1 Meaning of Software Maintenance

Software maintenance is the process of modifying, correcting, adapting, improving and supporting existing software after its initial delivery and throughout its useful life.

Exam Definition

Software maintenance is the modification of existing software after delivery to correct faults, adapt it to environmental changes, improve it or prevent future problems while preserving its integrity.

Maintenance is therefore much more than fixing bugs.

```
DELIVERY → OPERATION → CHANGE REQUEST → ANALYSIS → MODIFICATION → TEST → RELEASE → CONTINUED MAINTENANCE
```

12.2 Why Software Requires Maintenance

Software requires maintenance because:

1. Errors may be discovered after deployment.
2. User requirements change.
3. Operating systems change.
4. Hardware changes.
5. Security threats evolve.
6. Laws and regulations change.
7. Organizations change their business processes.
8. Performance may need improvement.
9. New features may be required.
10. Software dependencies may become obsolete.
11. Interfaces and external systems may change.
12. Technology may become outdated.

12.3 Software Evolution

Software evolution refers to the continuing modification of software in response to changing requirements, technology, environments and user expectations.

A successful software system does not remain permanently unchanged.

```
INITIAL SOFTWARE → CORRECTIONS → ADAPTATION → ENHANCEMENT → REENGINEERING → RETIREMENT
```

12.4 Maintenance vs Development

Development: Creates a new software system.

Maintenance: Modifies an existing software system.

However, maintenance can involve substantial development work.

Example:

Adding an online payment module to an existing school management system is maintenance, even though developers may write thousands of new lines of code.

CHAPTER 13: TYPES OF SOFTWARE MAINTENANCE

The four classical categories are:

1. Corrective
2. Adaptive
3. Perfective
4. Preventive

MEMORY CODE: CAPP

Corrective
Adaptive
Perfective
Preventive

13.1 Corrective Maintenance

Corrective maintenance fixes errors, defects and failures discovered during operation.

Examples:

- Fixing login failure
- Correcting wrong calculations
- Fixing broken links
- Repairing a database error
- Correcting incorrect examination scores

Purpose: Restore correct operation.

Keyword:

BUG FIX

13.2 Adaptive Maintenance

Adaptive maintenance modifies software so that it continues to operate in a changed environment.

Environmental changes may include:

- New operating system
- New hardware
- New database version
- New browser
- New external API
- New legislation
- New organizational policy

Example:

A school portal must be modified because the university changes its authentication system.

Purpose: Adapt software to environmental change.

Keyword:

ENVIRONMENT

13.3 Perfective Maintenance

Perfective maintenance improves existing software based on new user requirements or desired enhancements.

Examples:

- Adding dark mode
- Improving search
- Adding report generation
- Improving interface
- Increasing performance
- Adding new features

Purpose: Improve functionality, usability or performance.

Keyword:

IMPROVEMENT

13.4 Preventive Maintenance

Preventive maintenance modifies software to reduce the likelihood of future failures or make future maintenance easier.

Examples:

- Refactoring complicated code
- Improving documentation
- Removing dead code
- Updating obsolete libraries
- Improving architecture
- Adding automated tests

Purpose: Prevent future problems.

Keyword:

PREVENTION

CAPP EXAM TABLE

Type	Main Purpose	Example
Corrective	Fix existing defects	Repair login bug
Adaptive	Adjust to environment	Support new OS
Perfective	Improve/add functionality	Add search feature
Preventive	Reduce future problems	Refactor poor code

EXAM TRAP

Do not confuse:

Corrective = existing fault

Preventive = future fault

CHAPTER 14: SOFTWARE MAINTENANCE PROCESS

A general maintenance process is:

```
USER/INCIDENT/CHANGE REQUEST
          ↓
REQUEST LOGGING
          ↓
PRELIMINARY ANALYSIS
          ↓
IMPACT ANALYSIS
          ↓
CHANGE APPROVAL
          ↓
PLANNING
          ↓
CODE/DOCUMENT MODIFICATION
          ↓
TESTING
          ↓
CONFIGURATION CONTROL
          ↓
RELEASE
          ↓
MONITORING
          ↓
DOCUMENTATION
```

14.1 Request Identification

A maintenance request may come from:

- User
- Customer
- Tester
- Administrator
- Security team
- Management
- Monitoring system

14.2 Impact Analysis

Impact analysis determines what parts of the system may be affected by a proposed change.

Questions include:

- Which files will change?
- Which modules depend on them?
- Will the database be affected?
- Will interfaces change?
- Will security be affected?
- What tests are required?
- What documentation must change?
- Could the change break another function?

Exam Definition

Impact analysis is the systematic examination of the possible consequences of a proposed software change.

14.3 Modification

Approved changes are implemented.

14.4 Testing

The changed system is tested to determine whether:

- The requested change works.
- Existing functionality still works.
- No unacceptable regression has been introduced.

14.5 Release

The approved version is packaged and deployed.

14.6 Documentation

The change and resulting configuration are recorded.

CHAPTER 15: KEY SOFTWARE MAINTENANCE ISSUES

15.1 Technical Issues

Limited Program Understanding

Maintainers may inherit code written by developers who are no longer available.

Poor documentation increases the difficulty.

Testing Difficulty

Changing one component may affect other components.

Impact Analysis

Dependencies must be understood before changes are made.

Maintainability

Poorly structured software is expensive and difficult to maintain.

15.2 Management Issues

Maintenance managers must deal with:

- Staffing
- Budget
- Scheduling
- Customer expectations
- Prioritization

- Vendor management
- Technical debt
- Documentation
- Risk
- Change requests

15.3 Technical Debt

Technical debt refers to the future cost created when software teams choose quick or inadequate technical solutions instead of more sustainable solutions.

Example:

A developer writes poorly structured code to meet an urgent deadline.

It works today.

Later, another developer needs three days to understand and safely modify it.

That additional effort is part of the technical debt.

Preventing Technical Debt

- Good design
- Code review
- Testing
- Documentation
- Refactoring
- Appropriate architecture
- Automated quality checks

15.4 Software Maintainability

Maintainability is the degree to which software can be effectively and efficiently modified.

Maintainable software tends to have:

- Clear structure
- Good documentation
- Modular design

- Meaningful names
- Automated tests
- Low complexity
- Controlled dependencies
- Consistent coding standards

CHAPTER 16: SOFTWARE CONFIGURATION MANAGEMENT

16.1 Meaning of SCM

Software Configuration Management (SCM) is a disciplined process for identifying, controlling, tracking, documenting and auditing changes to software products and their associated configuration items throughout the software life cycle.

Simple Definition

SCM is the systematic control of changes to software and its configuration.

Why SCM?

Without SCM:

```
CHANGE → CONFUSION → CONFLICT → UNKNOWN VERSION → FAILURE
```

With SCM:

```
CHANGE → IDENTIFY → REVIEW → APPROVE → IMPLEMENT → TEST → RECORD → RELEASE
```

16.2 Configuration Item

A **Configuration Item (CI)** is a software-related item placed under configuration management control.

Examples:

- Source code
- Executable files
- Database schema

- Requirements documents
- Design documents
- Test scripts
- Configuration files
- Build scripts
- Deployment scripts
- User manuals
- APIs
- Libraries
- Infrastructure configuration

Exam Point

Not every file in a project must automatically be a CI.

A CI is an item selected for formal configuration control.

16.3 Software Configuration

A software configuration is the collection of specific versions of software components and associated items that constitute a particular state of the software product.

Example:

```
SCHOOL PORTAL v2.0
│
├── Frontend v2.0
├── Backend v2.0
├── Database Schema v5
├── API v3
├── Config File v7
├── Test Suite v2
└── Documentation v4
```

16.4 Configuration Management Baseline

A baseline is a formally identified and controlled version that serves as a reference point for future development or maintenance.

Examples:

- Requirements baseline
- Design baseline

- Development baseline
- Test baseline
- Release baseline

Baseline Principle

```
BASELINE = STABLE REFERENCE + CONTROLLED CHANGE
```

CHAPTER 17: OBJECTIVES OF SCM

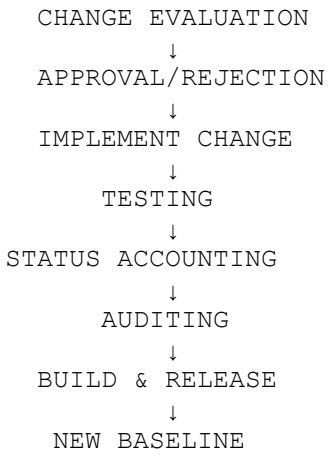
The major objectives are:

1. Control software changes.
2. Identify configuration items.
3. Maintain version integrity.
4. Prevent unauthorized changes.
5. Provide traceability.
6. Support reproducible builds.
7. Facilitate teamwork.
8. Support auditing.
9. Improve software quality.
10. Support reliable releases.
11. Enable rollback.
12. Preserve configuration history.
13. Reduce configuration errors.
14. Improve maintenance.

CHAPTER 18: SCM PROCESS

A simplified SCM process is:

```
        SOFTWARE PROJECT
               ↓
     CONFIGURATION PLANNING
               ↓
   CONFIGURATION IDENTIFICATION
               ↓
         BASELINE
               ↓
      CHANGE REQUEST
               ↓
      IMPACT ANALYSIS
               ↓
```



The Six Big SCM Activities

I-C-C-S-A-R

1. **I**dentification
2. **C**hange control
3. **C**ontrol/status accounting
4. **S**oftware auditing
5. **A**utomation/tools
6. **R**elease management

A more detailed academic structure is:

1. SCM management/planning
2. Configuration identification
3. Configuration change control
4. Configuration status accounting
5. Configuration auditing
6. Software release/build management
7. SCM tools

CHAPTER 19: SCM PLANNING

An SCM plan defines how configuration management will be performed on a project.

It may specify:

- Configuration items
- Roles and responsibilities
- Naming conventions

- Versioning scheme
- Baselines
- Change procedures
- Approval authority
- Audit procedures
- Repository structure
- Tools
- Release procedures
- Reporting
- Security
- Backup and recovery

SCM Manager Responsibilities

An SCM manager may coordinate:

- SCM planning
- Repository management
- Version control
- Change tracking
- Baselines
- Audits
- Releases
- Configuration status
- Team compliance
- SCM tools
- Documentation

CHAPTER 20: CONFIGURATION IDENTIFICATION

Configuration identification determines **what must be controlled and how it will be identified.**

Steps

1. Identify configuration items.
2. Define naming conventions.
3. Assign unique identifiers.
4. Define attributes.
5. Establish relationships.
6. Define baselines.

7. Record configuration information.

Example

```
CI-ID: WEB-LOGIN-001
Name: Login Module
Version: 2.4
Status: Approved
Owner: Development Team
Baseline: Release-2
```

CHAPTER 21: VERSION CONTROL

Version control is the management of different versions of files and software components.

Example

```
App v1.0 → v1.1 → v1.2 → v2.0
```

It allows teams to:

- Record changes
- Compare versions
- Restore earlier versions
- Work collaboratively
- Create branches
- Merge changes
- Identify contributors

21.1 Version Numbering

A common convention is:

```
MAJOR.MINOR.PATCH
```

Example:

2.4.1

- 2 = major version
- 4 = minor version
- 1 = patch

A major version may represent significant incompatible changes.

A minor version may introduce backward-compatible functionality.

A patch commonly represents bug/security fixes.

CHAPTER 22: SOFTWARE REPOSITORIES AND LIBRARIES

A repository is a controlled location where project files and their history are stored.

Examples:

- Git repositories
- Centralized version-control repositories
- Artifact repositories

A software library in SCM is a controlled storage area for software components, versions, builds or related artifacts.

Repository Functions

- Store source code
- Preserve history
- Support collaboration
- Enable version recovery
- Support branching/merging
- Store release artifacts where appropriate

CHAPTER 23: CHANGE MANAGEMENT

Change management is the controlled process of requesting, evaluating, approving, implementing, testing and recording changes.

Change Control Workflow

```
CHANGE REQUEST
      ↓
REGISTER REQUEST
      ↓
```

```
CLASSIFY/PRIORITIZE
      ↓
IMPACT ANALYSIS
      ↓
COST & RISK ANALYSIS
      ↓
APPROVAL AUTHORITY
      ↓
APPROVE / REJECT
      ↓
IMPLEMENT
      ↓
TEST
      ↓
REVIEW
      ↓
UPDATE CONFIGURATION
      ↓
RELEASE
```

23.1 Change Request

A change request should normally contain:

- Request ID
- Requester
- Date
- Description
- Reason
- Affected components
- Priority
- Impact
- Risk
- Approval status
- Implementation information
- Testing result
- Closure status

23.2 Configuration Control Board

A **Configuration Control Board (CCB)** is a group or authority responsible for reviewing and deciding on significant configuration changes.

The CCB may consider:

- Business value
- Technical impact
- Cost
- Schedule
- Risk
- Security
- Quality
- Dependencies

Important

Not every organization needs the same formal CCB structure. Smaller projects may use a simpler approval process.

CHAPTER 24: CONFIGURATION STATUS ACCOUNTING

Configuration Status Accounting (CSA) records and reports information about configuration items and their current states.

It answers:

- What version exists?
- Where is it?
- Who changed it?
- When?
- What is its status?
- Has it been approved?
- Which baseline contains it?
- Which release uses it?

Example

```
CI: Payment Module
Version: 3.2
Status: Tested
Baseline: Release 4
Change Request: CR-104
Approved: YES
Released: YES
```

CHAPTER 25: CONFIGURATION AUDITING

Configuration auditing verifies that software and documentation conform to approved configuration requirements.

Two classical audit concepts are:

25.1 Functional Configuration Audit (FCA)

Checks whether the software performs the required functions and satisfies specified requirements.

Question:

Does it do what it is supposed to do?

25.2 Physical Configuration Audit (PCA)

Checks whether the actual configuration items and documentation correspond to the approved configuration.

Question:

Is the product actually the configuration we say it is?

MEMORY

FCA = FUNCTION

PCA = PHYSICAL/PRODUCT CONTENT

CHAPTER 26: SOFTWARE BUILD MANAGEMENT

A build is the process of transforming source components and dependencies into a usable software product or artifact.

```
SOURCE CODE
```

```
    +
DEPENDENCIES
    +
CONFIGURATION
    ↓
BUILD PROCESS
    ↓
COMPILE
    ↓
LINK/PACKAGE
    ↓
TEST
    ↓
ARTIFACT
```

Build management ensures that software can be produced consistently.

Build Problems

- Missing dependency
- Wrong version
- Environment mismatch
- Broken configuration
- Compilation error
- Inconsistent tools

SCM helps by controlling versions and configurations used in builds.

CHAPTER 27: RELEASE MANAGEMENT

Release management controls the packaging, approval and delivery of a particular software version to users or another environment.

Typical flow:

```
DEVELOP → BUILD → TEST → APPROVE → PACKAGE → RELEASE → DEPLOY → MONITOR
```

A release should be identifiable.

Example:

```
SchoolPortal Release 2.5.0
```

It should be possible to determine:

- What changed?

- Which code was included?
- Which dependencies were used?
- Which tests passed?
- Who approved it?
- When was it released?

CHAPTER 28: SCM TOOLS

Common SCM and development tools include:

- Git
- GitHub
- GitLab
- Bitbucket
- Subversion (SVN)
- Mercurial

Associated tools may include:

- Jenkins
- GitHub Actions
- GitLab CI/CD
- Azure DevOps
- Jira
- SonarQube
- Docker

Important Distinction

Git = version control system.

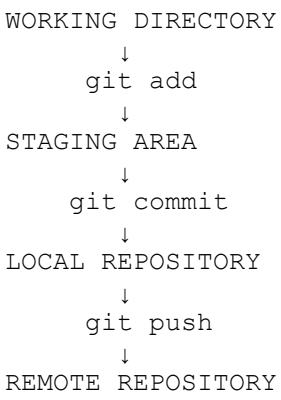
GitHub = platform for hosting Git repositories and collaboration.

They are related but not identical.

CHAPTER 29: GIT FUNDAMENTALS

Git is a distributed version-control system.

Basic workflow:



Common commands:

`git init` = initialize repository

`git status` = inspect current state

`git add` = stage changes

`git commit` = record changes

`git log` = view history

`git branch` = manage branches

`git merge` = combine branches

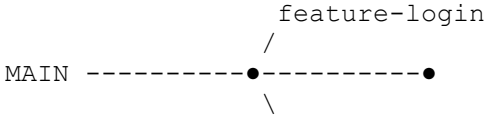
`git pull` = retrieve and integrate remote changes

`git push` = upload commits

CHAPTER 30: BRANCHING AND MERGING

A branch is an independent line of development.

Example:



```
bug-fix
```

Branches allow developers to work on features or fixes without immediately changing the main line.

Merge

Merging combines changes from one branch into another.

Merge Conflict

A conflict may occur when different changes affect the same part of a file in incompatible ways.

Conflict Resolution

1. Identify conflict.
2. Inspect competing changes.
3. Decide correct content.
4. Edit file.
5. Test.
6. Stage resolved file.
7. Commit merge.

CHAPTER 31: SCM AND SOFTWARE DEVELOPMENT

SCM supports development by:

1. Controlling source code.
2. Supporting collaboration.
3. Tracking changes.
4. Managing versions.
5. Protecting baselines.
6. Supporting releases.
7. Facilitating rollback.
8. Supporting audits.
9. Reducing accidental overwrites.
10. Improving reproducibility.

Development Without SCM

```
file_final.zip
file_final2.zip
file_final_REAL.zip
file_final_REAL2.zip
file_use_this_one.zip
```

This is not configuration management.

Development With SCM

```
Repository → History → Branches → Commits → Reviews → Controlled Releases
```

CHAPTER 32: IMPORTANCE OF SCM IN SOFTWARE DEVELOPMENT

SCM is important because it:

1. Controls Change

Prevents uncontrolled modification.

2. Provides Version History

Developers can identify previous states.

3. Supports Teamwork

Multiple developers can work systematically.

4. Enables Rollback

A stable previous version can be restored.

5. Improves Traceability

Changes can be linked to people, requests and versions.

6. Supports Quality

Controlled changes and testing reduce defects.

7. Improves Reliability

Stable configurations can be reproduced.

8. Supports Auditing

Organizations can determine what changed and why.

9. Reduces Risk

Unauthorized and poorly understood changes are minimized.

10. Supports Maintenance

Maintainers can understand the history of the software.

CHAPTER 33: SCM AND SOFTWARE QUALITY

SCM improves software quality through:

- Controlled changes
- Code reviews
- Version tracking
- Baselines
- Testing
- Auditing
- Reproducible builds
- Documentation
- Traceability
- Release control

Quality Relationship

```
CONTROLLED CHANGE → TESTED CHANGE → TRACEABLE CHANGE → HIGHER QUALITY
```

CHAPTER 34: SCM AND SOFTWARE RELIABILITY

Reliability is the ability of software to perform required functions consistently under specified conditions for a specified period.

SCM contributes to reliability by:

1. Preserving known-good versions.
2. Preventing unauthorized changes.
3. Supporting rollback.
4. Tracking defects.
5. Controlling dependencies.
6. Supporting repeatable builds.
7. Supporting testing.
8. Maintaining configuration consistency.
9. Enabling controlled releases.
10. Improving incident investigation.

ASSIGNMENT TOPIC 1:

HOW DOES SCM SUPPORT SOFTWARE MAINTENANCE?

Exam-Ready Answer Structure

SCM supports software maintenance by providing systematic control over changes made to existing software.

Major points:

1. **Version control:** Maintains different versions.
2. **Change control:** Ensures changes are reviewed and approved.
3. **Configuration identification:** Identifies components being maintained.
4. **Baseline management:** Provides stable reference points.
5. **Traceability:** Records who made changes, when and why.
6. **Impact analysis:** Helps identify components affected by changes.
7. **Rollback:** Allows restoration of stable versions.
8. **Auditing:** Verifies compliance with approved configurations.
9. **Release management:** Controls delivery of maintenance releases.
10. **Documentation:** Preserves information needed by future maintainers.
11. **Quality assurance:** Supports testing and controlled modification.
12. **Team collaboration:** Prevents developers from unknowingly overwriting one another's work.

Strong Conclusion

Therefore, SCM makes software maintenance more controlled, traceable, predictable and reliable by ensuring that modifications are properly identified, evaluated, implemented, tested, documented and released.

CHAPTER 35: CHALLENGES ASSOCIATED WITH SCM

ASSIGNMENT TOPIC 2

1. Large Number of Changes

Frequent modifications can become difficult to track.

2. Merge Conflicts

Different developers may modify the same components.

3. Configuration Complexity

Large systems may contain thousands of interconnected components.

4. Poor Documentation

Missing information makes maintenance difficult.

5. Human Error

Developers may commit incorrect files or bypass procedures.

6. Resistance to Process

Some developers may consider SCM procedures unnecessary bureaucracy.

7. Tool Complexity

SCM tools may require training and technical knowledge.

8. Poor Version Control Practices

Incorrect branching, merging or tagging can cause confusion.

9. Dependency Problems

A change to one dependency can break other components.

10. Security Risks

Unauthorized access to repositories can expose source code or credentials.

11. Cost

Tools, infrastructure and training require resources.

12. Distributed Teams

Different locations and time zones can complicate coordination.

13. Legacy Systems

Old software may lack proper version history or documentation.

14. Rapid Development

Agile/DevOps environments may generate changes extremely quickly.

15. Configuration Drift

Environments may gradually become different from their documented or approved configurations.

CHAPTER 36: PROBLEMS FACED BY SCM MANAGERS

During Software Installation

SCM managers may face:

- Wrong software versions
- Missing dependencies
- Incompatible operating systems

- Incorrect configuration
- Unauthorized installations
- Licensing problems
- Driver conflicts
- Security restrictions
- Environment differences
- Poor installation documentation

During Software Development

Problems include:

- Uncontrolled code changes
- Merge conflicts
- Branching problems
- Poor commit practices
- Inconsistent development environments
- Missing dependencies
- Unauthorized changes
- Inadequate testing
- Poor documentation
- Configuration drift

During Software Management

Problems include:

- Release failures
- Version confusion
- Legacy software
- Audit failures
- Repository security
- Poor backup
- Technical debt
- Incomplete traceability
- Uncontrolled emergency changes
- Vendor dependency

CHAPTER 37: CONFIGURATION DRIFT

Configuration drift occurs when a system gradually differs from its approved or documented configuration.

Example:

Production server:

```
Approved Java Version = X
```

Actual server:

```
Java Version = Y
```

Someone manually updated the server without updating the approved configuration.

Problems

- Unpredictable behavior
- Difficult troubleshooting
- Reproducibility failure
- Security risks
- Deployment inconsistency

Prevention

- Automated deployment
- Configuration management
- Infrastructure as code
- Documentation
- Auditing
- Version control

CHAPTER 38: CI/CD AND MODERN SCM

Modern software development increasingly connects SCM with continuous integration and delivery.

Continuous Integration (CI)

Developers frequently integrate code into a shared repository and automated processes build and test the software.

Continuous Delivery

Software is maintained in a releasable state and can be delivered through a controlled process.

Continuous Deployment

Approved changes may be automatically deployed to production.

Simplified Pipeline

```
CODE
 ↓
COMMIT
 ↓
PUSH
 ↓
BUILD
 ↓
AUTOMATED TESTS
 ↓
SECURITY/QUALITY CHECKS
 ↓
PACKAGE
 ↓
APPROVAL
 ↓
DEPLOY
 ↓
MONITOR
```

SCM is foundational because the pipeline needs to know exactly **which version of the code** is being built and deployed.

CHAPTER 39: SOFTWARE REENGINEERING AND REVERSE ENGINEERING

Reverse Engineering

Reverse engineering analyzes an existing system to understand its structure, behavior or design.

```
EXISTING SYSTEM → ANALYSIS → UNDERSTANDING
```

It does not necessarily change the software.

Reengineering

Software reengineering involves examining and modifying an existing system to improve its quality, maintainability or usefulness.

```
OLD SYSTEM → ANALYSIS → TRANSFORMATION → IMPROVED SYSTEM
```

Examples

- Refactoring old code
- Migrating an obsolete database
- Modernizing an old application
- Replacing outdated interfaces
- Restructuring architecture

CHAPTER 40: SOFTWARE RETIREMENT AND MIGRATION

Software Retirement

Software retirement is the controlled removal of a software system from active use.

Activities may include:

1. Identify replacement system.
2. Plan transition.
3. Preserve required records.
4. Migrate data.
5. Notify users.
6. Disable old services.
7. Securely archive or dispose of components.
8. Update documentation.

Software Migration

Migration involves moving software, data or services from one environment to another.

Examples:

- Local server → cloud
- Old database → new database
- Legacy OS → modern OS
- Old application → new platform

Migration Risks

- Data loss
- Compatibility problems
- Downtime
- Security issues
- Incomplete migration
- User resistance
- Unexpected dependencies

CHAPTER 41: SCM SECURITY

Modern SCM must protect software repositories and configuration information.

Security practices include:

- Authentication
- Authorization
- Role-based access
- Multi-factor authentication
- Secret management
- Audit logs
- Repository backups
- Branch protection
- Code review
- Dependency scanning
- Secure CI/CD pipelines

Important Rule

Never store passwords, API keys or private credentials directly in source-code repositories.

CHAPTER 42: SCM MEASUREMENT

SCM should be measurable.

Useful measurements include:

- Number of change requests
- Change approval time

- Number of rejected changes
- Number of configuration errors
- Build failure rate
- Release frequency
- Rollback frequency
- Number of defects after release
- Audit findings
- Mean time to restore service
- Number of unauthorized changes

Why Measure?

Measurement helps management determine whether SCM processes are effective.

CHAPTER 43: SOFTWARE QUALITY, RELIABILITY AND MAINTAINABILITY

These concepts are related but different.

Quality

Overall degree to which software satisfies stated and implied requirements.

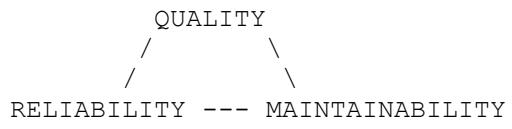
Reliability

Ability of software to perform required functions consistently under specified conditions.

Maintainability

Ease and efficiency with which software can be modified.

Memory Triangle



SCM supports all three by controlling changes and preserving configuration integrity.

CHAPTER 44: SCM PRINCIPLES

Principle 1: Every Important Change Must Be Traceable

A change should be linked to a reason, request or authorized activity.

Principle 2: Control Before Change

Changes should not be allowed to modify controlled baselines arbitrarily.

Principle 3: Know What You Have

Configuration items must be identified.

Principle 4: Know Which Version You Have

Versions must be identifiable.

Principle 5: Test Before Release

Changes should be validated before delivery.

Principle 6: Maintain Historical Records

Previous configurations must remain traceable.

Principle 7: Automate Where Practical

Automation reduces human error.

Principle 8: Protect the Repository

Configuration data and source code require security.

Principle 9: Keep Environments Consistent

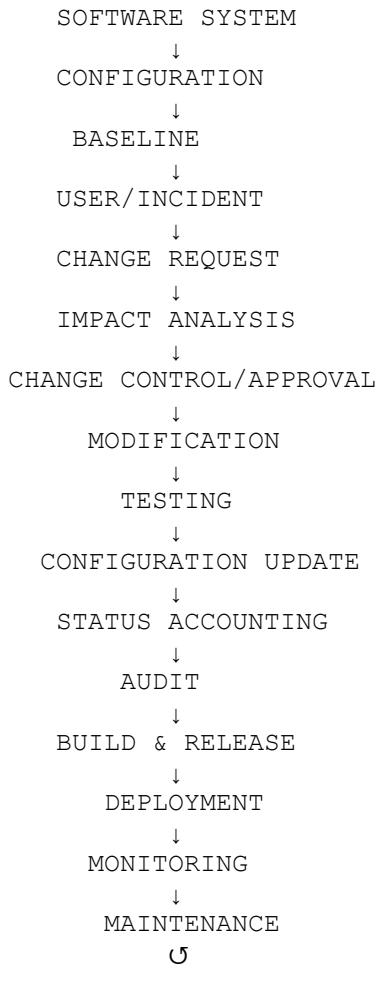
Development, testing and production differences should be controlled.

Principle 10: Document Important Decisions

Future maintainers need to understand why changes occurred.

CHAPTER 45: INTEGRATED SCM + MAINTENANCE MODEL

This is one of the most important diagrams to memorize.



This diagram connects almost the entire course.

CHAPTER 46: IMPORTANT DIFFERENCES FOR EXAMS

SCM vs Software Maintenance

SCM: Controls and tracks software configurations and changes.

Maintenance: Modifies existing software to correct, adapt, improve or prevent problems.

Version Control vs SCM

Version control: Manages versions and changes of files/software.

SCM: Broader discipline that includes identification, change control, status accounting, auditing, release management and more.

Therefore:

```
VERSION CONTROL ⊂ SCM
```

Corrective vs Preventive Maintenance

Corrective: Fix current problems.

Preventive: Reduce future problems.

Adaptive vs Perfective Maintenance

Adaptive: Respond to environmental change.

Perfective: Improve functionality/performance/usability.

FCA vs PCA

FCA: Does the software satisfy required functions?

PCA: Does the delivered configuration correspond to the approved physical/configuration content?

Baseline vs Version

Version: A particular state of a component/product.

Baseline: A formally approved and controlled reference configuration.

Git vs GitHub

Git: Version-control system.

GitHub: Hosting/collaboration platform built around Git repositories.

Compiler vs Interpreter

Compiler: Translates source program into another form before execution in the traditional compilation model.

Interpreter: Processes/executes instructions during execution.

CHAPTER 47: EXAMINATION COMMAND WORDS

"Define"

Give precise meaning.

Example:

Define SCM.

Answer with a direct definition first.

"Explain"

Give meaning + details + example.

"Discuss"

Give multiple dimensions, explanations, advantages/problems and conclusion.

"Differentiate"

Show differences clearly.

"List"

Give points without unnecessary essay.

"Describe"

Explain characteristics/process in an organized manner.

"Evaluate"

Give strengths, weaknesses and judgment.

CHAPTER 48: MASTER EXAM QUESTIONS AND ANSWERS

100 CSD 421 QUESTIONS + ANSWERS

SECTION A: DEFINITIONS

1. What is software?

Software is a collection of programs, instructions, data and associated documentation that directs a computer to perform tasks.

2. What is software maintenance?

It is the modification and support of existing software after delivery to correct, adapt, improve or prevent problems.

3. What is SCM?

Software Configuration Management is the systematic identification, control, tracking, documentation and auditing of software configurations and changes.

4. What is a configuration item?

A configuration item is a software-related item placed under formal configuration control.

5. What is a baseline?

A baseline is an approved and controlled configuration used as a reference point for future changes.

6. What is version control?

Version control is the systematic management and recording of different versions of software components.

7. What is change management?

It is the controlled process of requesting, evaluating, approving, implementing, testing and recording changes.

8. What is configuration status accounting?

It is the recording and reporting of information about configuration items, versions, baselines and their statuses.

9. What is configuration auditing?

It is the examination of software configurations to verify that they conform to approved requirements and records.

10. What is a software repository?

A repository is a controlled storage location for software components, versions and related information.

11. What is a device driver?

Software that enables an operating system to communicate with and control hardware.

12. What is an operating system?

System software that manages computer resources and provides services for applications and users.

13. What is utility software?

Software designed to maintain, protect, diagnose, manage or optimize a computer system.

14. What is application software?

Software designed to perform tasks directly useful to users.

15. What is a bootstrap loader?

A program or boot mechanism that initiates the process of loading the operating system.

16. What is a compiler?

A translator that converts source code into another form such as object or executable code before execution in the traditional compilation model.

17. What is an interpreter?

A program that processes or executes source instructions during execution.

18. What is software reliability?

The ability of software to perform required functions consistently under specified conditions.

19. What is software maintainability?

The degree to which software can be efficiently modified.

20. What is technical debt?

The future cost created by choosing quick or inadequate technical solutions instead of sustainable ones.

SECTION B: SHORT ANSWERS

21. State four types of software maintenance.

Corrective, adaptive, perfective and preventive.

22. What does corrective maintenance do?

It fixes existing defects or failures.

23. What does adaptive maintenance do?

It adapts software to changes in its operating environment.

24. What does perfective maintenance do?

It improves functionality, performance or usability.

25. What does preventive maintenance do?

It reduces the likelihood of future problems and improves maintainability.

26. State five functions of an operating system.

Process management, memory management, file management, device management and security management.

27. State five examples of application software.

Word processor, spreadsheet, browser, presentation software and database application.

28. State five utility software functions.

Security, backup, disk management, compression and diagnostics.

29. Why are device drivers important?

They enable the OS to communicate with hardware devices.

30. State five post-OS-installation activities.

Install drivers, install updates, configure security, install applications and test the system.

31. What is impact analysis?

It determines the possible effects of a proposed software change.

32. Why is impact analysis important?

It helps identify affected components, risks, dependencies, testing requirements and costs.

33. State five SCM objectives.

Change control, version management, traceability, auditing and reliable release management.

34. What is a change request?

A formal request to modify a software configuration.

35. What is a CCB?

A Configuration Control Board is an authority that reviews and decides on significant configuration changes.

36. What is a software build?

A process that transforms source code, dependencies and configuration into a usable software artifact.

37. What is release management?

The controlled process of preparing, approving and delivering a software release.

38. What is configuration drift?

The gradual difference between an actual system configuration and its approved or documented configuration.

39. What is reverse engineering?

Analyzing an existing system to understand its structure or behavior.

40. What is reengineering?

Transforming an existing software system to improve its structure, maintainability or usefulness.

SECTION C: EXPLANATORY QUESTIONS

41. Why is software maintenance necessary?

Because software environments, requirements, technology, security threats and user expectations continually change.

42. Why is SCM necessary?

To control changes, preserve configuration integrity, support collaboration, provide traceability and improve release reliability.

43. How does SCM support software maintenance?

Through version control, configuration identification, baselines, change control, impact analysis, auditing, documentation, rollback and release management.

44. How does SCM improve software quality?

It controls changes, supports testing, provides traceability, reduces configuration errors and helps maintain approved configurations.

45. How does SCM improve reliability?

It preserves known-good versions, supports controlled releases, enables rollback and reduces unauthorized or inconsistent changes.

46. Why is version control important?

It records history, supports collaboration, permits comparison and enables restoration of previous versions.

47. Why are baselines important?

They provide controlled reference configurations against which future changes can be compared.

48. Why is documentation important in SCM?

It allows developers and maintainers to understand configurations, changes, decisions and dependencies.

49. Why should changes be tested?

To verify that the modification works and has not introduced unacceptable regression or other defects.

50. Why is release management important?

It ensures that only approved, tested and correctly identified versions reach users or deployment environments.

SECTION D: PROCESS QUESTIONS

51. State the major steps in SCM.

SCM planning, configuration identification, change control, status accounting, auditing and release/build management.

52. State the basic software maintenance process.

Request, analysis, impact analysis, approval, modification, testing, release, monitoring and documentation.

53. State the OS installation process.

Check requirements, backup, prepare media, configure boot, install OS, partition storage, configure system, install drivers, update, test and document.

54. State the application installation process.

Check requirements, obtain installer, verify license/source, run installer, select options, install dependencies, complete installation, activate, update and test.

55. State the change-control process.

Request → register → analyze → evaluate → approve/reject → implement → test → review → release → record.

56. State the version-control workflow using Git.

Working directory → staging → commit → local repository → push → remote repository.

57. State the release process.

Develop → build → test → approve → package → release → deploy → monitor.

58. State the audit process.

Define expected configuration → inspect actual configuration → compare → identify discrepancies → report → correct.

59. What happens after a maintenance change?

The change is tested, reviewed, documented, placed under configuration control and released if approved.

60. Why is rollback important?

It allows the organization to restore a stable previous version when a new change causes serious problems.

SECTION E: CHALLENGES

61. State five SCM challenges.

Merge conflicts, configuration complexity, poor documentation, human error and tool complexity.

62. What causes merge conflicts?

Conflicting modifications made by different developers to overlapping portions of software.

63. How can merge conflicts be reduced?

Good branching strategy, frequent synchronization, communication, small commits and code review.

64. What is configuration drift?

Difference between actual and approved configuration.

65. How can configuration drift be controlled?

Automation, audits, documented baselines, controlled deployments and configuration management tools.

66. What causes SCM failure?

Poor planning, inadequate training, weak change control, poor documentation, tool misuse and lack of management support.

67. What problems can occur during software installation?

Missing dependencies, incompatible OS, insufficient storage, permission issues, corrupted installers and network problems.

68. What problems can occur during software development?

Version conflicts, merge conflicts, inconsistent environments, dependency conflicts and unauthorized changes.

69. What problems can occur during software management?

Release failures, configuration drift, security problems, poor documentation and legacy-system difficulties.

70. Why can legacy software be difficult to maintain?

It may have outdated technology, poor documentation, obsolete dependencies and developers who are no longer available.

SECTION F: COMPARISON QUESTIONS

71. Differentiate corrective and preventive maintenance.

Corrective fixes existing defects; preventive reduces the likelihood of future defects.

72. Differentiate adaptive and perfective maintenance.

Adaptive responds to environmental changes; perfective improves functionality, performance or usability.

73. Differentiate SCM and maintenance.

SCM controls software configurations and changes; maintenance performs modifications to preserve or improve software.

74. Differentiate version and baseline.

A version is a particular state of software; a baseline is an approved and controlled reference state.

75. Differentiate FCA and PCA.

FCA verifies required functionality; PCA verifies the actual configuration against approved configuration records.

76. Differentiate compiler and interpreter.

A compiler traditionally translates a program before execution; an interpreter processes/executes instructions during execution.

77. Differentiate system software and application software.

System software manages computer resources; application software performs user-oriented tasks.

78. Differentiate utility software and application software.

Utility software supports system management and maintenance; application software performs specific user tasks.

79. Differentiate Git and GitHub.

Git is a distributed version-control system; GitHub is a platform for hosting and collaborating on Git repositories.

80. Differentiate reverse engineering and reengineering.

Reverse engineering focuses on understanding an existing system; reengineering transforms it to improve its structure or usefulness.

SECTION G: LONG ESSAY QUESTIONS

81. Discuss the importance of SCM in software development.

SCM controls software changes, manages versions, supports collaboration, provides traceability, protects baselines, enables rollback, supports audits and improves quality and reliability.

82. Explain how SCM supports software maintenance.

SCM identifies maintenance items, controls changes, records versions, establishes baselines, supports impact analysis, enables rollback, documents changes, performs audits and controls releases.

83. Discuss the challenges associated with SCM.

Major challenges include configuration complexity, merge conflicts, human error, poor documentation, tool complexity, security, legacy systems, dependency problems, configuration drift and resistance to SCM procedures.

84. Explain how SCM improves software quality and reliability.

SCM improves quality through controlled changes, testing, traceability, audits, documentation and reproducible configurations. It improves reliability by preserving known-good versions, controlling releases, preventing unauthorized changes and enabling rollback.

85. Explain the four types of software maintenance.

Corrective fixes faults; adaptive responds to environmental changes; perfective improves functionality/performance; preventive reduces future maintenance problems.

86. Discuss software maintenance challenges.

Challenges include poor documentation, limited program understanding, testing difficulties, impact analysis, staffing, cost, technical debt, obsolete technology and changing requirements.

87. Explain configuration identification.

Configuration identification determines the items that must be controlled, assigns identifiers, defines attributes and relationships, and establishes baselines.

88. Explain configuration change control.

It involves requesting, analyzing, evaluating, approving, implementing, testing and documenting software changes.

89. Explain configuration status accounting.

It records and reports the identity, version, status, location, baseline and change history of configuration items.

90. Discuss configuration auditing.

Configuration auditing verifies that the actual software configuration conforms to approved requirements and configuration records.

SECTION H: SCENARIO QUESTIONS

91. A developer accidentally overwrites another developer's code. What SCM practice could prevent this?

Version control and controlled collaboration.

92. A new software update breaks the application. What SCM capability is useful?

Rollback to a known-good version.

93. Management asks who changed a module last week. What SCM feature provides the answer?

Version history/status accounting.

94. A new operating system causes the software to stop working. What type of maintenance is required?

Adaptive maintenance.

95. A login bug is discovered. What maintenance type applies?

Corrective maintenance.

96. Users request a faster search feature. What maintenance type applies?

Perfective maintenance.

97. Developers refactor complicated code to reduce future maintenance difficulty. What maintenance type applies?

Preventive maintenance.

98. A production server has a different configuration from the approved configuration. What problem is this?

Configuration drift.

99. A company wants to determine whether a proposed change could break other modules. What should it perform?

Impact analysis.

100. A software release must prove that the delivered configuration matches the approved configuration. What SCM activity is relevant?

Configuration auditing, particularly physical configuration verification where appropriate.

FLASHCARDS

FLASHCARD SET 1: CORE DEFINITIONS

Q: What is SCM?
A: Systematic control and tracking of software configurations and changes.

Q: What is maintenance?
A: Modification/support of existing software after delivery.

Q: What is a CI?
A: Configuration Item placed under formal configuration control.

Q: What is a baseline?
A: Approved controlled reference configuration.

Q: What is version control?
A: Management of different software versions and their histories.

Q: What is impact analysis?
A: Examination of possible consequences of a proposed change.

Q: What is configuration auditing?
A: Verification that actual configuration matches approved requirements/records.

Q: What is status accounting?
A: Recording/reporting configuration states and histories.

Q: What is release management?
A: Controlled preparation and delivery of software releases.

Q: What is technical debt?
A: Future cost resulting from inadequate or shortcut technical decisions.

FLASHCARD SET 2: MAINTENANCE

Corrective → Current bug

Adaptive → Environment change

Perfective → Improvement

Preventive → Future protection

FLASHCARD SET 3: SCM

Identification → What do we control?

Baseline → What approved state are we starting from?

Change Control → Who can change it?

Status Accounting → What is its current state?

Audit → Does reality match the approved configuration?

Release Management → What version reaches users?

FLASHCARD SET 4: INSTALLATION

OS → System platform

Driver → OS ↔ Hardware

Utility → Maintain/protect/manage

Application → User task

Bootloader → Starts OS loading

Compiler → Translate before execution

Interpreter → Process during execution

RAIDs: RAPID ACTIVE INFORMATION DRILLS

RAID 1: 30-SECOND SCM

Say aloud:

SCM identifies software components, controls changes, manages versions, establishes baselines, records status, performs audits and controls releases.

RAID 2: 20-SECOND MAINTENANCE

Corrective fixes; adaptive adapts; perfective improves; preventive prevents.

RAID 3: 20-SECOND CHANGE CONTROL

Request → Analyze → Approve → Implement → Test → Release → Record.

RAID 4: 20-SECOND SCM PROCESS

Plan → Identify → Baseline → Change → Account → Audit → Release.

RAID 5: 20-SECOND INSTALLATION

Requirements → Backup → Install → Configure → Drivers → Update → Test → Document.

RAID 6: 20-SECOND QUALITY

SCM improves quality through control, testing, traceability, auditing and reproducibility.

RAID 7: 20-SECOND RELIABILITY

Known-good versions + controlled changes + testing + rollback + consistent configuration.

MEMORY BOX: THE CSD 421 MASTER BOX

```
SCM
│
├── PLAN
│
├── IDENTIFY
│   └── Configuration Items
│
├── BASELINE
│
├── CONTROL CHANGE
│   ├── Request
│   ├── Impact Analysis
│   ├── CCB/Approval
│   └── Implementation
│
├── STATUS ACCOUNTING
│
├── AUDIT
│   ├── FCA
│   └── PCA
│
├── BUILD
│
├── RELEASE
│
└── TOOLS
    ├── Git
    ├── GitHub
    ├── GitLab
    └── CI/CD
```

MEMORY BOX: THE FOUR MAINTENANCE TYPES

```
C = CORRECTIVE = FIX
A = ADAPTIVE   = ENVIRONMENT
```

```
P = PERFECTIVE = IMPROVE
P = PREVENTIVE = FUTURE
```

MEMORY BOX: THREE ASSIGNMENT QUESTIONS

1. HOW DOES SCM SUPPORT SOFTWARE MAINTENANCE?

Remember:

V-C-B-I-T-R-A-D-R

Version control
Change control
Baselines
Impact analysis
Traceability
Rollback
Auditing
Documentation
Release management

---

2. CHALLENGES OF SCM

Remember:

C-M-H-D-T-S-L-D

Complexity
Merge conflicts
Human error
Documentation problems
Tool complexity
Security
Legacy systems
Dependency/configuration problems

---

3. HOW DOES SCM IMPROVE QUALITY AND RELIABILITY?

Remember:

C-T-T-A-R

Control
Testing
Traceability
Auditing
Rollback/reproducibility

CHAPTER QUIZ: MAINTENANCE

1. Define software maintenance.
2. Why is maintenance necessary?
3. Explain corrective maintenance.
4. Explain adaptive maintenance.
5. Explain perfective maintenance.
6. Explain preventive maintenance.
7. Differentiate corrective and preventive maintenance.
8. What is impact analysis?
9. What is technical debt?
10. State four maintenance challenges.

CHAPTER QUIZ: SCM

1. Define SCM.
2. What is a configuration item?
3. What is a baseline?
4. State six SCM activities.
5. What is configuration identification?
6. Explain change control.
7. What is a CCB?
8. Explain status accounting.
9. Explain configuration auditing.
10. Differentiate FCA and PCA.

CHAPTER QUIZ: MODERN SCM

1. What is Git?
2. What is GitHub?
3. What is a branch?
4. What is a merge conflict?
5. What is CI?
6. What is CD?
7. Why is automation useful in SCM?
8. What is configuration drift?
9. What is a reproducible build?
10. Why should secrets not be stored in source code?

FINAL 48-HOUR REVISION MAP

ROUND 1: UNDERSTANDING

Study in this order:

```
Software → OS → Installation → Application Software → Drivers → Utilities → Maintenance → SCM
```

Do not spend excessive time memorizing definitions before understanding the relationships.

ROUND 2: MEMORIZE THESE 15 THINGS

1. Definition of software
2. Definition of OS
3. OS functions
4. Software installation steps
5. Post-installation activities
6. Device driver
7. Bootstrap loader
8. Four maintenance types
9. SCM definition

10. Configuration item
11. Baseline
12. SCM process
13. Change-control process
14. Configuration auditing
15. SCM benefits/challenges

ROUND 3: MEMORIZE THE DIAGRAMS

Diagram 1: Software Environment

```
USER
 ↓
APPLICATION
 ↓
OPERATING SYSTEM
 ↓
DEVICE DRIVER
 ↓
HARDWARE
```

Diagram 2: Maintenance

```
REQUEST
 ↓
ANALYSIS
 ↓
IMPACT ANALYSIS
 ↓
APPROVAL
 ↓
MODIFICATION
 ↓
TEST
 ↓
RELEASE
 ↓
MONITOR
```

Diagram 3: SCM

```
PLAN
 ↓
IDENTIFY
 ↓
BASELINE
 ↓
CHANGE CONTROL
 ↓
```

```
STATUS ACCOUNTING
 ↓
AUDIT
 ↓
BUILD
 ↓
RELEASE
```

ROUND 4: PRACTICE THE 100 QUESTIONS

Do not merely read the answers.

Cover the answers and attempt to produce them yourself.

For essay questions use:

```
DEFINITION → EXPLANATION → POINTS → EXAMPLE → CONCLUSION
```

ROUND 5: ASSIGNMENT QUESTIONS

QUESTION 1

How does SCM support software maintenance?

Answer using:

```
Version → Change Control → Baseline → Impact Analysis → Traceability →
Rollback → Audit → Documentation → Release
```

QUESTION 2

Discuss the challenges associated with SCM.

Answer using:

```
Complexity → Conflicts → Human Error → Documentation → Tools → Security →
Dependencies → Legacy Systems → Drift → Cost
```

QUESTION 3

Explain how SCM improves software quality and reliability.

Answer using:

```
Controlled Change → Testing → Traceability → Audit → Stable Baselines → Reproducibility → Rollback
```

LAST-MINUTE EXAM CRAM SHEET

SOFTWARE

Software = programs + data + procedures + documentation.

OPERATING SYSTEM

OS = manages hardware/resources + provides environment for applications.

DEVICE DRIVER

Driver = software interface between OS and hardware.

UTILITY

Utility = maintenance/protection/management software.

BOOTSTRAP

Bootstrap = initiates operating-system loading.

MAINTENANCE

Maintenance = modify/support existing software.

CAPP

Corrective = Fix
Adaptive = Environment
Perfective = Improve
Preventive = Future

SCM

SCM = identify + control + track + audit software configurations and changes.

CI

Configuration Item = item under configuration control.

BASELINE

Baseline = approved controlled reference state.

CHANGE CONTROL

Request → Analyze → Approve → Implement → Test → Release.

STATUS ACCOUNTING

Records configuration status/history.

AUDIT

Checks actual configuration against approved configuration.

FCA

Functional Configuration Audit = does it perform required functions?

PCA

Physical Configuration Audit = does the delivered configuration match approved configuration?

VERSION CONTROL

Tracks versions and history.

GIT

Distributed version-control system.

RELEASE

Approved software version delivered to users/deployment.

QUALITY

SCM improves quality through controlled change and verification.

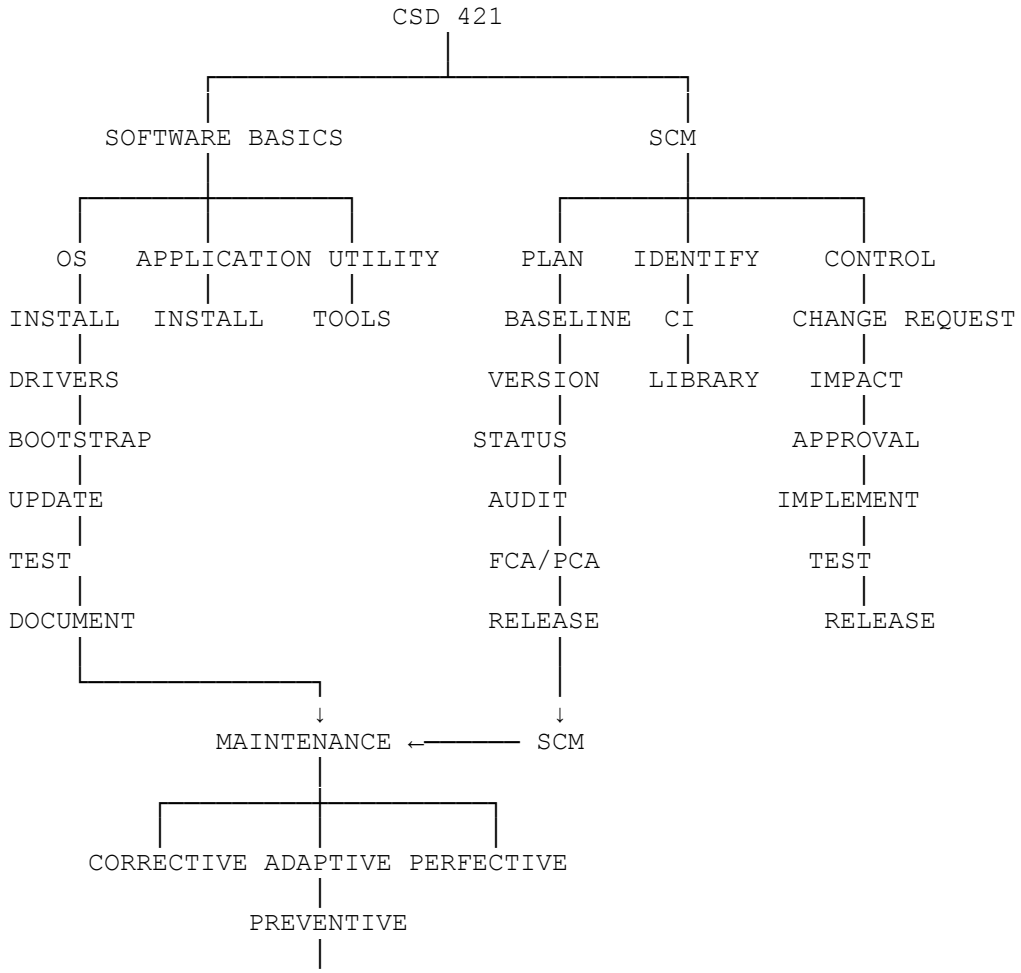
RELIABILITY

SCM improves reliability through stable configurations, testing, traceability and rollback.

MAINTAINABILITY

Good structure + documentation + modularity + testing = easier maintenance.

THE ONE-PAGE BRAIN MAP



FINAL EXAM COMMANDMENTS

1. **Never define SCM as simply "saving different versions."** That is only part of SCM.
2. **Never say maintenance means only fixing bugs.**
3. **Remember all four maintenance types.**
4. **Know the difference between version and baseline.**
5. **Know FCA vs PCA.**
6. **Know configuration item.**
7. **Know impact analysis.**
8. **Know change-control workflow.**
9. **Know how SCM supports maintenance.**
10. **Know SCM challenges.**
11. **Know how SCM improves quality and reliability.**
12. **Know installation and post-installation.**
13. **Know drivers, utilities and bootstrap loader.**
14. **Know Git fundamentals.**
15. **Always give examples in essay answers.**
16. **Use diagrams when explaining processes.**
17. **For "discuss", do not merely list points. Explain each one.**
18. **For "differentiate", present clear contrasts.**
19. **For "explain", definition + mechanism + example is powerful.**
20. **If stuck in an essay, use: Definition → Importance → Process → Advantages → Problems → Example → Conclusion.**

CSD 421 FINAL MEMORY SENTENCE

"Identify what you have, baseline what is approved, control what changes, record what happened, audit what exists, build what is needed, release what is tested, and maintain what users depend on."

That single sentence captures the heart of Software Configuration Management and Maintenance.

END OF MASTER NOTES

CSD 421: SOFTWARE CONFIGURATION MANAGEMENT AND MAINTENANCE IN COMPUTER EDUCATION

STUDY → RECALL → PRACTICE → WRITE → PASS

This educational material is created for academic study and revision purposes.

First Edition: 2026