Configuration Management and Maintenance

CSC224

Lecture 9

Learning Objectives:

- Understand the role of configuration management in software development.
- Learn the processes and tools used for version control, change management, and release management.
- Understand the types and importance of software maintenance.
- Explore best practices for maintaining long-term software quality and reliability.

What is?

- **1. What is Configuration Management?**
- **Definition:**
  Configuration management (CM) is a discipline for managing changes in software systems, ensuring consistency, reliability, and traceability throughout the development and maintenance lifecycle.
- Configuration management (CM) is concerned with the policies, processes, and tools for managing changing software systems (Aiello and Sachs 2011).

2. Components of Configuration Management

- **Key Goals of CM:**
  - **A. Version Control:** Track changes to code and documents.
  - **B. Change Management:** Control and document modifications.
  - **C. Build Management:** Automate and streamline the software build process.
  - **D. Release Management:** Plan and manage software deployments.

1. *Version control* This involves 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.
2. *System building* This is the process of assembling program components, data, and libraries, then compiling and linking these to create an executable system.
3. *Change management* This involves keeping track of requests for changes to delivered software from customers and developers, working out the costs and impact of making these changes, and deciding if and when the changes should be implemented.
4. *Release management* This involves preparing software for external release and keeping track of the system versions that have been released for customer use.

2. Components of Configuration Management

- **A. Version Control**
- Tracks revisions to software artifacts (e.g., source code, configuration files).
- Enables collaboration among developers by allowing multiple people to work on the same project.
- **Version Control Systems (VCS):**
  - **Centralized VCS:** Single repository for storing files (e.g., Subversion, CVS).
  - **Distributed VCS:** Each developer has a complete copy of the repository (e.g., Git, Mercurial).

- **B. Change Management**
- Manages requests for modifications to software or systems.
- Ensures all changes are reviewed, tested, and approved before implementation.
- **Change Management Process:**
  1. Submit a change request (CR).
  2. Analyze the impact of the proposed change.
  3. Approve or reject the CR.
  4. Implement and test the change.
  5. Update documentation.

- **C. Build Management**
  - Automates the process of compiling code, linking libraries, and packaging the software.
  - Tools: Maven (Java), Gradle (Android), Make (C/C++).
- **D. Release Management**
  - Coordinates the planning, development, testing, and deployment of software.
  - Ensures smooth transitions between development, testing, and production environments.

Software Maintenance

- **3. What is Software Maintenance?**
- **Definition:**
  Software maintenance is the process of modifying and updating software after deployment to correct issues, improve performance, or adapt to changing requirements.

4. Types of Software Maintenance

- **A. Corrective Maintenance**
  - Fixes bugs and defects identified after the software is deployed.
  - Example: Patching a security vulnerability in a web application.
- **B. Adaptive Maintenance**
  - Modifies software to work with changes in the environment (e.g., operating system updates, hardware upgrades).
  - Example: Updating a mobile app to be compatible with a new iOS version.

4. Types of Software Maintenance

- **C. Perfective Maintenance**
  - Enhances the software to improve functionality or performance.
  - Example: Adding new features based on user feedback.
- **D. Preventive Maintenance**
  - Makes changes to prevent potential issues and ensure future reliability.
  - Example: Refactoring code to reduce technical debt.

5. Challenges in Configuration Management and Maintenance

- **Complexity of Changes:**
  - Large systems often involve interdependencies, making changes challenging.
- **Version Conflicts:**
  - When multiple developers make changes to the same part of the system, merging conflicts can occur.
- **Technical Debt:**
  - Accumulation of shortcuts or poor practices in the codebase that require rework.
- **Evolving Requirements:**
  - Adapting the software to meet changing business or user needs.
- **Resource Constraints:**
  - Balancing maintenance activities with new feature development.

6. Tools for Configuration Management and Maintenance

- **Version Control Tools:**
    - Git, GitHub, GitLab, Bitbucket.
- **Build Management Tools:**
    - Maven, Gradle, Ant.
- **Continuous Integration/Continuous Deployment (CI/CD) Tools:**
    - Jenkins, CircleCI, Travis CI.
- **Bug Tracking and Change Management Tools:**
    - Jira, Bugzilla, Redmine.

7. Best Practices for Configuration Management

- **Establish Baselines:**
  - Define key milestones where the system is considered stable and ready for release.
- **Use Branching Strategies:**
  - Implement branching models like GitFlow to manage feature development, bug fixes, and releases.
- **Automate Builds and Tests:**
  - Use CI/CD pipelines to ensure consistent builds and testing.
- **Maintain Documentation:**
  - Update logs, configurations, and deployment instructions with every change.
- **Regular Audits:**
  - Periodically review configurations, codebases, and deployment environments.

8. Maintenance Metrics

- **Mean Time to Repair (MTTR):**
  - Average time taken to fix a defect after detection.
- **Defect Density:**
  - Number of defects per unit of code (e.g., per 1,000 lines of code).
- **Code Churn:**
  - Amount of code added, modified, or deleted over a period.
- **Customer-Reported Issues:**
  - Number of defects reported by users post-deployment.

9. Example: Managing a Library Management System

Scenario: A library management system is experiencing issues with book inventory synchronization.

- **Configuration Management:**
  - Use Git to track code changes made by developers fixing the synchronization bug.
  - Implement a branching strategy to isolate bug fixes from feature development.
- **Maintenance Activities:**
  - **Corrective Maintenance:** Fix the synchronization bug.
  - **Perfective Maintenance:** Add a feature for automatic email reminders for overdue books.
  - **Preventive Maintenance:** Refactor the inventory management module to improve performance.

10. Key Takeaways

- Configuration management ensures systematic handling of changes and consistency across software versions.
- Software maintenance is an ongoing process crucial for adapting to evolving needs and ensuring reliability.
- Best practices and tools for configuration management streamline development and reduce risks.
- Maintenance activities should be prioritized to balance addressing defects, adapting to changes, and improving performance.

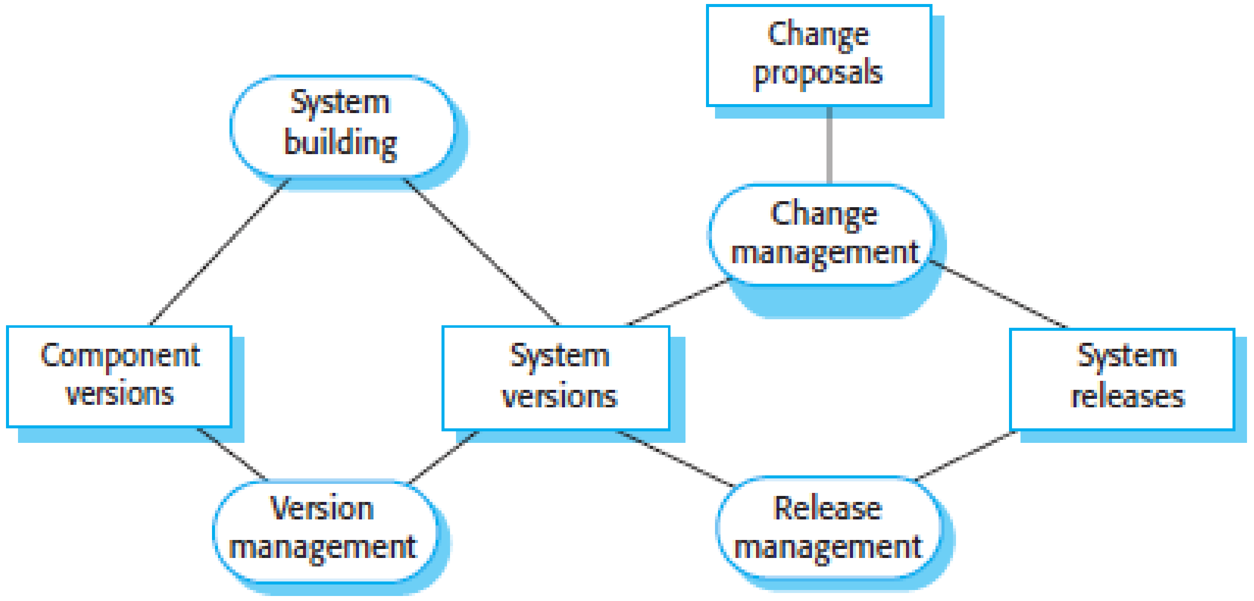

Figure 25.1 Configuration management activities

Discussion Questions

- Why is version control essential for collaborative software development?
- Compare corrective and adaptive maintenance. Why are both critical for long-term software reliability?
- How can automated build and deployment tools improve configuration management?

Practical Activity

- **Objective:** Practice configuration management and maintenance.
- **Task:**
  - Choose a simple project (e.g., a library system or task tracker).
  - Create a Git repository and implement branching for feature development and bug fixes.
  - Simulate a maintenance activity by identifying a "bug" and resolving it.
  - Document the process, including change logs and test results.