Software Configuration Management

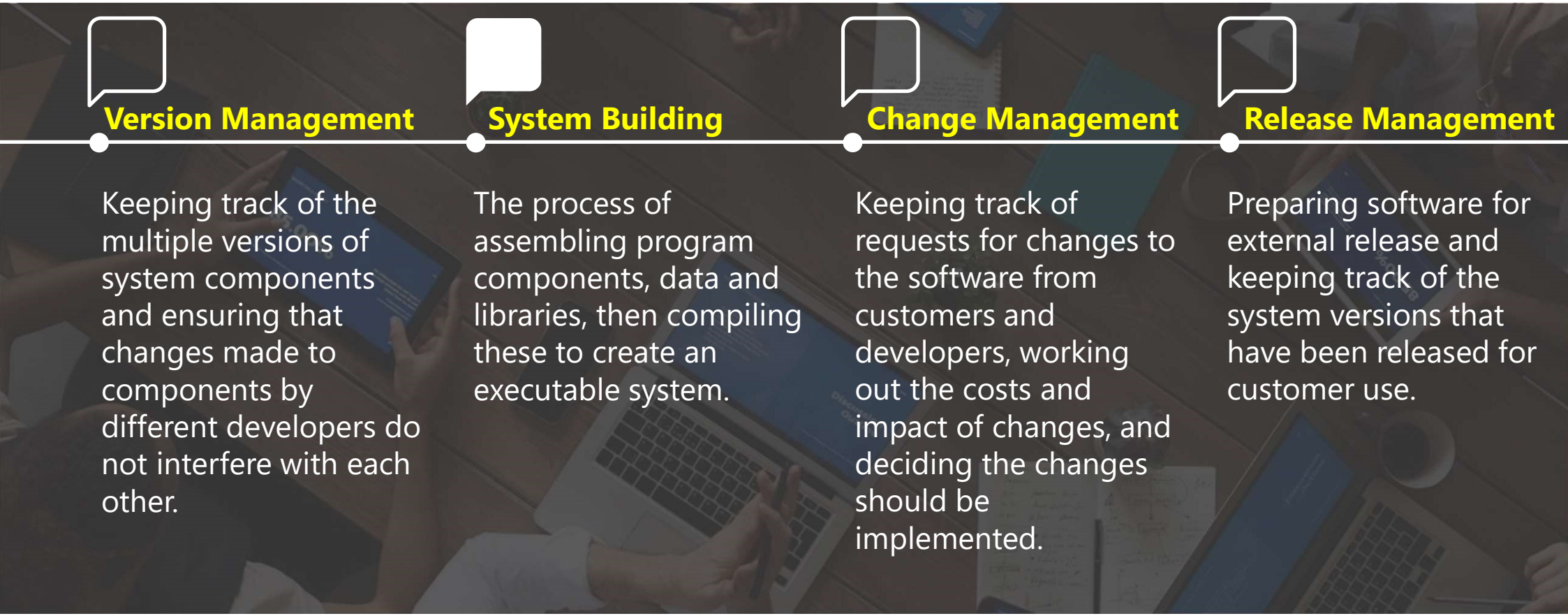
Chapter 7

Table of Contents

7.1 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.
- You 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

7.1 Configuration Management - Activities



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.

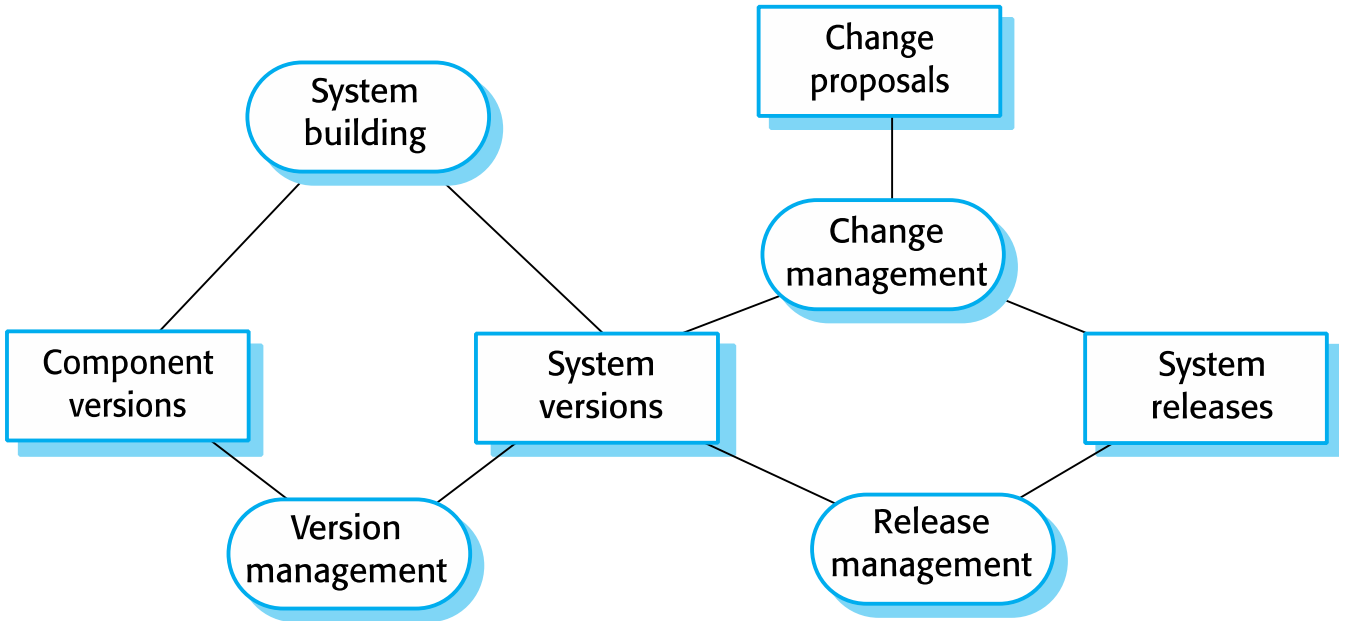
Change Management

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.

7.1 Configuration Management - Activities



Configuration Management at Different Phases

Development phase

The development team is responsible for managing the software configuration when new functionality is being added to the software.

System testing phase

A version of the system is released internally for testing. No new system functionality is added. Changes made are bug fixes, performance improvements and security vulnerability repairs.

Release phase

The software is released to customers for use. New versions of the released system are developed to repair bugs and vulnerabilities and to include new features.

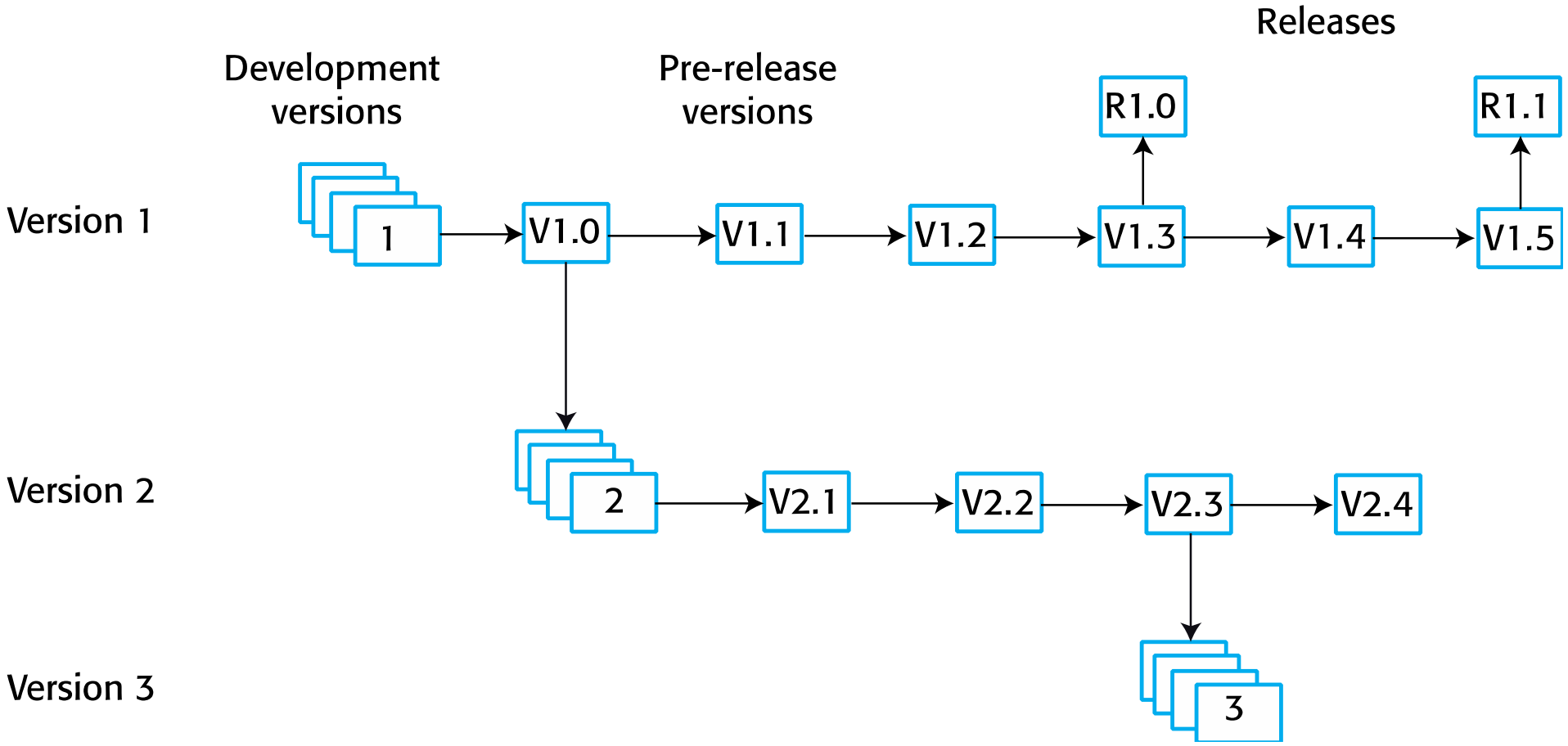
Multi-Version Systems

7.1 Configuration Management

- For large systems, there is never just one 'working' version of a system.
- There are always several versions of the system at different stages of development.
- There may be several teams involved in the development of different system versions.

Multi-version System Development

7.1 Configuration Management



Configuration Management Terminology

7.1 Configuration Management

Term	Explanation
Baseline	A baseline is a collection of component versions that make up a system. Baselines are controlled, which means that the versions of the components making up the system cannot be changed. This means that it is always possible to recreate a baseline from its constituent components.
Branching	The creation of a new codeline from a version in an existing codeline. The new codeline and the existing codeline may then develop independently.
Codeline	A codeline is a set of versions of a software component and other configuration items on which that component depends.
Configuration (version) control	The process of ensuring that versions of systems and components are recorded and maintained so that changes are managed and all versions of components are identified and stored for the lifetime of the system.
Software configuration item (SCI)	Anything associated with a software project (design, code, test data, document, etc.) that has been placed under configuration control. There are often different versions of a configuration item. Configuration items have a unique name.
Mainline	A sequence of baselines representing different versions of a system.

Configuration Management Terminology

7.1 Configuration Management

Term	Explanation
Merging	The creation of a new version of a software component by merging separate versions in different codelines. These codelines may have been created by a previous branch of one of the codelines involved.
Release	A version of a system that has been released to customers (or other users in an organization) for use.
Repository	A shared database of versions of software components and meta-information about changes to these components.
System building	The creation of an executable system version by compiling and linking the appropriate versions of the components and libraries making up the system.
Version	An instance of a configuration item that differs, in some way, from other instances of that item. Versions always have a unique identifier.
Workspace	A private work area where software can be modified without affecting other developers who may be using or modifying that software.

7.2 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.

Codelines and Baselines

7.2 Version Management

Codeline

- A codeline is a sequence of versions of source code with later versions in the sequence derived from earlier versions.
- Codelines normally apply to components of systems so that there are different versions of each component.

Baseline

- A baseline is a definition of a specific system.
- The baseline therefore specifies the component versions that are included in the system plus a specification of the libraries used, configuration files, etc.

Codelines and Baselines

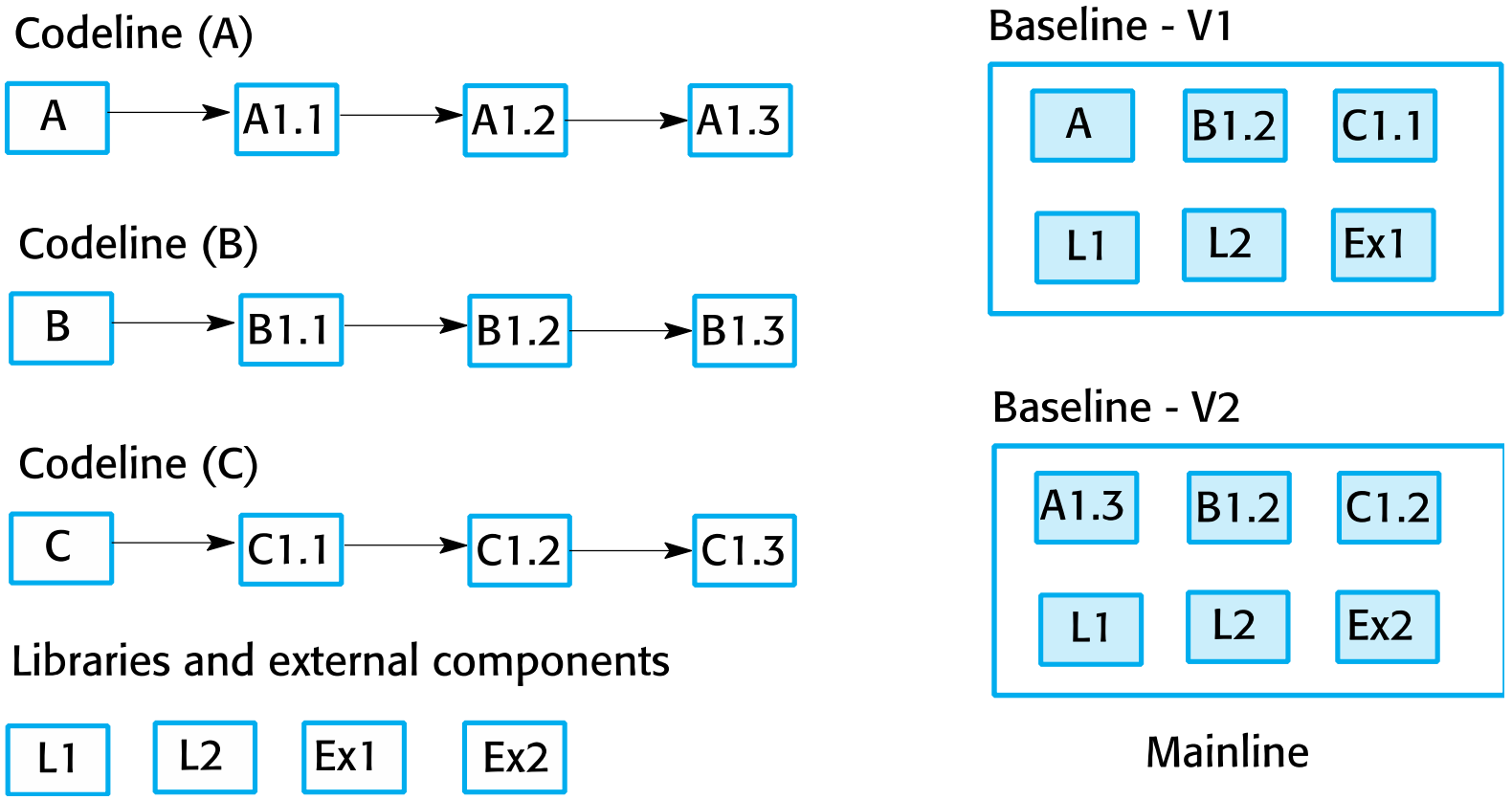
7.2 Version Management

Baseline

- Baselines may be specified using a configuration language, which allows you to define what components are included in a version of a particular system.
- Baselines are important because you often have to recreate a specific version of a complete system.
- For example, a product line may be instantiated so that there are individual system versions for different customers. You may have to recreate the version delivered to a specific customer if, for example, that customer reports bugs in their system that have to be repaired.

Codelines and Baselines

7.2 Version Management



Version Control Systems

7.2 Version Management

- 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

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

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 VC Systems

7.2 Version Management

- Version and release identification
- Change history recording
- Support for independent development
- Project support
- Storage management

Version and release identification: managed versions are assigned identifiers when submitted.

Change history recording: Changes made to the code of a system or component are recorded.

Support for independent development: keeps track of components for editing and ensures that changes made to a component by different developers do not interfere.

Project support: support development of several projects which share components.

Storage management manage storage space, reduce storage space required by multiple versions of components.

Public Repository & Private Workspaces

7.2 Version Management

- To support independent development without interference, version control systems use the concept of a project repository and a private workspace.
- The project repository maintains the 'master' version of all components. It is used to create baselines for system building.
- When modifying components, developers copy (check-out) these from the public repository into their private workspace and work on these copies.
- When they have finished their changes, the changed components are returned (checked-in) to the public repository.

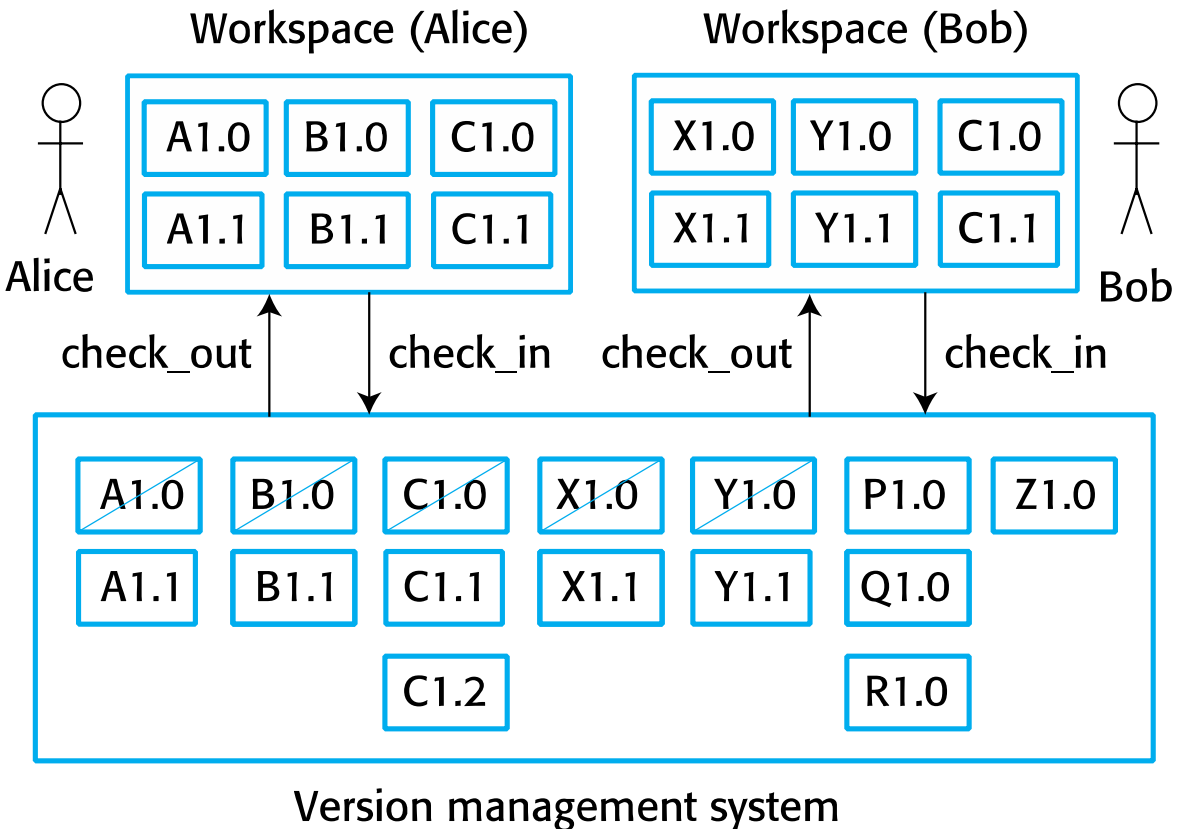
Public Repository & Private Workspaces

Centralized Version Control

- If several people are working on a component at the same time, each check it out from the repository. If a component has been checked out, the VC system warns other users wanting to check out that component that it has been checked out by someone else.

Repository Check-in/Check-out

7.2 Version Management

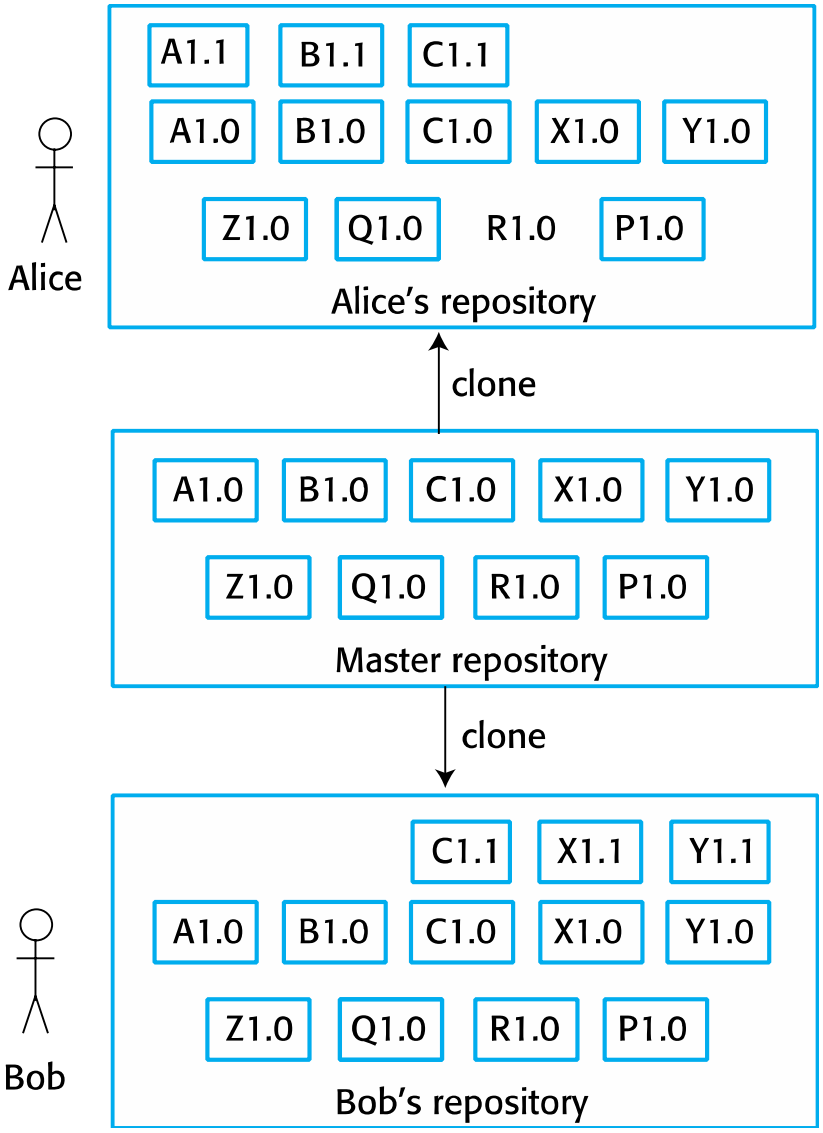


Public Repository & Private Workspaces

7.2 Version 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.



Repository Cloning

Public Repository & Private Workspaces

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.

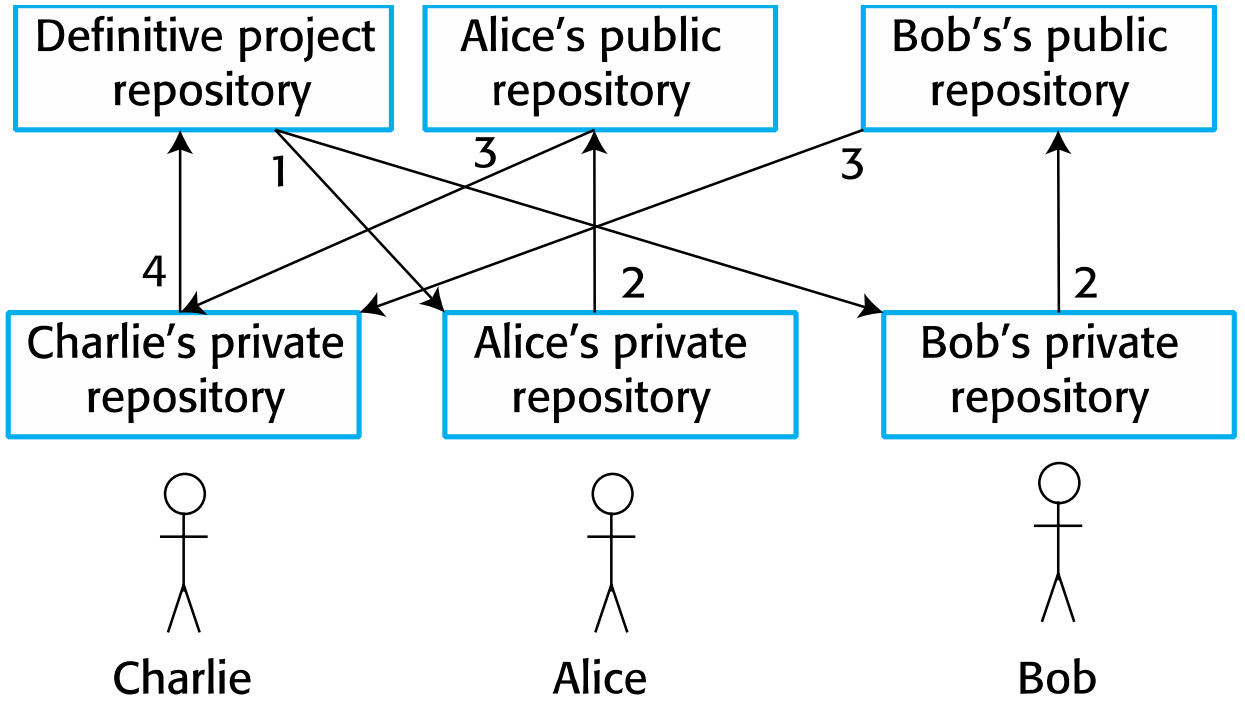
Public Repository & Private Workspaces

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.

Open-Source Development

7.2 Version Management



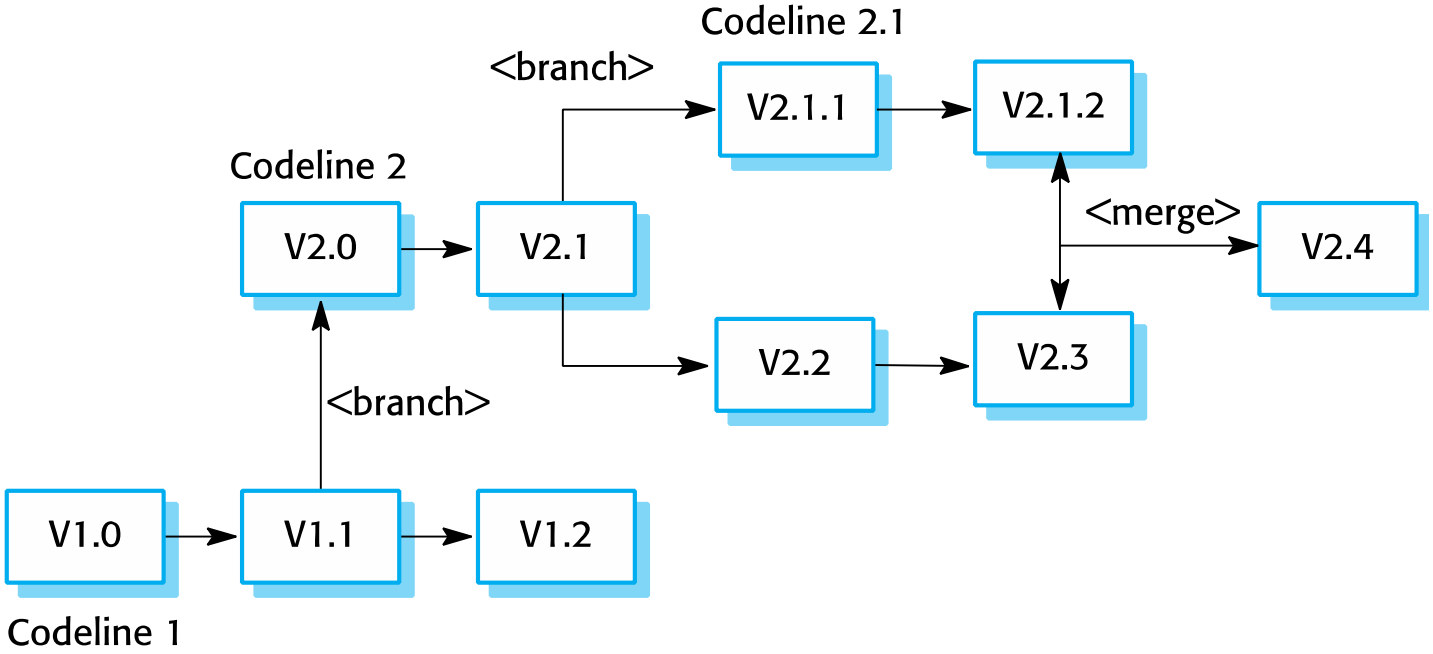
Branching and Merging

7.2 Version Management

- 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.

Branching and Merging

7.2 Version Management



Storage Management

7.2 Version 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.

7.3 System Building

7.2 Version Management

- 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 tools and version management tools must communicate as the build process involves checking out component versions from the repository managed by the version management system.
- The configuration description used to identify a baseline is also used by the system building tool.

Build System Platforms

7.3 System Building

Development System

Includes development tools such as compilers, source code editors, etc.

Developers check out code from the version management system into a private workspace before making changes to the system.

Build Server

Used to build definitive, executable versions of the system.

Developers check-in code to the version management system before it is built. The system build may rely on external libraries that are not included in the version management system.

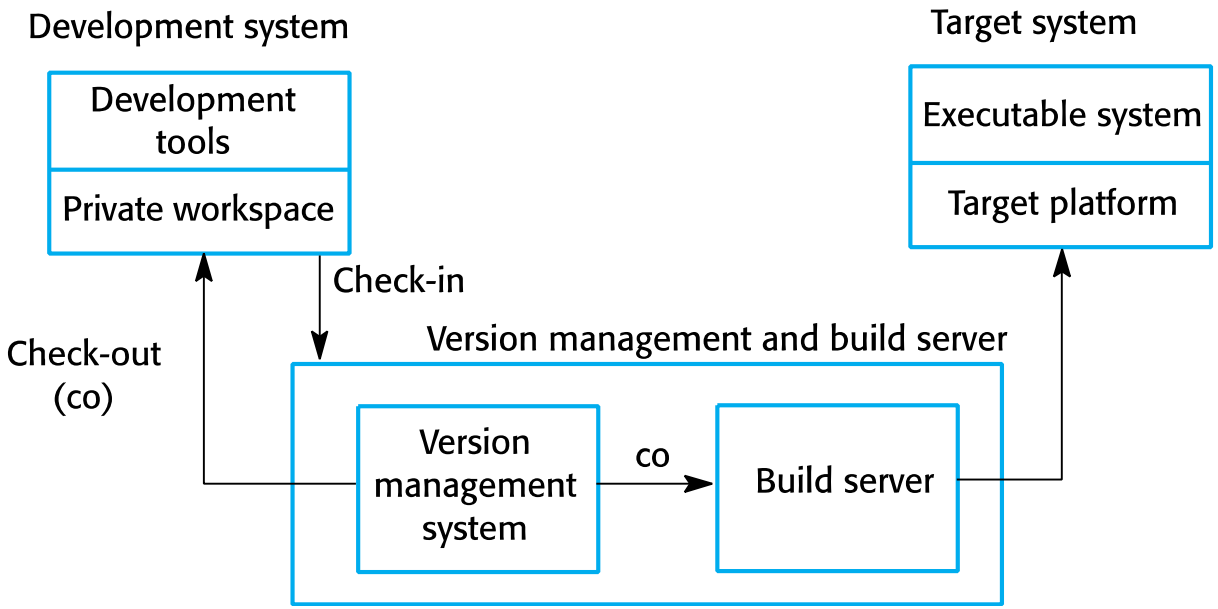
Target Environment

The platform on which the system executes.

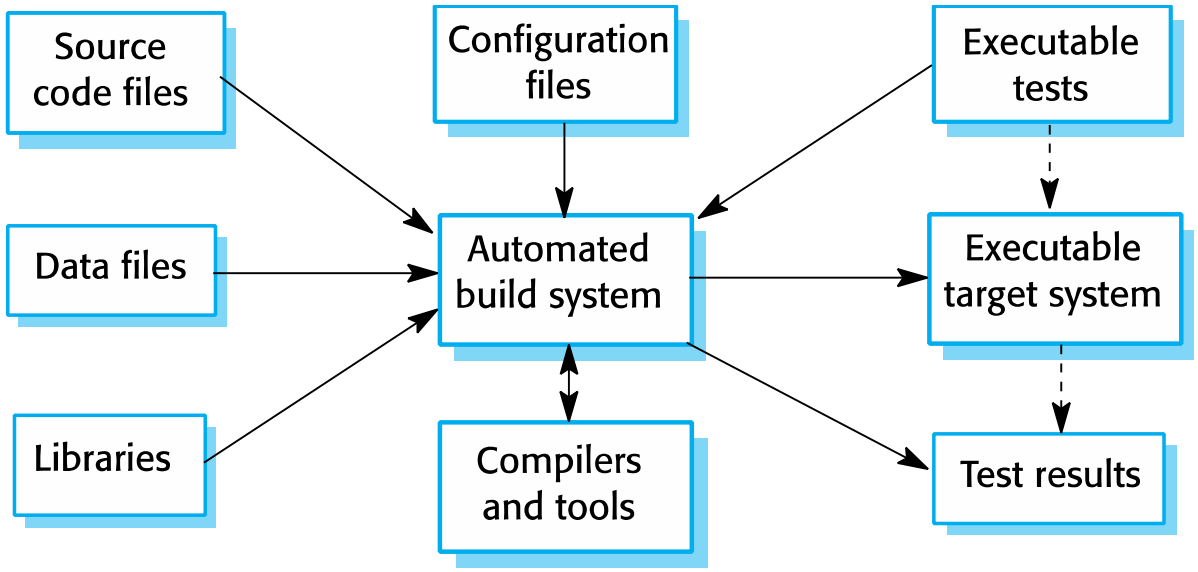
For real-time and embedded systems, the target environment is often smaller and simpler than the development environment (e.g. a cell phone)

Build System Platforms

7.3 System Building



7.3 System Building



Build System Functionality

7.3 System Building

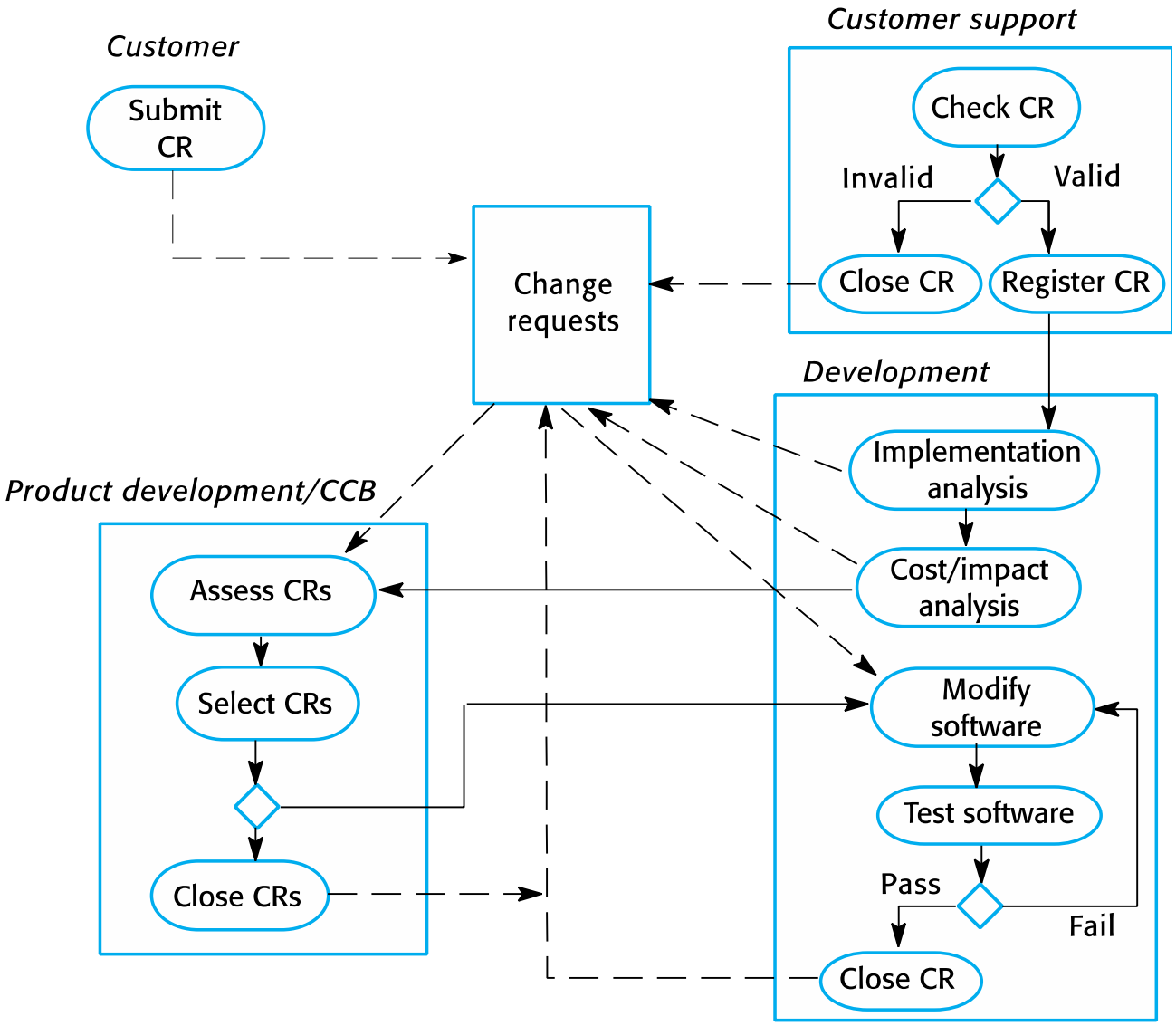
- Build script generation
- Version management system integration
- Minimal re-compilation
- Executable system creation
- Test automation
- Reporting
- Documentation generation

Daily Building Example

- The development organization sets a delivery time (say 2 p.m.) for system components.
  - If developers have new versions of the components that they are writing, they must deliver them by that time.
  - A new version of the system is built from these components by compiling and linking them to form a complete system.
  - This system is then delivered to the testing team, which carries out a set of predefined system tests
  - Faults that are discovered during system testing are documented and returned to the system developers. They repair these faults in a subsequent version of the component.

7.4 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.



The change management process

Note: CR = Change Request

Change request form

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 QA: **QA decision:**
Date submitted to CM:
Comments:

Factors in Change Analysis

7.4 Change Management

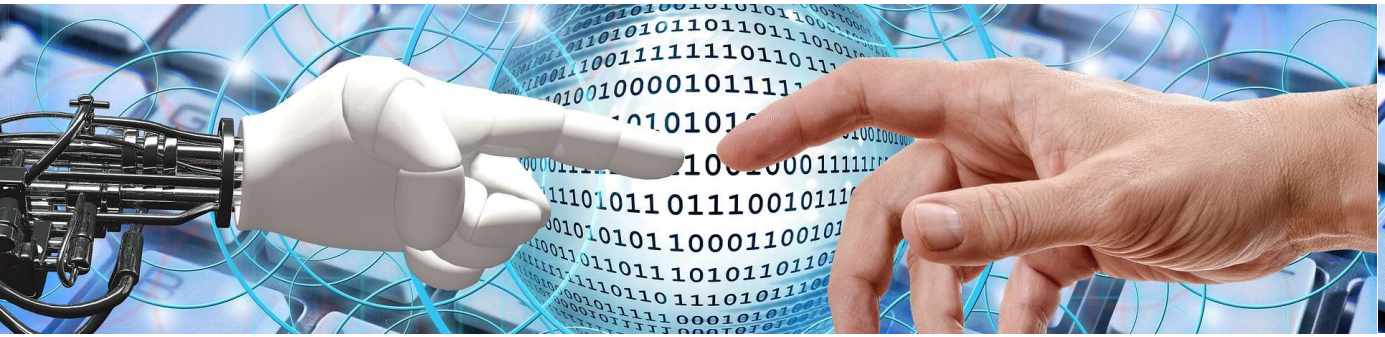
- The consequences of not making the change
- The benefits of the change
- The number of users affected by the change
- The costs of making the change
- The product release cycle

Derivation History

```
// SICSA project (XEP 6087)
//
// APP-SYSTEM/AUTH/RBAC/USER_ROLE
//
// Object: currentRole
// Author: R. Looek
// Creation date: 13/11/2012
//
// © St Andrews University 2012
//
// Modification history
// Version Modifier   Date          Change        Reason
// 1.0     J. Jones   11/11/2009    Add header    Submitted to CM
// 1.1     R. Looek   13/11/2012    New field     Change req. R07/02
```

7.5 Release Management

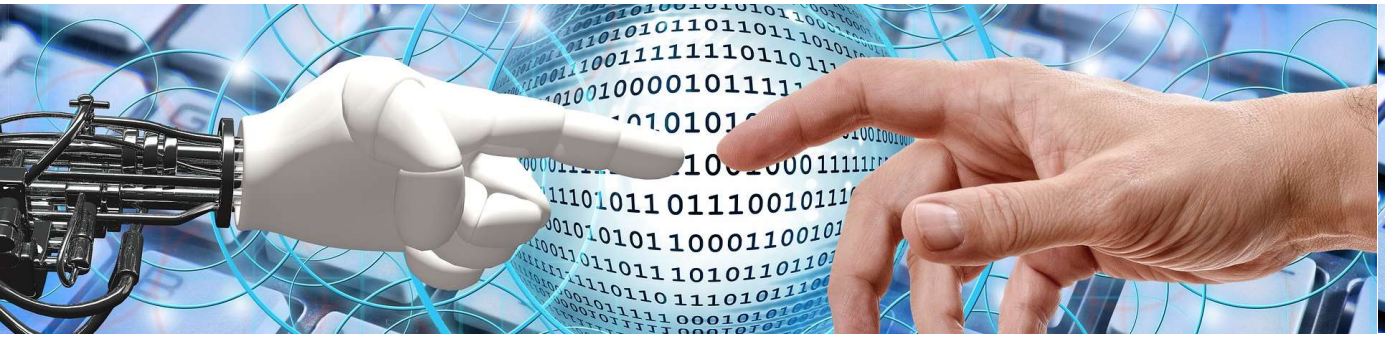
- A system release is a version of a software system that is distributed to customers.
- For mass market software, it is usually possible to identify two types of release: major releases which deliver significant new functionality, and minor releases, which repair bugs and fix customer problems that 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.



Release Components

7.5 Release Management

- As well as the executable code of the system, a release may also include:
  - configuration files defining how the release should be configured for particular installations;
  - data files, such as files of error messages, that are needed for successful system operation;
  - an installation program that is used to help install the system on target hardware;
  - electronic and paper documentation describing the system;
  - packaging and associated publicity that have been designed for that release.



Factors Influencing System Release Planning

7.5 Release Management

Factor	Description
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.
Marketing requirements	The marketing department of an organization may have made a commitment for releases to be available at a particular date.
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.
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.

Summary

7.1 Configuration Management
7.2 Version Management
7.3 System Building
7.4 Change Management
7.5 Release Management