9. CONFIGURATION MANAGEMENT

Contents:

8.0 Configuration management

Configuration Management

- Software systems are constantly changing during development and use.
- **Configuration management** (CM) is concerned with the policies, processes and tools for managing changing software systems.
- We need CM because it is easy to lose track of what changes and component versions have been incorporated into each system version.
- CM is essential for team projects to control changes made by different developers.

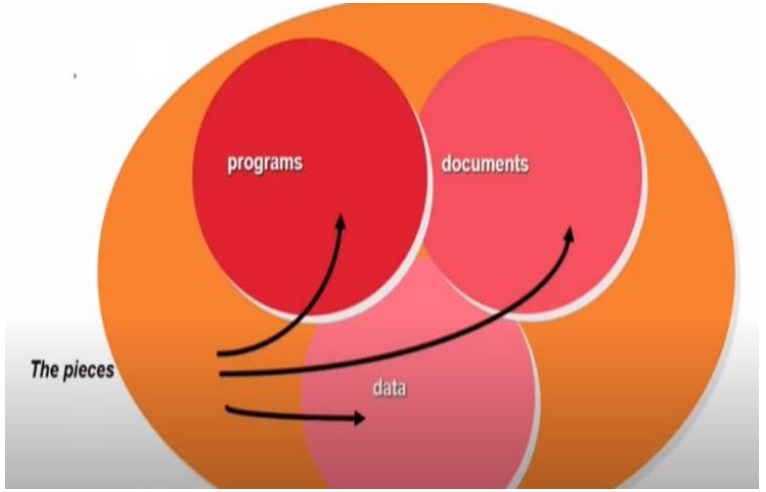


Figure:
Software Configurations

In Software Engineering,

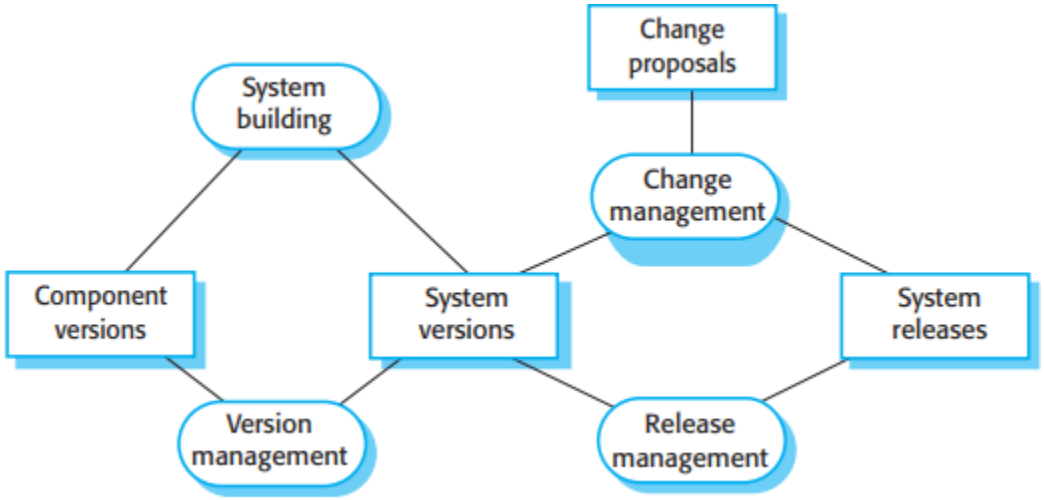
- Software Configuration Management(SCM) **is a process to systematically manage, organize, and control the changes in the documents, codes, and other entities** during the Software Development Life Cycle.

Aim of Configuration management

- 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.

Configuration Management Activities

1. **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.
2. **System building:** The process of assembling program components, data and libraries, then compiling these to create an executable system.
3. **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.
4. **Release management:** Preparing software for external release and keeping track of the system versions that have been released for customer use.



8.1 Version management

i) Version Management:

- Version management (VM) is the process of **keeping track of different versions** of software components or configuration items and the systems in which these components are used.
- It also involves ensuring that changes made by different developers to these versions do not interfere with each other.
- Therefore, version management can be thought of as **the process of managing codelines and baselines.**
- Instead of keeping a complete copy of each version, the **system stores a list of differences (deltas) between one version and another.**
- By applying these to a master version (usually the most recent version), a target version can be recreated.

- 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**.

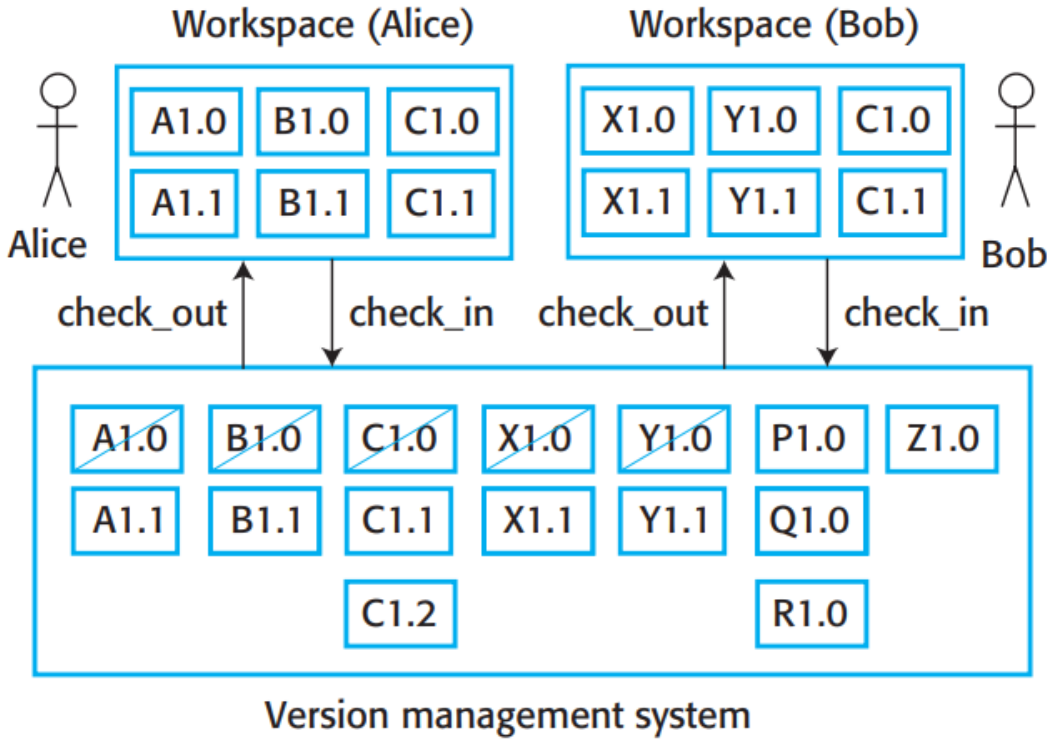


Figure: Check-in and check-out from a centralized version repository

Version Control Systems

- Version control (VC) systems identify, store and control access to the different versions of components. There are two types of modern version control system
  - **Centralized systems,** where there is a **single master repository** that maintains all versions of the software components that are being developed. Subversion is a widely used example of a centralized VC system.
  - **Distributed systems,** where **multiple versions of the component repository** exist at the same time. Git is a widely-used example of a distributed VC system.
- **Key Features of Version Control System:**
  1. Version and release identification
  2. Change history recording
  3. Support for independent development
  4. Project support
  5. Storage management

Distributed version control

- **A ‘master’ repository is created on a server** that maintains the code produced by the development team.
- Instead of checking out the files that they need, a **developer creates a clone** of the project repository that is downloaded and installed on their computer.
- Developers work on the files required and maintain the new versions on their private repository on their own computer.
- **When changes are done, they ‘commit’ these changes and update their private server repository.** They may then ‘push’ these changes to the project repository.

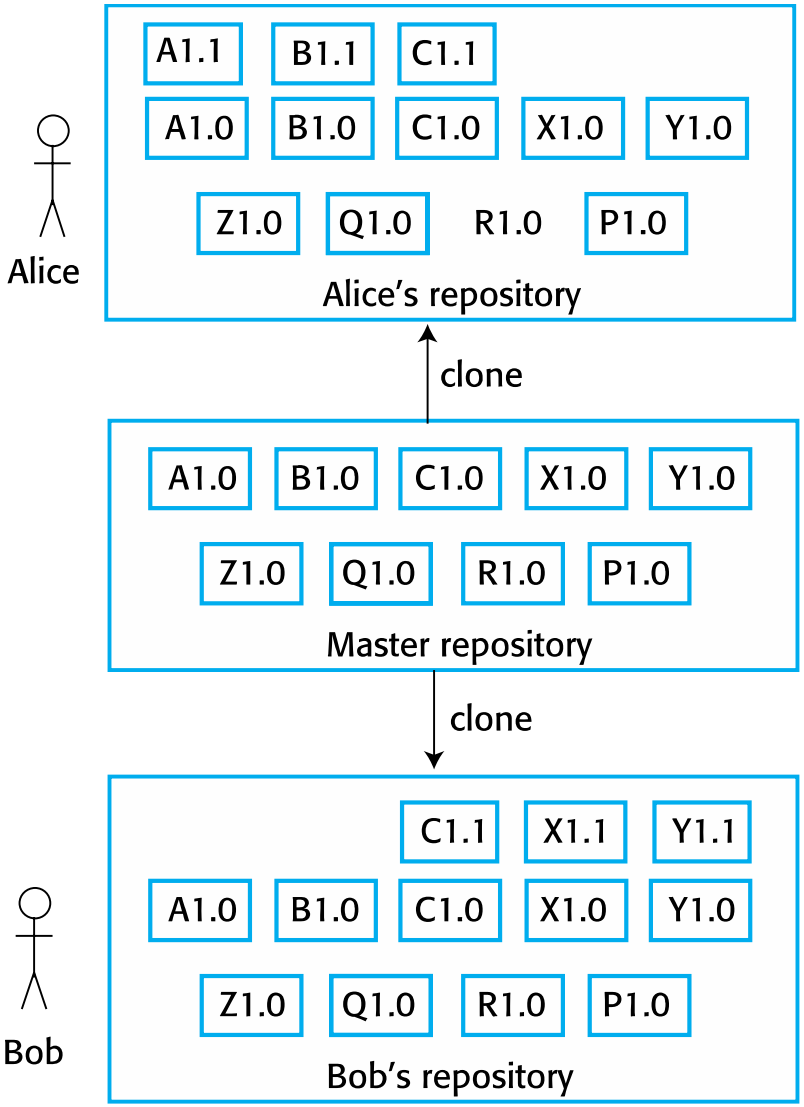


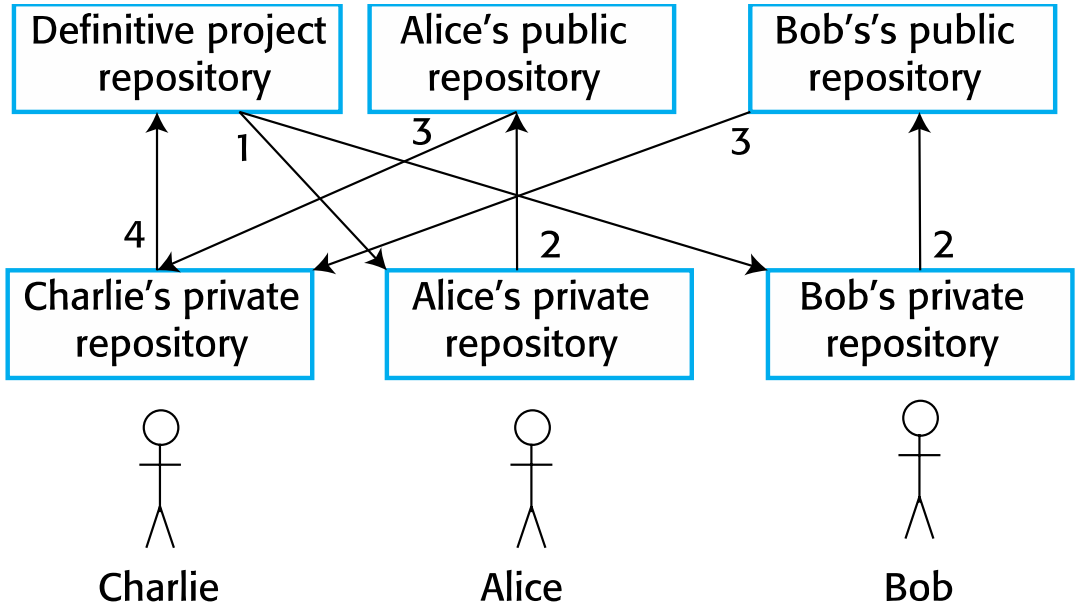
Figure: Repository cloning

Benefits of distributed version control

- It **provides a backup mechanism** for the repository.
  - If the repository is corrupted, work can continue and the project repository can be restored from local copies.
- It **allows for off-line working** so that developers can commit changes if they do not have a network connection.
- Project support is the default way of working.
  - Developers can compile and test the entire system **on their local machines** and test the changes that they have made.

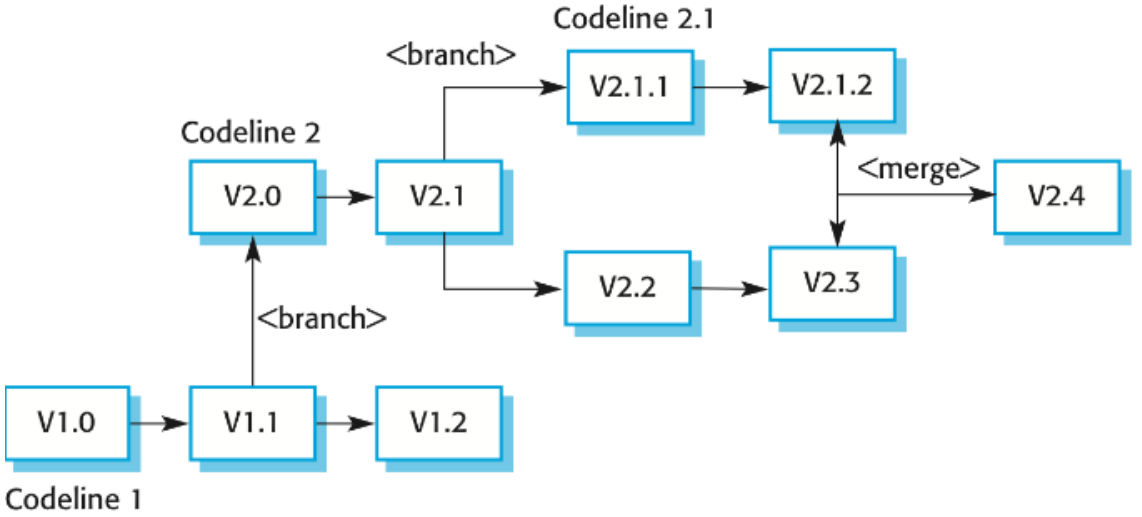
Open source development

- Distributed version control is essential for open source development.
  - Several people may be **working simultaneously** on the same system without any central coordination.
- As well as a **private repository** on their own computer, developers also maintain a public server repository to which they push new versions of components that they have changed.
  - It is then up to the open-source system 'manager' to decide when to pull these changes into the definitive system.



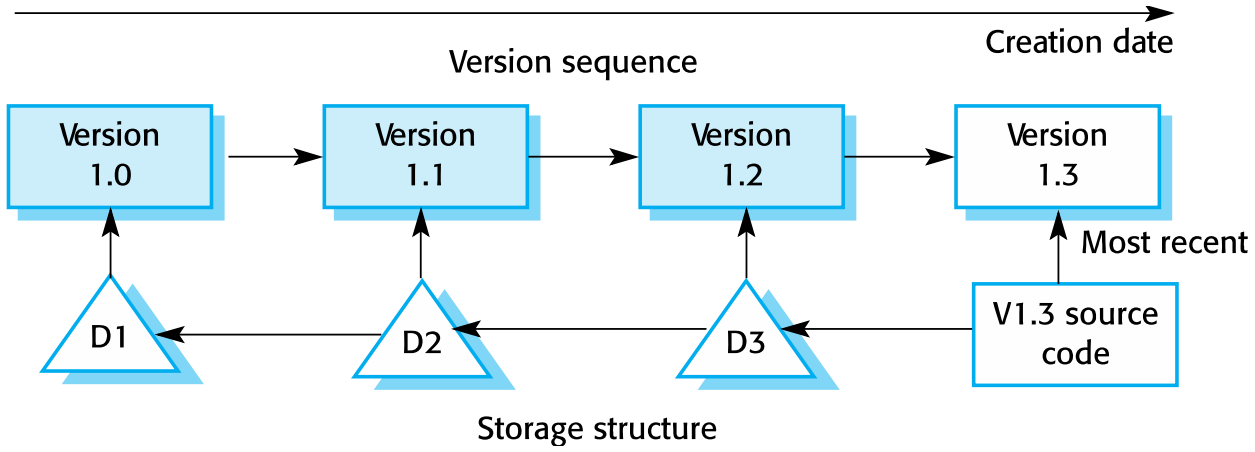
Branching and Merging

- Rather than a linear sequence of versions that reflect changes to the component over time, there may be several independent sequences.
  - This is normal in system development, where 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.
  - If the changes made involve different parts of the code, the component versions may be merged automatically by combining the deltas that apply to the code.



Storage management

- When version control systems were first developed, storage management was one of their most important functions.
- Disk space was expensive and it was important to minimize the disk space used by the different copies of components.
- **Instead of keeping a complete copy of each version, the system stores a list of differences (deltas) between** one version and another.
  - By applying these to a master version (usually the most recent version), a target version can be recreated.



8.2 System Building

ii) 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.

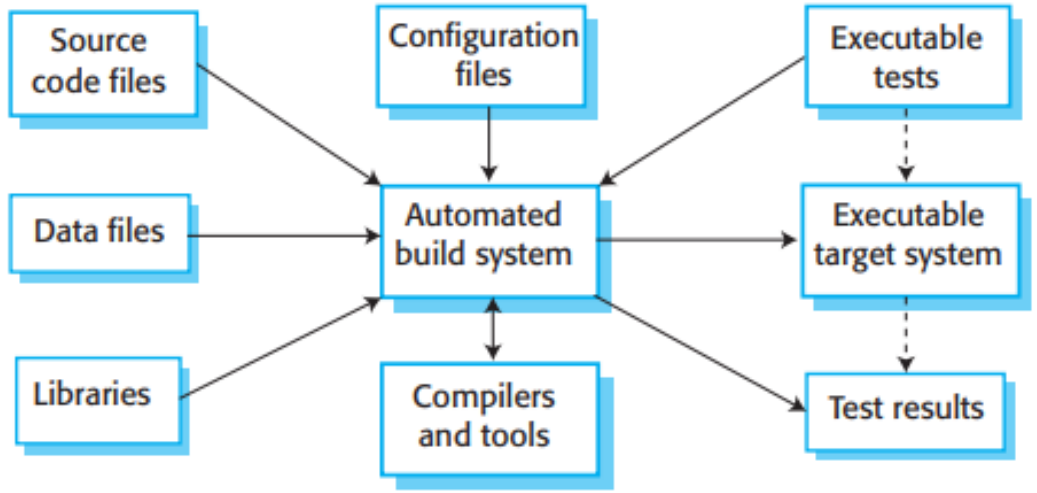
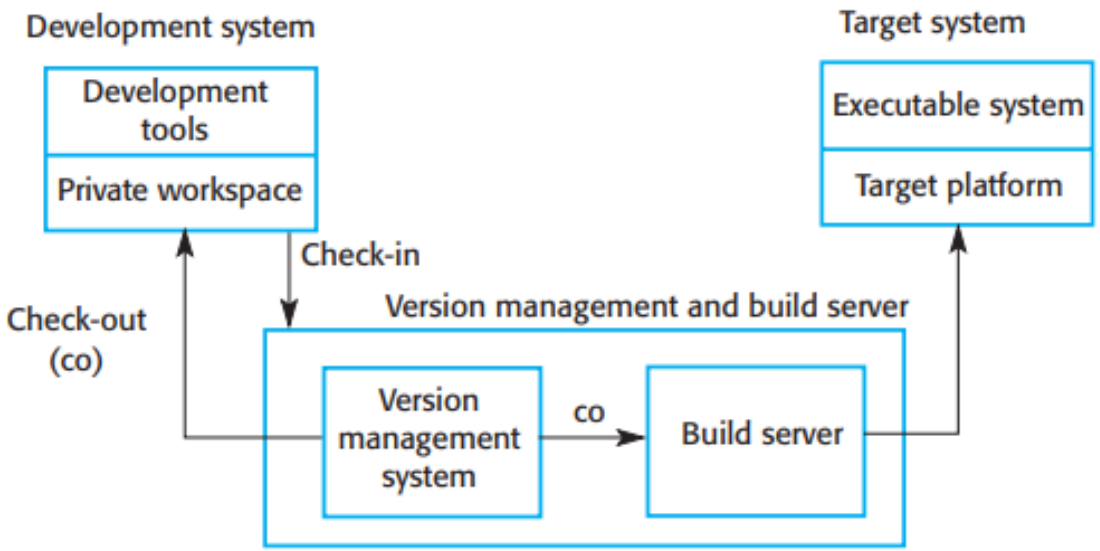


Figure: System building

- 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.



8.3 Change management

iii) Change Management

- **Organizational needs and requirements change** during the lifetime of a system.
- Bugs have to be repaired and systems have to adapt to changes in their environment.
- Change management is intended to **ensure that system evolution is a managed process** and that **priority is given to the most urgent and cost-effective changes**.
- The change management process is concerned with
  - analyzing the costs and benefits of proposed changes,
  - approving those changes that are worthwhile and
  - tracking which components in the system have been changed.

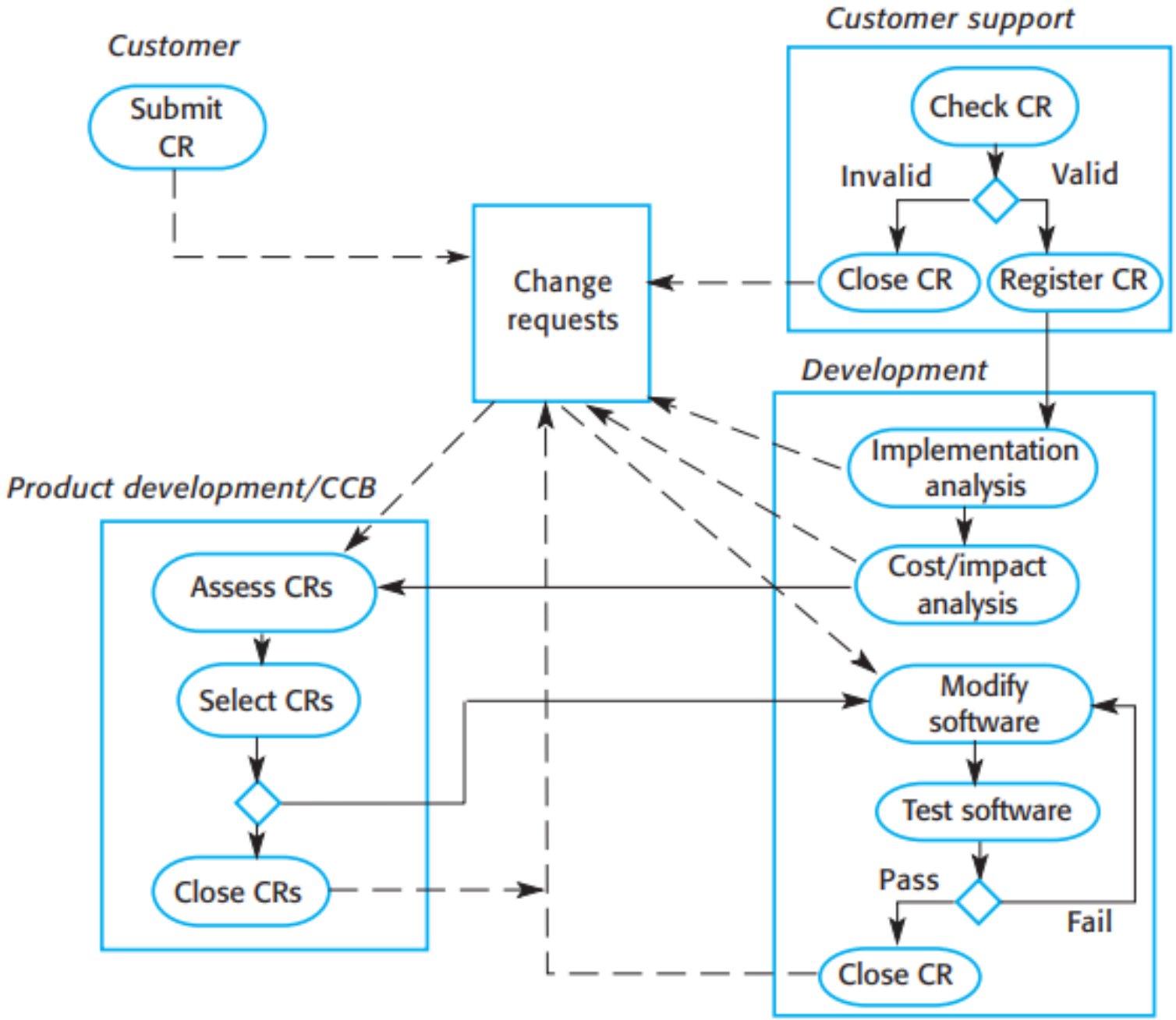


Figure: The change management process

Change Request Form

Project: SICSA/AppProcessing **Number:** 23/02
Change requester: I. Sommerville **Date:** 20/07/12
Requested change: The status of applicants (rejected, accepted, etc.) should be shown visually in the displayed list of applicants.

Change analyzer: R. Looek **Analysis date:** 25/07/12
Components affected: ApplicantListDisplay, StatusUpdater

Associated components: StudentDatabase

Change assessment: Relatively simple to implement by changing the display color according to status. A table must be added to relate status to colors. No changes to associated components are required.

Change priority: Medium
Change implementation:
Estimated effort: 2 hours
Date to SGA app. team: 28/07/12 **CCB decision date:** 30/07/12
Decision: Accept change. Change to be implemented in Release 1.2
Change implementor: **Date of change:**
Date submitted to QM: **QM decision:**
Date submitted to CM:
Comments:

Figure: A partially completed change request form

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

8.4 Release management

iv) Release Management

- 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.
- 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
- For **mass market software**, it is usually possible to identify two types of release:
  - major releases which deliver significant new functionality, and
  - minor releases, which repair bugs and fix customer problems that have been reported.
- 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.

Factors influencing system release planning:

1. **Competition:**
   - For mass-market software, a new system release may be necessary because a competing product has introduced new features.
   - and market share may be lost if these are not provided to existing customers.
2. **Marketing requirements:**
   - The marketing department of an organization may have made a commitment for releases to be available at a particular date.
3. **Platform changes:**
   - You may have to create a new release of a software application when a new version of the operating system platform is released.
4. **Technical quality of the system:**
   - If serious system faults are reported which affect the way in which many customers use the system, it may be necessary to issue a fault repair release.
   - Minor system faults may be repaired by issuing patches (usually distributed over the Internet) that can be applied to the current release of the system.

Release creation

- The executable code of the programs and all associated data files must be identified in the version control system.
- Configuration descriptions may have to be written for different hardware and operating systems.
- Update instructions may have to be written for customers who need to configure their own systems.
- Scripts for the installation program may have to be written.
- Web pages have to be created describing the release, with links to system documentation.
- When all information is available, an executable master image of the software must be prepared and handed over for distribution to customers or sales outlets.

Release tracking

- In the event of a problem, it may be necessary to reproduce exactly the software that has been delivered to a particular customer.
- When a system release is produced, it must be documented to ensure that it can be re-created exactly in the future.
- This is particularly important for customized, long-lifetime embedded systems, such as those that control complex machines.
  - Customers may use a single release of these systems for many years and may require specific changes to a particular software system long after its original release date.

Release reproduction

- To document a release, you have to record the specific versions of the source code components that were used to create the executable code.
- You must keep copies of the source code files, corresponding executables and all data and configuration files.
- You should also record the versions of the operating system, libraries, compilers and other tools used to build the software.

Release planning

- As well as the technical work involved in creating a release distribution, advertising and publicity material have to be prepared and marketing strategies put in place to convince customers to buy the new release of the system.
- Release timing
  - If releases are too frequent or require hardware upgrades, customers may not move to the new release, especially if they have to pay for it.
  - If system releases are too infrequent, market share may be lost as customers move to alternative systems.

- Reference for Release management:
- https://www.simplilearn.com/release-management-process-article

8.5 Case tools for configuration management

CASE tools for Configuration Management

- CASE tools help in this by automatic tracking, version management and release management.
- For example, Fossil, Git, Accu REV.
- An instance of software is released under one version. Configuration Management tools deal with –
  1. Version and revision management
  2. Baseline configuration management
  3. Change control management

Some popular tools for SCM

1. **Git:** Git is a free and open source tool which helps version control. It is designed to handle all types of projects with speed and efficiency.

   Download link: https://git-scm.com/

2. **Team Foundation Server:** Team Foundation is a group of tools and technologies that enable the team to collaborate and coordinate for building a product.

   Download link: https://azure.microsoft.com/en-us/services/devops/server/

3. **Ansible**: It is an open source Software configuration management tool. Apart from configuration management it also offers application deployment & task automation.

   Download link: https://www.ansible.com/

Git for Version Management

- As disk storage is now relatively cheap, Git uses an alternative, faster approach.
- Git does not use deltas but applies a standard compression algorithm to stored files and their associated meta-information.
- It does not store duplicate copies of files.
- Retrieving a file simply involves decompressing it, with no need to apply a chain of operations.
- Git also uses the notion of packfiles where several smaller files are combined into an indexed single file.
- **Git** is a distributed version control system that tracks changes in any set of computer files, usually used for coordinating work among programmers who are collaboratively developing source code during software development.

References:

- https://cs.ccsu.edu/~stan/classes/CS530/Notes18/25-ConfigurationManagement.html

Brief Answer Questions:

1. Define software configurations.
2. Define SCM.
3. What is the aim of SCM?
4. Mention the SCM activities.
5. What are the key features of version control system(VCS)?
6. List any four tools for configuration management.
7. What do you mean by clone and commit in version control system?
8. Mention any two benefits of Distributed version control system.
9. What are the factors that need to be considered during change analysis? Mention them.
10. Mention the factors influencing system release plan.

Short Answer Questions:

1. What do you mean by software configuration management? Why do we need software configuration management?
2. Explain different software management activities in detail.
3. What is version and version management? How version management is conducted in SCM? Explain.
4. Discuss on different types of version control systems.
5. How distributed version control can be implemented for open source development?
6. Explain the branching and merging approach for version management.
7. How can we optimize disk space during version management? Explain with the storage management technique.
8. How systems are build? What platforms are required in system building?
9. Changes are necessary, but, all the changes cannot be accommodated. Explain this with the change management process.
10. What is release management? Explain release planning , release tracking, release reproduction and release creation.
11. Explain ***github*** as a popular CASE tool for software configuration management.

End of Chapter