Course: Software Configuration Management

Week 2: Core SCM Process Activities

Lecturer: Yimer Amedie (MSc.)

Addis Ababa Science and Technology University, Ethiopia

September, 2025

Contents

Figure 1: Concepts of SCM (Source: OpenAI, 2025)

Learning Outcomes

After completing this lesson, you will be able to:

- Identify and differentiate between key SCM concepts
- Explain the core SCM process activities and their relationships
- Articulate the complete SCM workflow
  - From Planning (Input), through the Core Activities (Process) ➔
  - To Build & Release (Output).

.

Introduction

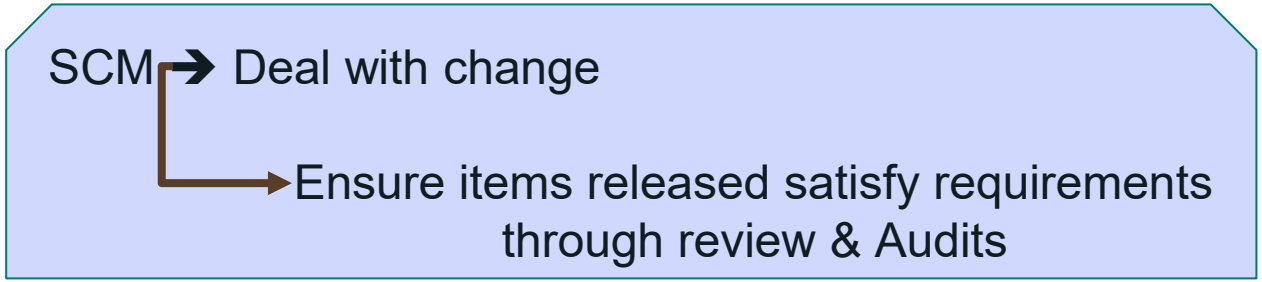
- SCM is the set of activities that are performed throughout the project life cycle (Leon, 2015)
  - from requirements analysis ➔ maintenance
- SCM is important because software is subject to **constant change**
  - software systems undergo changes when designed, when built, and even after being built.

Introduction … (2)

- Uncontrolled and unmanaged change can create
  - confusion and lead to communications breakdown problems
  - shared data problems
  - multiple maintenance problems
  - simultaneous update problems

Introduction … (3)

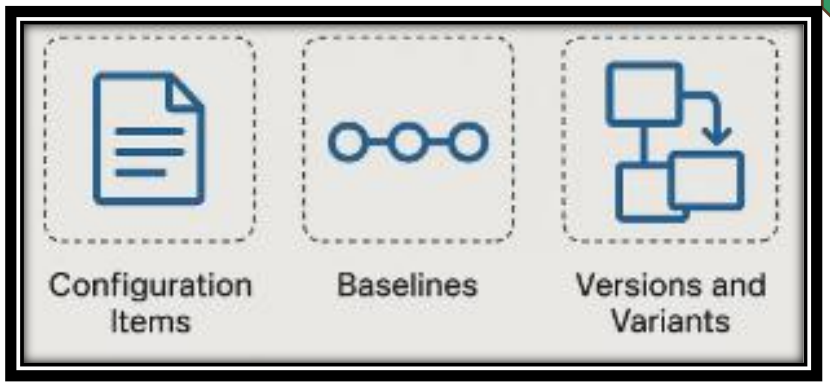
- SCM is a critical aspect of software development, as it helps to ensure that
  - Software components are properly controlled, tracked, and stored
  - Changes made to a software system are properly coordinated
  - The system is always in a known and stable state.



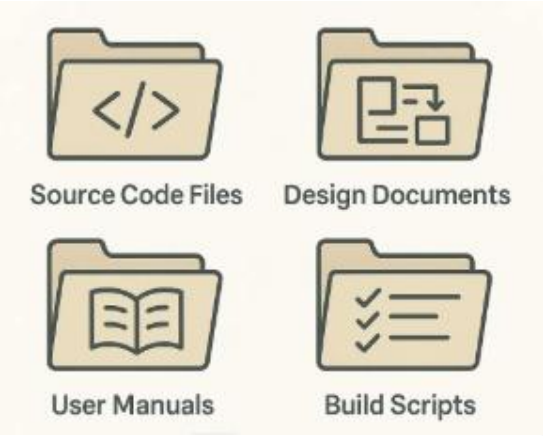
Core Concepts in SCM Process Activities

Core Concepts

The SCM process is a structured set of activities designed to manage and control the evolution of software systems.



These activities are interconnected and form the backbone of any robust software engineering practice.



Configuration Items (CIs)

The building blocks that make software control possible

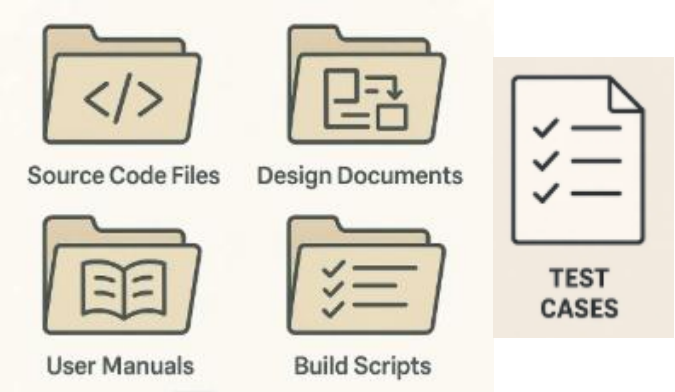
Configuration Item (CI)

What is Configuration Item?

- A configuration item is any identifiable component of a software project that needs to be managed and controlled.
- Each CI is treated as a distinct entity with its own version history.

Configuration Item (CI)

- Examples:



A functional baseline is established for each CI, which can breakdown into allocated baselines for subsystems

Baselines

The Foundation of Control

Baselines

- A milestone that marks a known, good, and stable state of the project.

Formally Approved

An agreed-upon version of software artifacts.

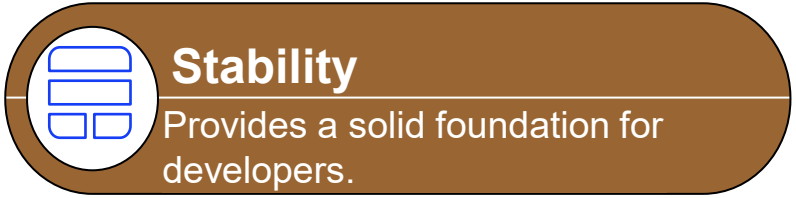
Placed under formal change control;
not to be modified arbitrarily.

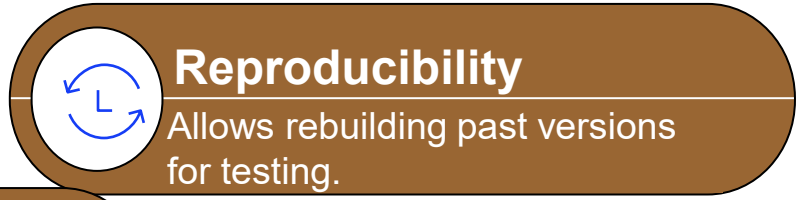
Frozen State

Reference Point

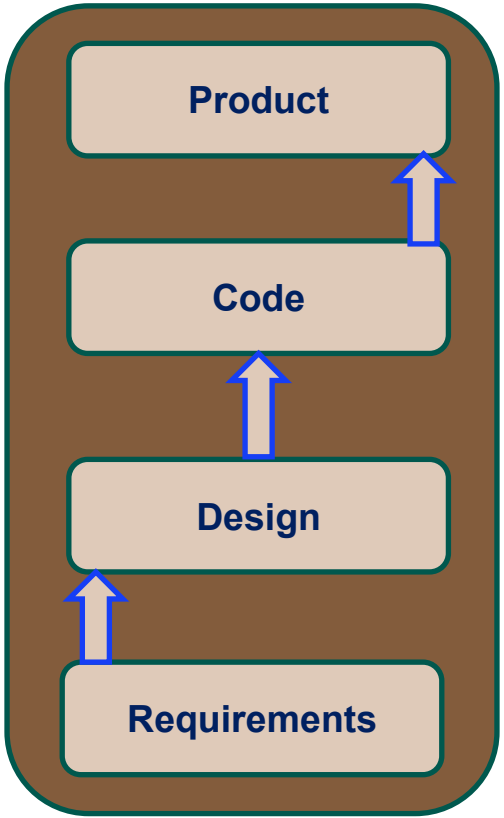
Serves as a stable foundation for all future development.

Baselines – Importance



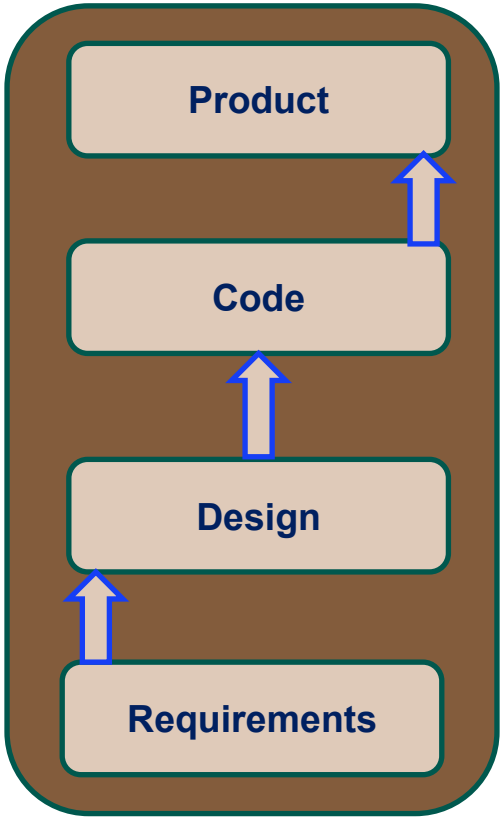


Baselines – Key Types



- Milestones in the Lifecycle
  1. **Functional Baseline:**
     - Approved system requirements (what the system will do).
  2. **Allocated/Design Baseline:**
     - Approved system architecture (how it will be done).

Baselines – Key Types … (2)



- **Milestones in the Lifecycle**
  3. **Product Baseline:**
     - The final released software + documentation (the final product).
  4. **Other common types:**
     - Development Baseline, Alpha/Beta Release Baselines.

Baselines – The Process

How it's Done?

1. **Identify & Select**
   - The correct versions of all components.
2. **Build & Test**
   - The integrated set to ensure it works.
3. **Formal Review**
   - The team and Change Control Board (CCB)

Baselines – The Process … (2)

How it's Done?

4. **Approve and Tag**
   - In the version control system
5. **Communicate**
   - The new baseline for the entire team

Baselines: Example

Imagine the team is building a new Learning Management System

- Key Modules:
  - user-auth/ (Login, Registration)
  - course-content/ (Upload, View Materials)
  - assignments/ (Create, Submit, Grade)
  - discussion-forum/ (Threads, Posts)

Goal: Establish Version 1.0 (V1.0) Product Baseline.

Baselines: Example – Create Baseline

- **V1.0 Baseline**
  - Step 1: All modules are developed and unit tested.
  - Step 2: We integrate and perform end-to-end testing.
  - Step 3: The build is successful and stable!
  - Step 4: We formally tag this state as the baseline.

```
# This command captures the exact state of the codebase
git tag -a "v1.0-baseline" -m "LMS Product Baseline for first release."
```

The code is now FROZEN. This is our official reference point.

Baselines: Example – Change Requested

After the Baseline: A Change Request

- Post-Baseline Change Request:
  - **Title:** **"Add Due Date Reminder for Assignments"**
  - **Request:** After V1.0 is released, users request an email reminder 24 hours before an assignment is due.

This is a NEW feature. It was NOT in the V1.0 baseline.

How do we handle this change without breaking our stable baseline?

Baselines: Example – Managing the Change

Branching from Baseline: The Process:

1. **Branch from the Baseline**

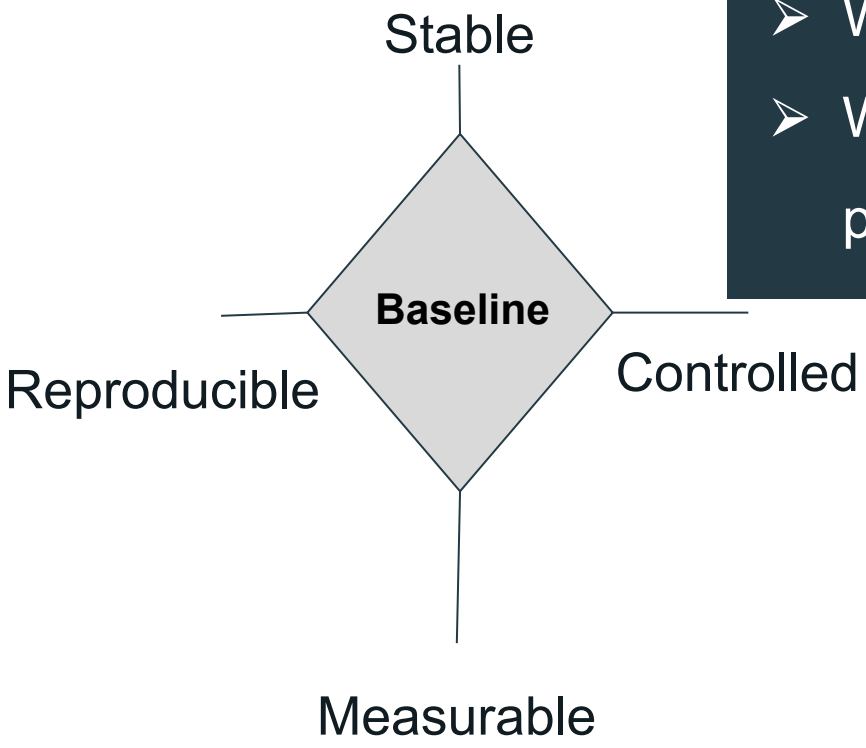
```
git checkout v1.0-baseline   # Go to the frozen baseline
git checkout -b hotfix/add-reminders  # Create a new branch from it
```

2. **Develop the new feature** (**add-reminders** module).
3. **Test rigorously** in isolation.
4. **Merge back** through a formal review process.

Baselines – Final Remark

- A Baseline is a formally approved, frozen snapshot.
- It provides Stability, Reproducibility, and Control.
- It is created at key milestones (Functional, Design, Product).
- The process involves Selection, Testing, Approval, and Tagging.
- It is enabled by Version Control tools like Git.

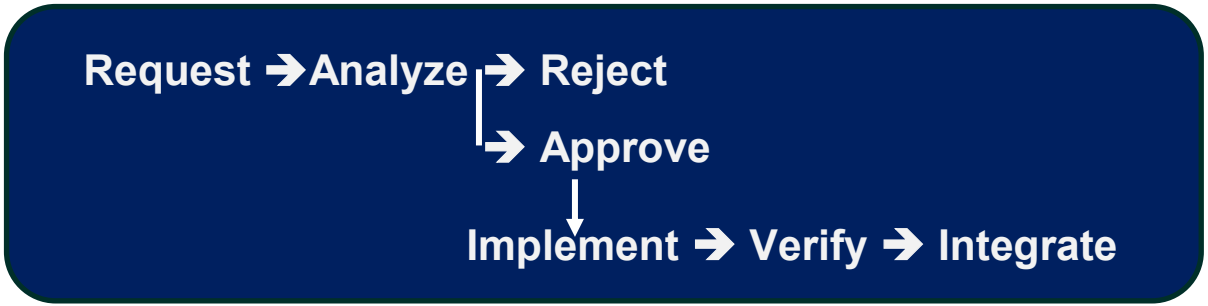
Baselines – Final Remark … (2)



- Without baselines, you have history.
- With baselines, you have a managed project.

Change Control

- The formal process for managing changes to the software's components.
- Its main job is to prevent chaos and ensure that every change is justified, reviewed, and documented.
- The process for requesting, approving, and implementing changes.



Versioning

- Versioning refers to the systematic process of assigning unique identifiers to different states or revisions of software artifacts to track changes over time.
- It ensures that developers can
  - manage multiple iterations of a software item
  - compare changes
  - retrieve previous versions
  - coordinate work among team members.

Versioning – Key points

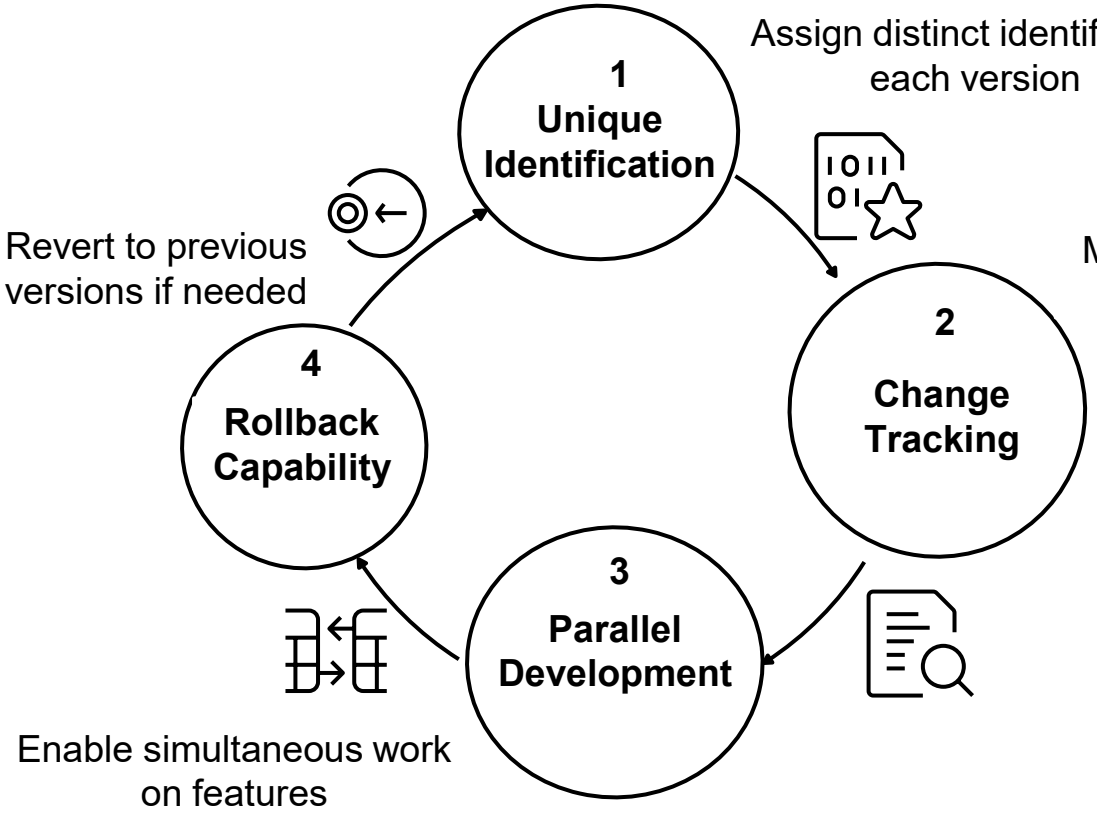
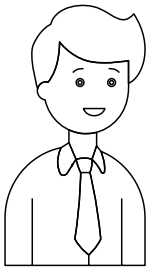


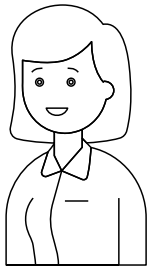
Figure 2: Versioning Cycle (Source: NapkinAI, 2025)

Variants

What are software variants?



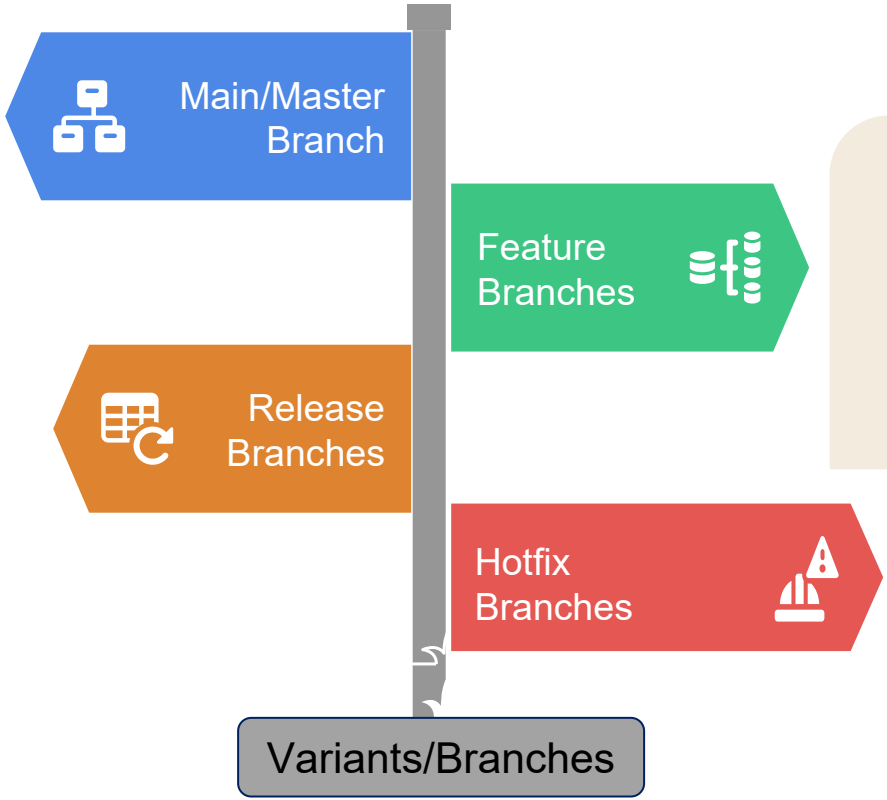
They are versions of the software developed in parallel.



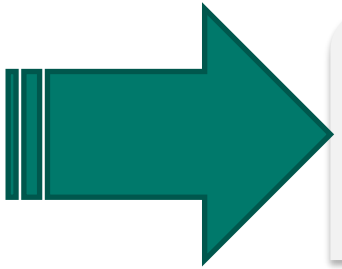
Why use them?

They allow us to work on multiple lines of development without interfering with each other.

Variants – Common Types



The power of variants comes from the ability to **merge** changes from one branch into another



The Core SCM Process Activities

The Core Four and Their Enablers

- **Configuration Planning**
  - The **Input**, foundation
  - Defines the rules for how you will perform each of the core activities.
  - It answers:
    - What tools will we use for Identification?
    - What is the process for Control?

The Core Four and Their Enablers

- The four core activities are the **process**.
  - **Configuration Identification**
    - What to control?
  - **Configuration Control**
    - How to change it
  - **Configuration Status Accounting**
    - How to track it
  - **Configuration Auditing**
    - Is it correct?

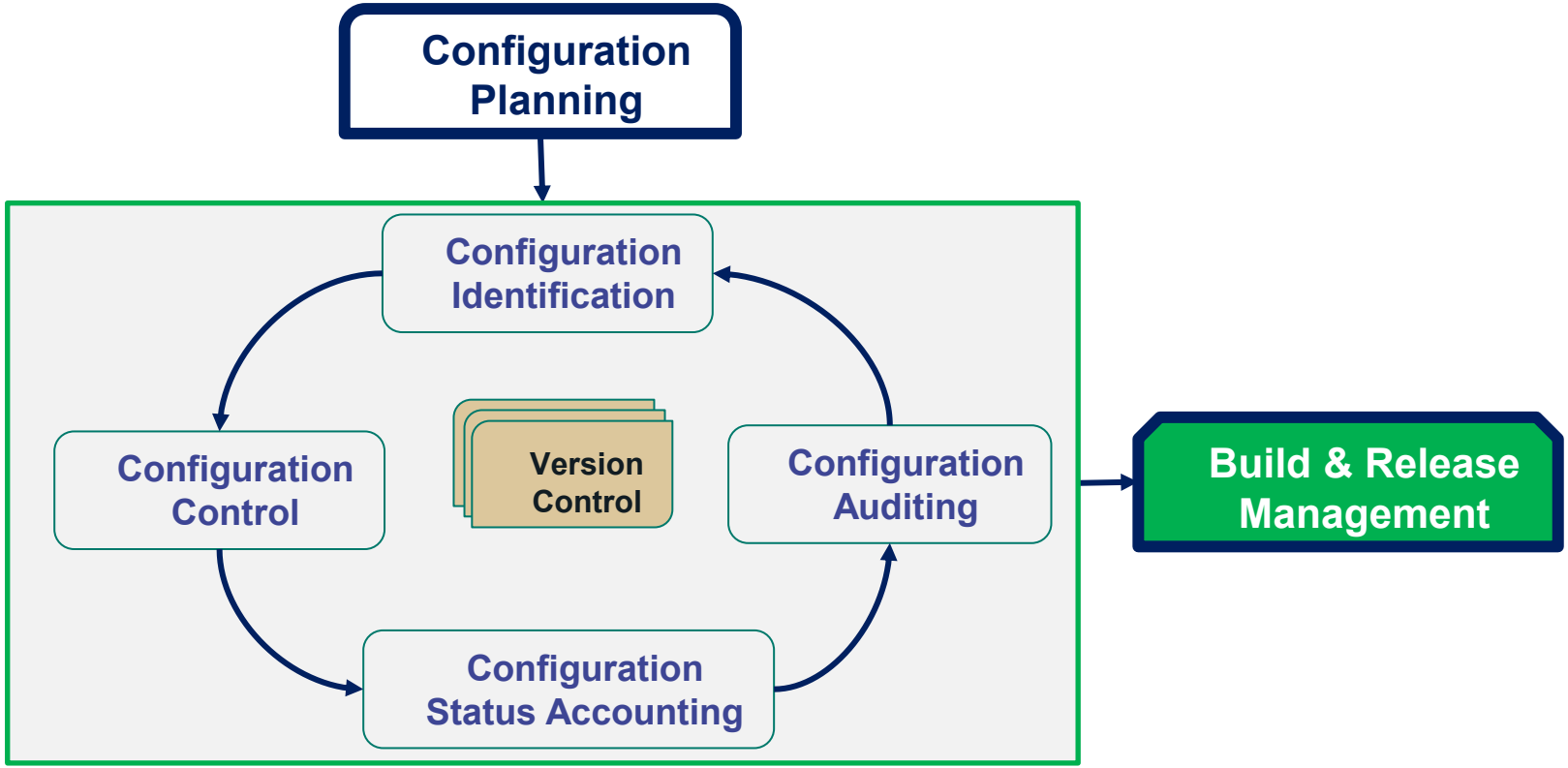
The Core Four and Their Enablers

- **Version Control**
  - The **Toolbox**
    - Enables the cycle
  - Managing different versions of files
    - Code
    - docs
    - configs

The Core Four and Their Enablers

- **Build & Release Management**
  - The **Output**
    - Creating the product
  - Managing the delivery of software to users.

The SCM Framework: A Unified View



Summary

- In SCM, the core concepts are the Building Blocks:
  - **Configuration Item (CI):** Any important artifact
    - code, docs, models.
  - **Baseline:** A frozen snapshot of CIs at a point in time;
    - a milestone.
  - **Version:** An iteration in time
    - v1.0 → v1.1 → v2.0.
  - **Variant:** A parallel fork for a different purpose
    - Mobile vs. Web app.

Summary … (2)

- SCM activities includes:
  - **Input =** ➔ The rulebook.
  - **Process =** ➔ The engine.
  - **Output =** ➔ The product.
  - **Tool =** ➔ The automation (Git).

Processes ➔ Core activities

Input, Output, Tool ➔ Enablers

Summary … (3)

- The four SCM core activities:-
  1. **Configuration Identification:** where we decide which items need control and assign them identifiers.
  2. **Configuration Control:** which manages change requests and approvals.
  3. **Configuration Status Accounting:** which records and reports the status of all items.
  4. **Configuration Auditing:** which ensures that items are correct and consistent.

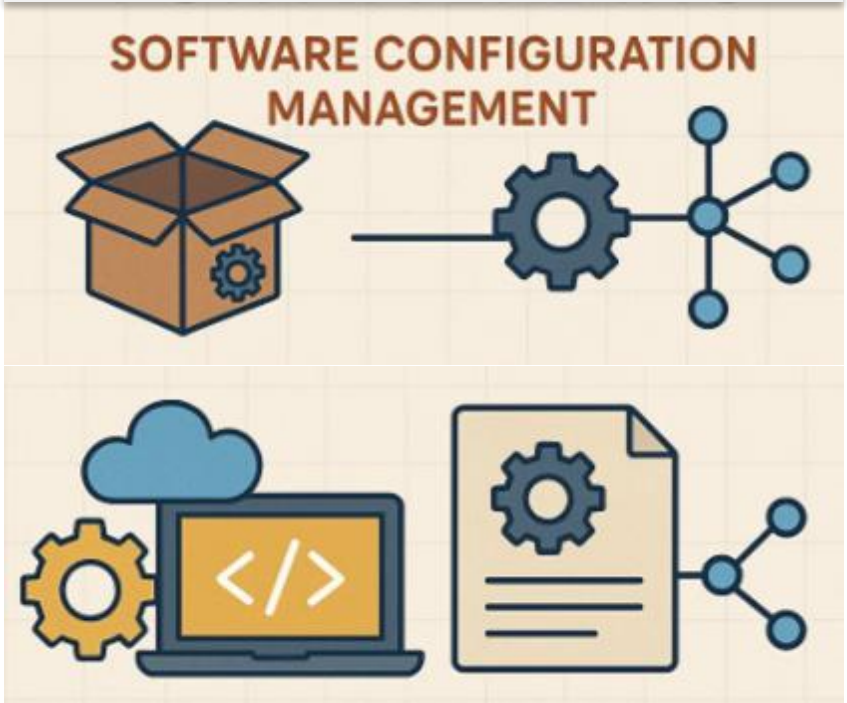
Summary … (4)

- **The enablers:-**
  - **Configuration Planning**
    - which defines how SCM will be implemented in a project.
  - **Release management**
    - Which manages the deliver of the software to users.
  - **Version Control System**
    - The technology that automates and enables the entire process.

References

1. Leon, A. (2015). Software Configuration Management Handbook (3rd ed.). Norwood: Artechhouse. Retrieved September 4, 2025.
2. OpenAI. (2025, September 4). SCM history and concepts [AI-generated image]. ChatGPT (Sora). https://chat.openai.com/
3. Napkin AI. (2025, September 19). Versioning Cycle in Software development [AI-generated image]. Retrieved from https://app.napkin.ai/

Thank You!



NEXT LECTURE

Audit and Status Reporting