Chapter 7

Software Configuration and Management

7.1 Software Configuration Items

Software Configuration Items (SCIs) are the parts or components of a software system that are tracked and managed to ensure everything works well together. These items can be anything used to develop, operate, or maintain software.

Some examples of software configuration items are:

SCIs	Description	Examples
1. **Source Code**	The actual programming code written by developers.	▪ A Python script for data analysis (data_analysis.py). ▪ A JavaScript file for a website's navigation menu (menu.js).
2. **Requirements Documents**	A list of what the software needs to do.	▪ A document specifying "The system must allow users to log in using email." ▪ A spreadsheet listing all features the app should have.
3. **Design Documents**	Blueprints showing how the software will be built.	▪ Diagrams showing how different modules in an app connect. ▪ A UI mockup of how a webpage should look.
4. **Test Cases**	Steps and conditions to test if the software works as expected.	▪ A checklist to verify if the login page accepts valid usernames. ▪ A script to test if an API returns the correct data.
5. Executable Files	The final, compiled program users run.	▪ A .exe file for a Windows application. ▪ A .apk file for an Android app.
6. **User Manuals and Documentation**	Guides and instructions for using the software.	▪ A PDF showing users how to install and use the software. ▪ Help pages explaining how to fix common errors.

7. **Configuration Files**	Files that define settings and preferences.	▪ A .json file specifying server connection settings. ▪ A .yaml file used to configure a deployment pipeline.
8. **Tools and Libraries**	Additional components that support development.	▪ A third-party library like React.js for building a web interface. ▪ A testing framework like JUnit for Java.

Software Configuration Management:

Software Configuration Management (SCM) is a process used in software development to systematically manage, organize, and track changes to all the components of a software project. It ensures that the software's components (such as code, documents, and tools) are properly controlled and up-to-date throughout the development lifecycle.

Software systems undergo continuous changes throughout their development and usage due to bug fixes, evolving requirements, hardware updates, and competitive pressures. Each modification results in a new version of the system, necessitating effective management of these versions.

Software Configuration Management (SCM) focuses on the policies, processes, and tools required to manage these changing software systems. It helps track changes and component versions, ensuring that modifications, fault corrections, and adaptations for different platforms are properly implemented. Without proper SCM, organizations risk working on the wrong version, delivering incorrect systems to customers, or losing track of source code.

CM is particularly beneficial for individual projects to maintain awareness of changes but is essential in team environments where multiple developers collaborate, often across various locations. A SCM system enables teams to access accurate information about the system under development and minimizes interference with each other's work.

The aim of configuration management is **to support the system integration process** so that all developers:

- can access the project code and documents in a controlled way,
- find out what changes have been made, and
- compile and link components to create a system.

SCM Process:

The **SCM process** ensures systematic control and tracking of changes in software development to maintain consistency and quality. It involves these key steps:

1. **Identification**: Define and track all components (e.g., code, documents, tools) as configuration items (SCIs).
2. **Version Control**: Manage and track changes to SCIs using tools like Git.
3. **Change Management**: Review, approve, and document all changes systematically.
4. **Configuration Status Accounting**: Keep records and reports of all changes and their status.
5. **Configuration Audits**: Verify that components are complete, consistent, and meet requirements.
6. **Build Management**: Assemble and test software components into working versions.
7. **Release Management**: Package and distribute software versions with proper documentation.

7.2 Configuration Management Activities

Configuration Management activities help ensure that software projects are well-organized, changes are controlled, and the final product is consistent and reliable.

The activities are:

1. Configuration Item Identification
2. Change Management
3. Version Management
4. System Building
5. Release Management

Configuration Item Identification	Identifying all components (configuration items) that need to be managed.
Version management	Keeping track of the multiple versions of system components and ensuring that changes made to components by different developers do not interfere with each other.

System Building	The process of assembling program components, data and libraries, then compiling these to create an executable system.
Change Management	Keeping track of requests for changes to the software from customers and developers, working out the costs and impact of changes, and deciding the changes should be implemented.
Release Management	Preparing software for external release and keeping track of the system versions that have been released for customer use.

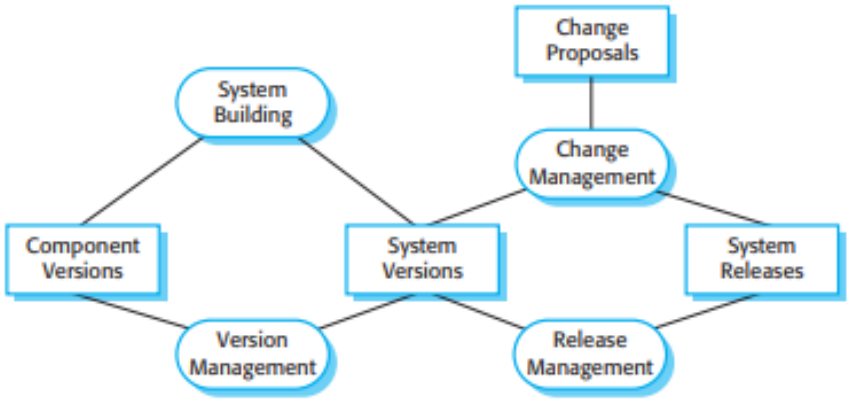

Figure: Configuration Management Activities

1. Change Management

Change Management is a key activity in **Software Configuration Management (SCM)**, aimed at controlling, tracking, and documenting changes in software projects to minimize disruption and ensure quality. It involves a systematic process to manage all change requests, whether they relate to requirements, design, code, or documentation.

Factors to be considered in change Analysis:

- The consequences of not making the change
- The benefits of the change
- The number of users affected by the change
- The costs of making the change
- The product release cycle

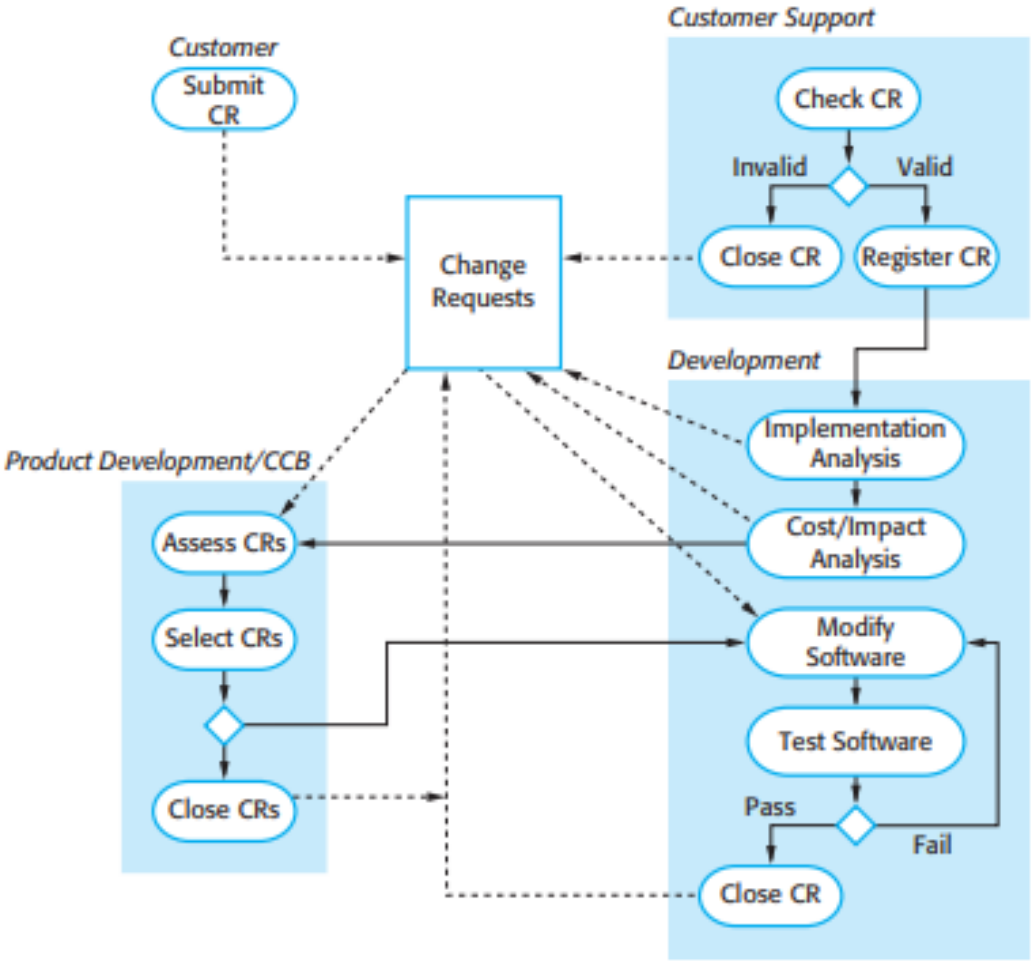

Figure: Change Management Activities

The activities in change management Process are:

Activities	Description
1. Change Request Submission	○ Changes can originate from users, developers, testers, or stakeholders. ○ Each request is documented with details like the reason for the change, the affected components, and the priority.
2. Change Request Evaluation	○ Analyze the impact of the change on: ▪ The software's functionality. ▪ The development schedule and budget. ▪ Other configuration items (e.g., code, documents). ○ Assess whether the change aligns with the project's goals.
3. Change Request Approval or Rejection	○ A Change Control Board (CCB) or similar authority reviews the request. ○ The decision is based on: ▪ Feasibility of implementation. ▪ Cost vs. benefit analysis. ▪ Priority and necessity of the change.
4. Change Implementation	○ If approved, the change is assigned to the development team. ○ The required components (e.g., code, design) are modified. ○ Version control is used to track the changes and ensure proper integration.
5. Change Verification and Testing	○ The changes are tested to ensure they meet requirements and do not introduce new issues. ○ Testing may include unit tests, integration tests, or user acceptance tests.
6. Change Documentation and Closure	○ Document the change in the change log. ○ Update related configuration items, such as design documents or user manuals. ○ Close the change request once implementation and testing are complete.

Example of Change Management:

1. A user requests adding a **"Forgot Password" feature** to an application.
2. The development team evaluates its impact:
    - Code: Add a password reset module.
    - Time: 2 weeks to develop and test.
    - Cost: Moderate, but adds significant user value.
3. The request is approved by the Change Control Board.
4. Developers implement the change, track it in the version control system, and test the new feature.
5. The change is documented, and the updated software is released.

Benefits of Effective Change Management

- Ensures **controlled and predictable changes**.
- Minimizes the risk of **errors and disruptions**.
- Enhances **project transparency** and stakeholder communication.
- Helps maintain the **quality and integrity** of the software.

2. Version Management

Version Management is a key component of SCM that focuses on controlling and tracking the multiple versions of software and its components. Version management ensures that changes made to software over time are documented, and specific versions can be retrieved when needed.

It also involves ensuring that changes made by different developers to these versions do not interfere with each other.

Key activities in Version Management are:

1. Version Identification	• Assign unique identifiers (e.g., numbers, tags) to distinguish software versions. • Example: v1.0, v2.1.3, release-2025-01.
2. Change Tracking	• Record every change made to configuration items (e.g., code, documentation). • Use tools like Git to log commit messages describing each modification.
3. Version Storage	• Maintain a repository of all software versions. • Ensure backups are available to retrieve any past version.
4. Branching and Merging	• **Branching**: Create separate lines of development for features, bug fixes, or experiments. • **Merging**: Integrate changes from branches into the main codebase. • Example: Developers create a branch for a new feature and merge it back after testing.
5. Baseline Creation	• Define stable versions of the software as "baselines. • Baselines act as reference points for further development or release. • Example: After extensive testing, version v1.5 is marked as a baseline.
6. Version Selection	• Identify which version of the software is appropriate for a specific purpose (e.g., testing, production). • Use v2.0-beta for testing while v1.8 is in production.

Importance of Version Management

1. **Prevents Overwriting**: Ensures multiple team members don't overwrite each other's work.
2. **Supports Parallel Development**: Allows simultaneous work on features, bug fixes, and experiments.
3. **Enables Rollback**: Makes it easy to revert to a previous stable version if an issue arises.
4. **Improves Transparency**: Keeps a complete history of changes, helping with audits and troubleshooting.

Example of Version Management:

1. A team is developing an app and releases v1.0 for production.
2. A bug is reported, so they create a bug-fix branch to fix it without disrupting ongoing development on the main branch.
3. After testing, the fix is merged into the main branch and released as v1.0.1.
4. Meanwhile, the team works on new features in a feature-update branch, which will later be merged and released as v2.0.

Codelines and Baselines

A **codeline** is a sequence of versions of source code files that evolve over time. It represents a branch or path of development in a version control system.

A **baseline** is a fixed point in the software's development that serves as a reference. It's a stable, tested version of the software that shouldn't change unexpectedly.

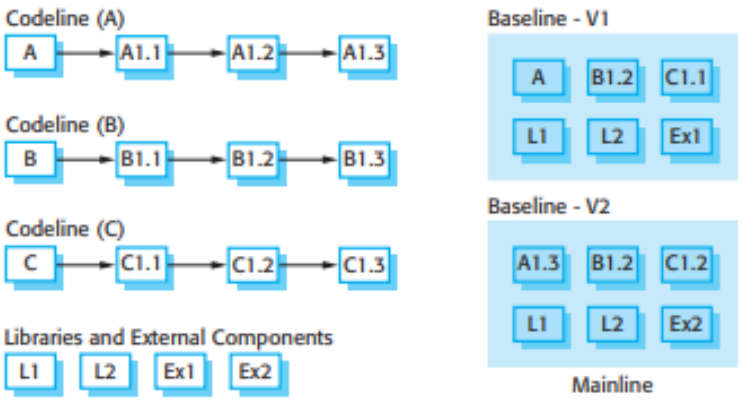

Figure: Codelines and Baselines

Centralized Version Repository:

- A **Centralized Version Repository** is a single, central location where all versions of a software project's files, including source code, documents, and other assets, are stored and managed. It serves as the authoritative source of truth for the project's codebase.
- To support independent development without interference, all version control systems use the concept of a project repository and a private workspace.
- The project repository maintains the "master" version of all components, which is used to create baselines for system building.
- When modifying components, developers copy (check-out) these from the repository into their workspace and work on these copies.
- When they have completed their changes, the changed components are returned (checked-in) to the repository.

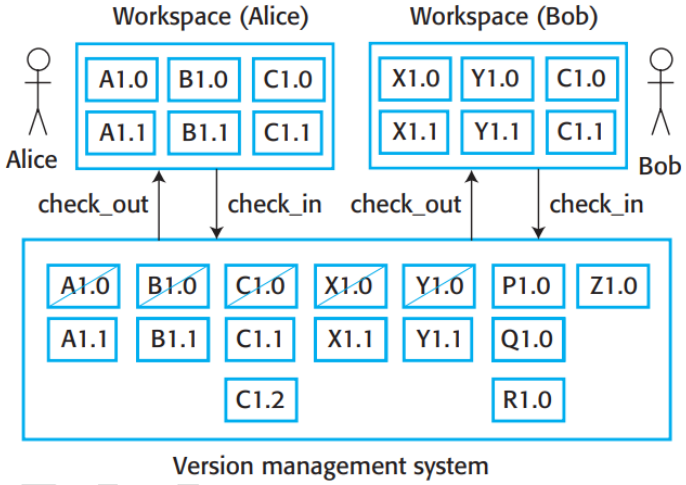

Figure: Check-in and Check-out from a Centralized Version Repository

Branching and Merging:

- Rather than a linear sequence of versions that reflect changes to the component over time, there may be several independent sequences.
- Different developers work independently on different versions of the source code and so change it in different ways.
- At some stage, it may be necessary to merge codeline branches to create a new version of a component that includes all changes that have been made.

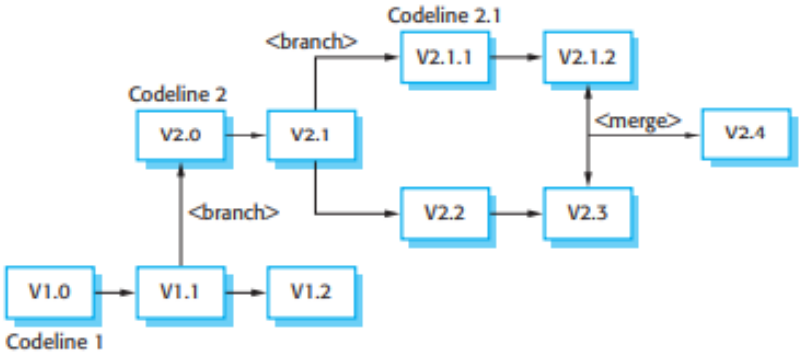

Figure: Branching and Merging

3. System Building

System building is the process of creating a complete, executable system by compiling and linking the system components, external libraries, configuration files, etc.

System building involves assembling a large amount of information about the software and its operating environment.

You may have to link these with externally provided libraries, data files (such as a file of error messages), and configuration files that define the target installation.

You may have to specify the versions of the compiler and other software tools that are to be used in the build.

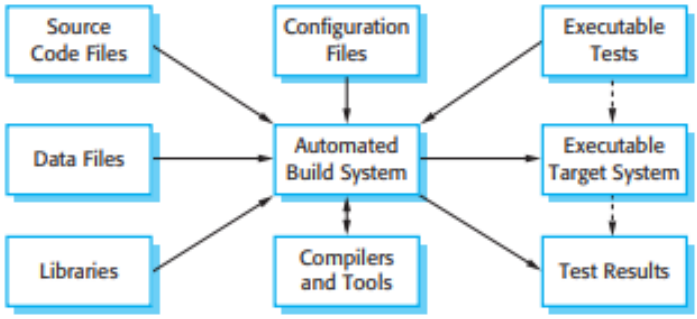

Figure: System Building

System building involves three different system platforms:

1. **The development system**, which includes development tools such as compilers and source code editors. Developers check out code from the version control system into a private workspace before making changes to the system.
2. **The build server**, which is used to build definitive, executable versions of the system.

3. **The target environment**, which is the platform on which the system executes.

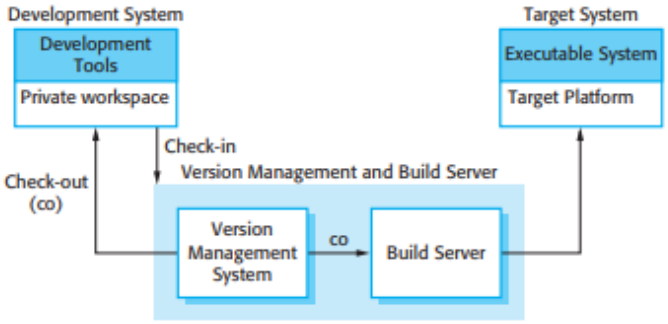

Figure: Development, build and target Platforms

4. System Release

- A system release is a version of a software system that is distributed to customers.
- In simple terms, a release is new or altered software, including the process of its creation
- Release management refers to the **process of planning, designing, scheduling, testing, deploying, and controlling software releases.**
- It ensures that release teams efficiently deliver the applications and upgrades required by the business while maintaining the integrity of the existing production environment.
- For **mass market software**, it is usually possible to identify two types of release:
    - **major releases** which deliver significant new functionality (e.g. v2.0), and
    - **minor releases,** which Focuses on smaller updates or improvements (e.g. v2.1)..
- For **custom software or software product lines**, releases of the system may have to be produced for each customer and individual customers may be running several different releases of the system at the same time.
- System Release involves **Release planning.**
    - Involves deciding when and what to release based on development progress and user needs.
    - Release schedules are typically aligned with project milestones or business goals.

***