Chapter 25 – Configuration Management

Topics covered

- ✧ Version management
- ✧ System building
- ✧ Change management
- ✧ Release management

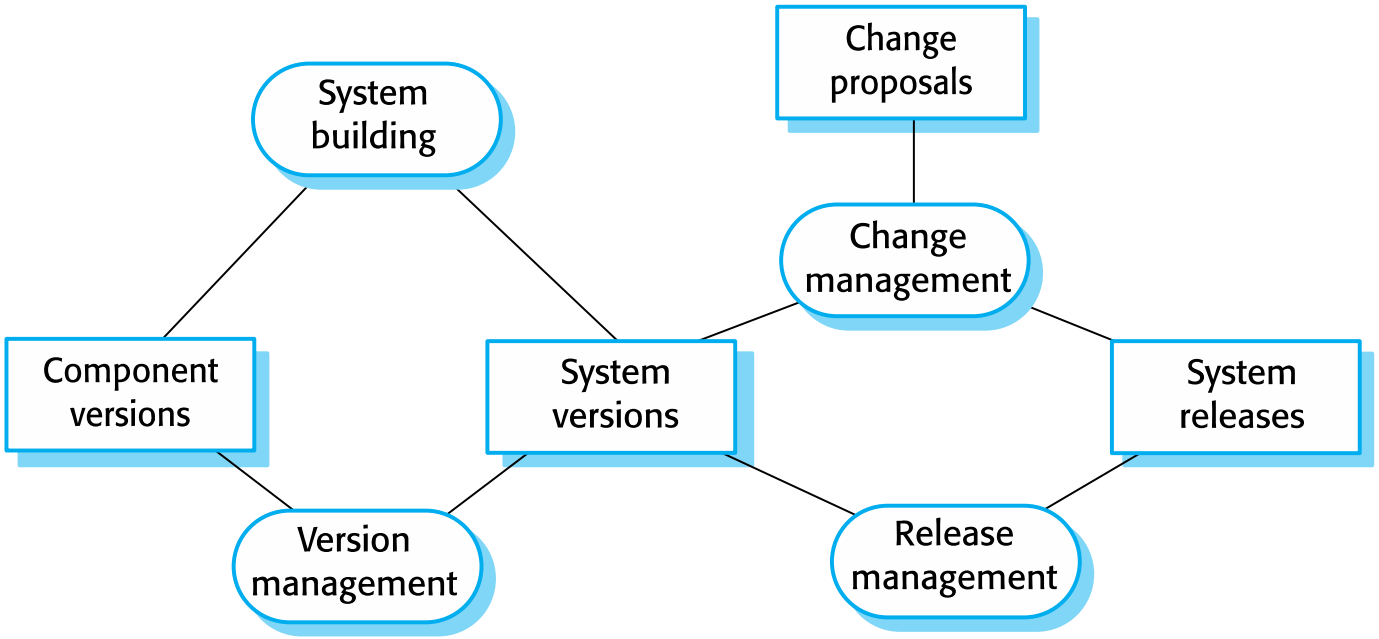
Configuration management

- 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.
- 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
  - Ensuring changes made to components by different developers do not interfere with each other

CM 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.

Configuration management activities



Agile development and CM

- Agile development, where components and systems are changed several times per day, is impossible without using CM tools.
- The definitive versions of components are held in a shared project repository and developers copy these into their own workspace.
- They make changes to the code then use system building tools to create a new system on their own computer for testing. Once they are happy with the changes made, they return the modified components to the project repository.

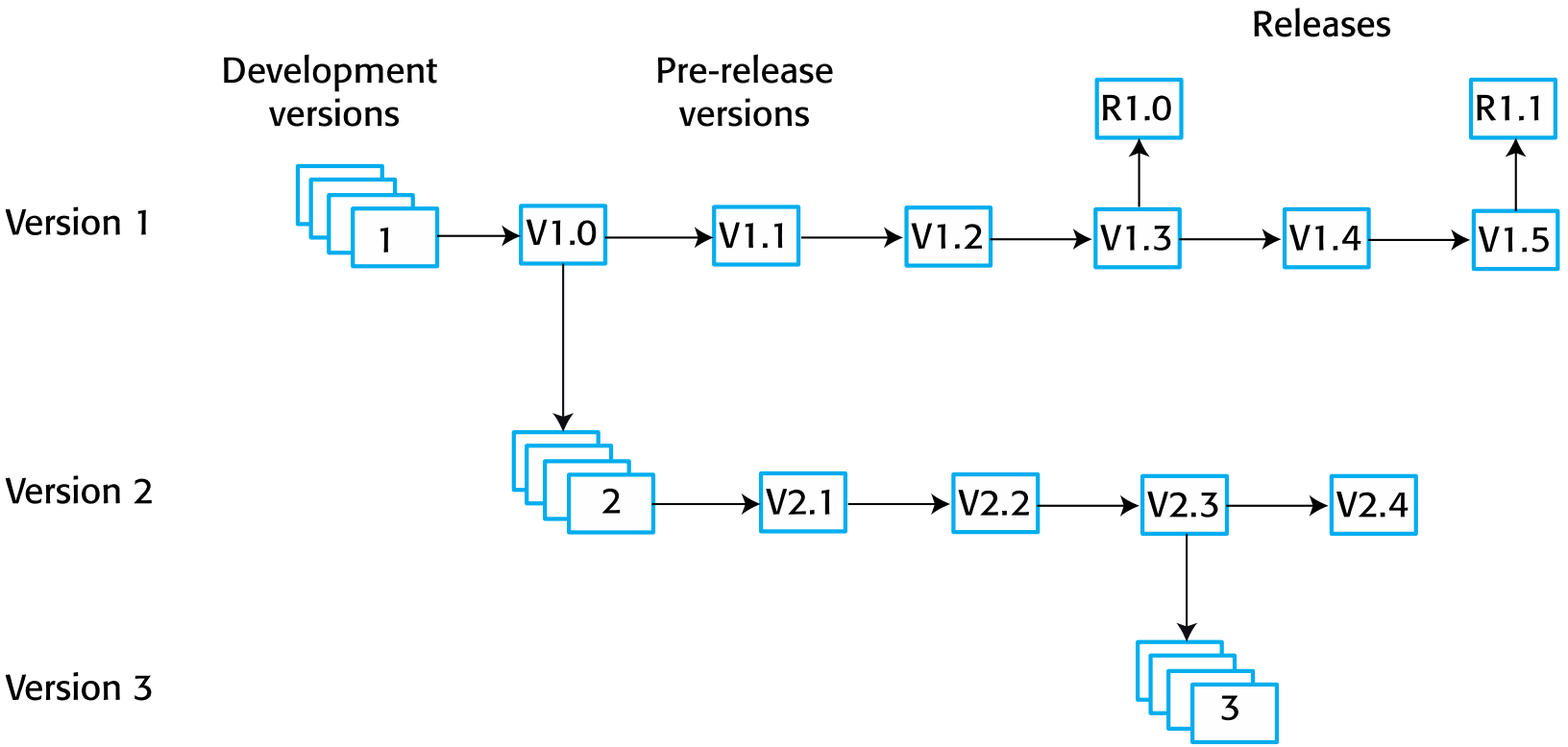
Development phases

- A development phase where the development team is responsible for managing the software configuration and new functionality is being added to the software.
- A system testing phase where 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.
- A release phase where 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

- 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



CM terminology

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.
Configuration item or 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.

CM terminology

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.

Version management

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.

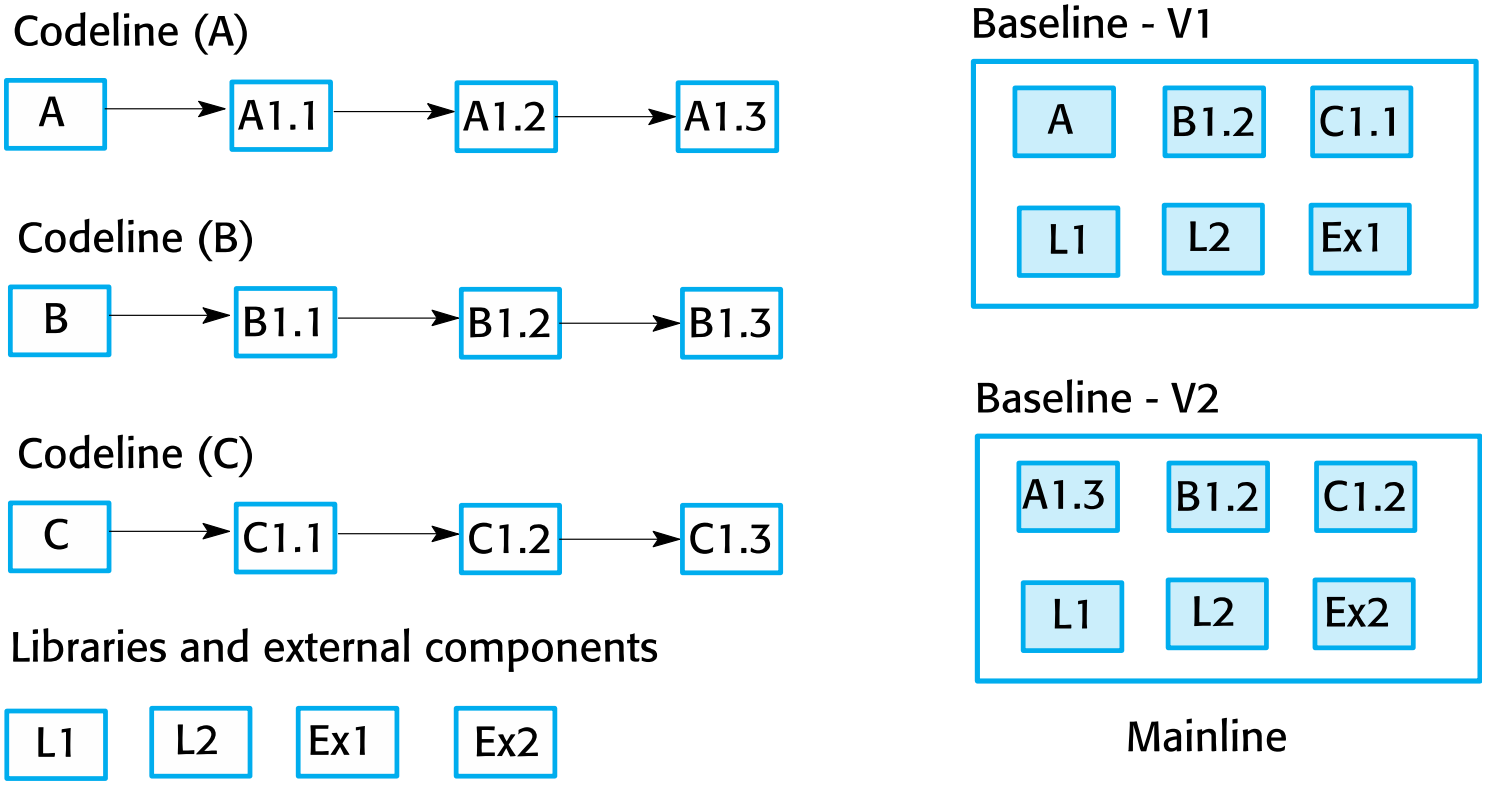
Codelines and baselines

- 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.
- 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.

Baselines

- 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



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 systems

- Version and release identification
  - Allow different versions of the same component to be managed
- Change history recording
  - Keep records of the changes that have been made to create a new version of a component from an earlier version
- Support for independent development
  - Different developers can work on the same component at the same time
- Project support
  - VC can support the development of several projects, which share components. It is usually possible to check in and out all of the files associated with a project
- Storage management
  - Ensure duplicate copies of identical files are not maintained

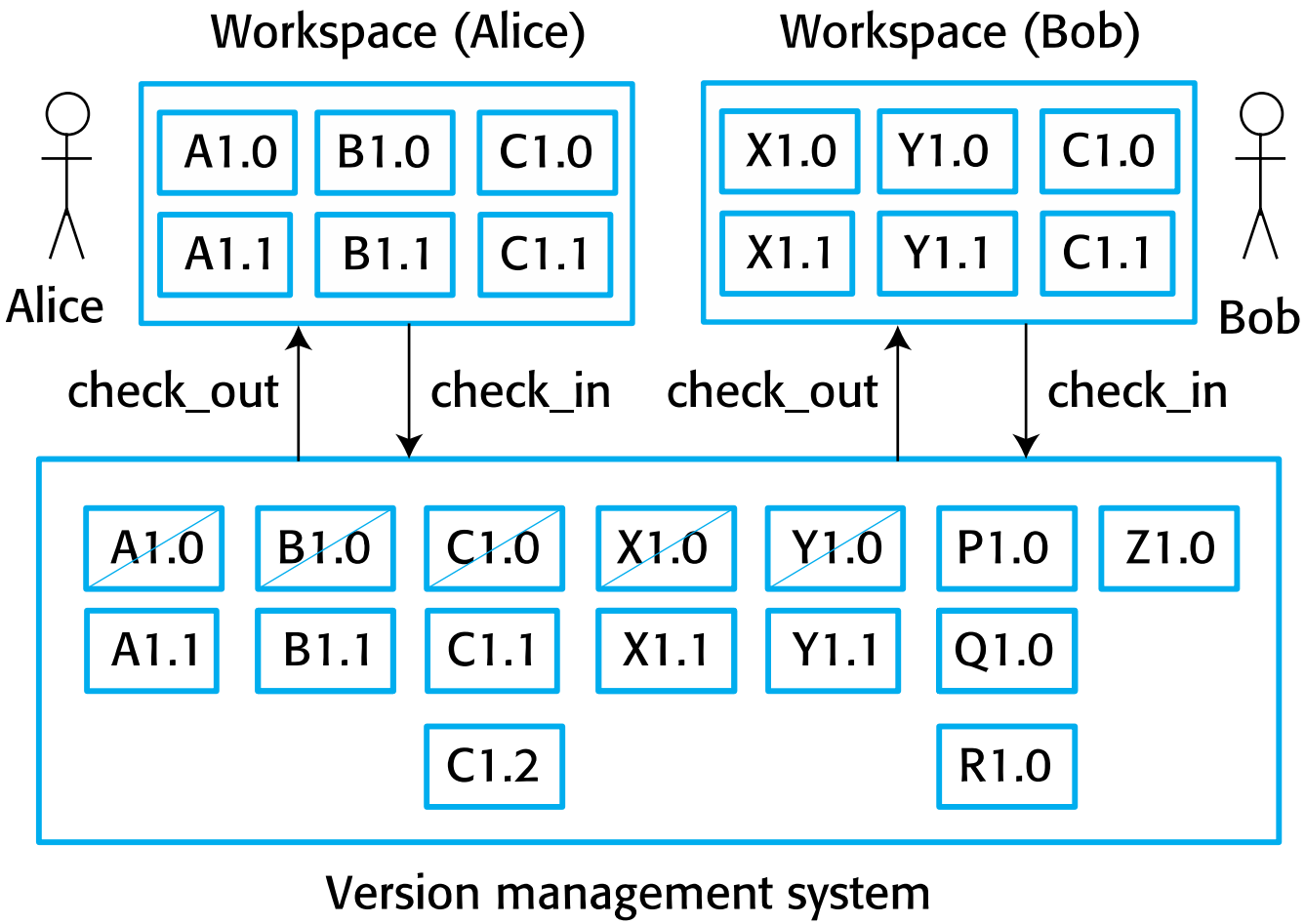
Public repository and private workspaces

- 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 repository into their workspace and work on these copies.
- When they have finished their changes, the changed components are returned (**checked-in**) to the repository.

Centralized version control

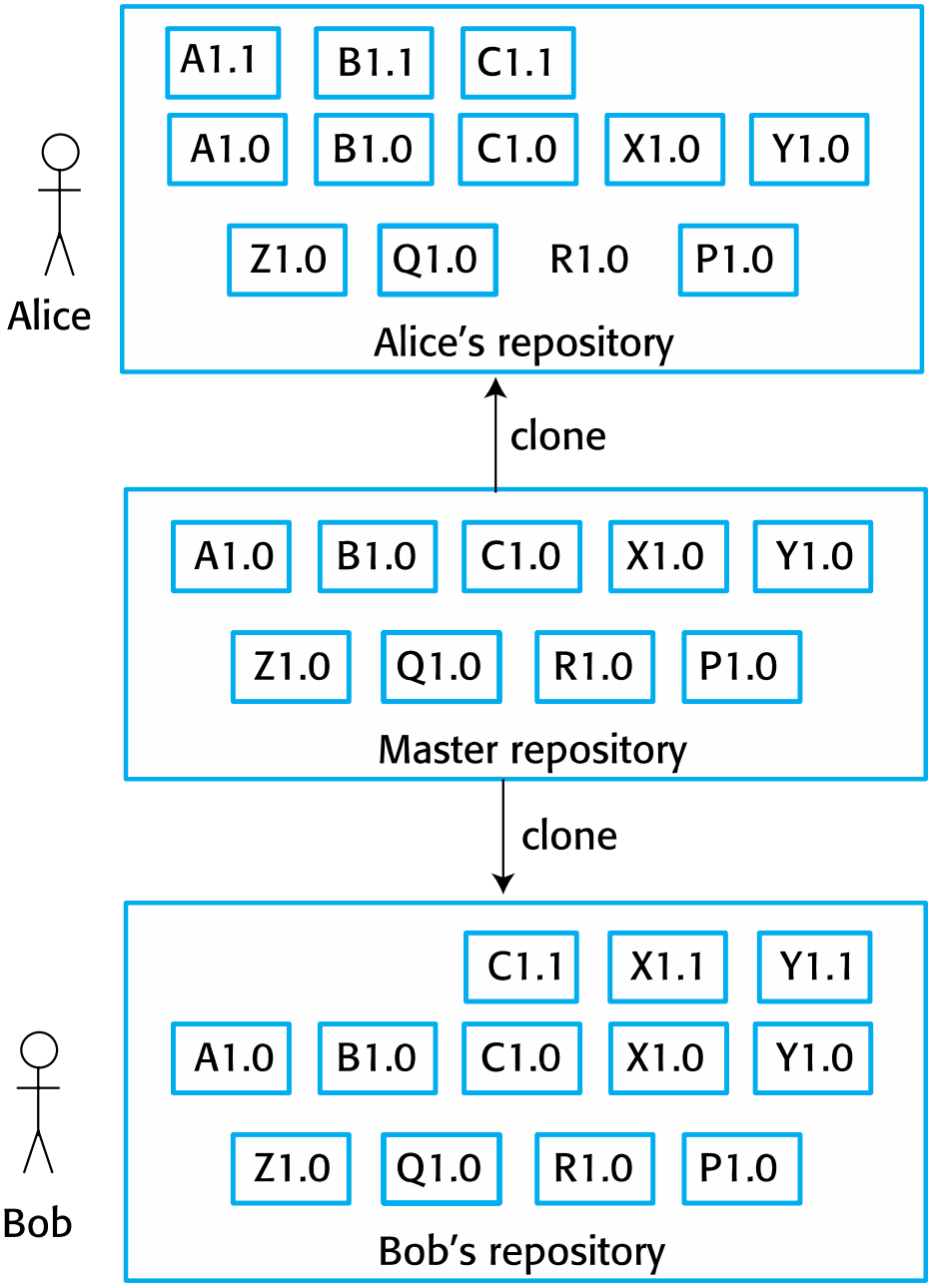
- Developers **check out** **components** or **directories of components** from the project repository into their private workspace and work on these copies in their private workspace.
- When their changes are complete, they **check-in** the components back to the repository.
- If several people are 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 (Centralized version control)



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 (Distributed version control)

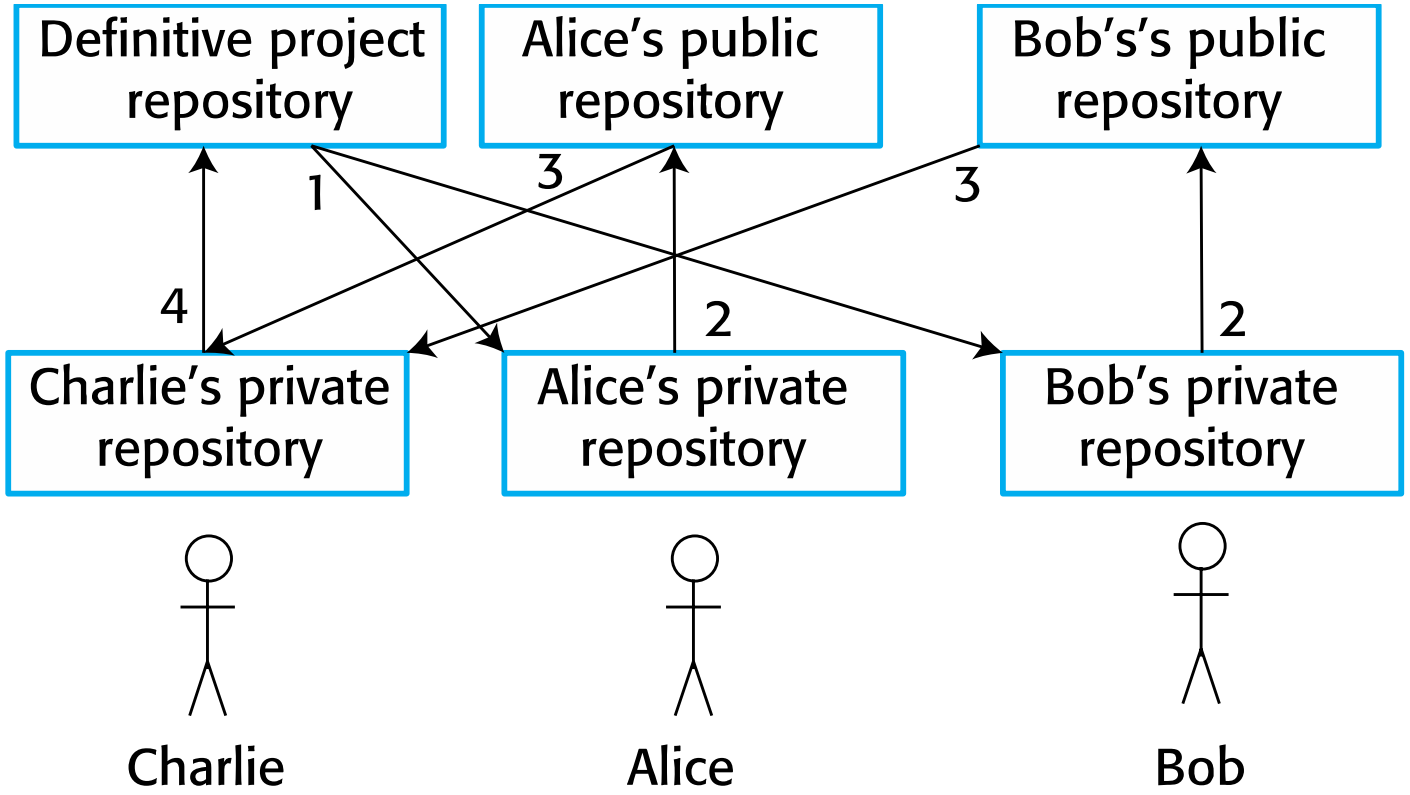
Benefits of distributed version control

- 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.

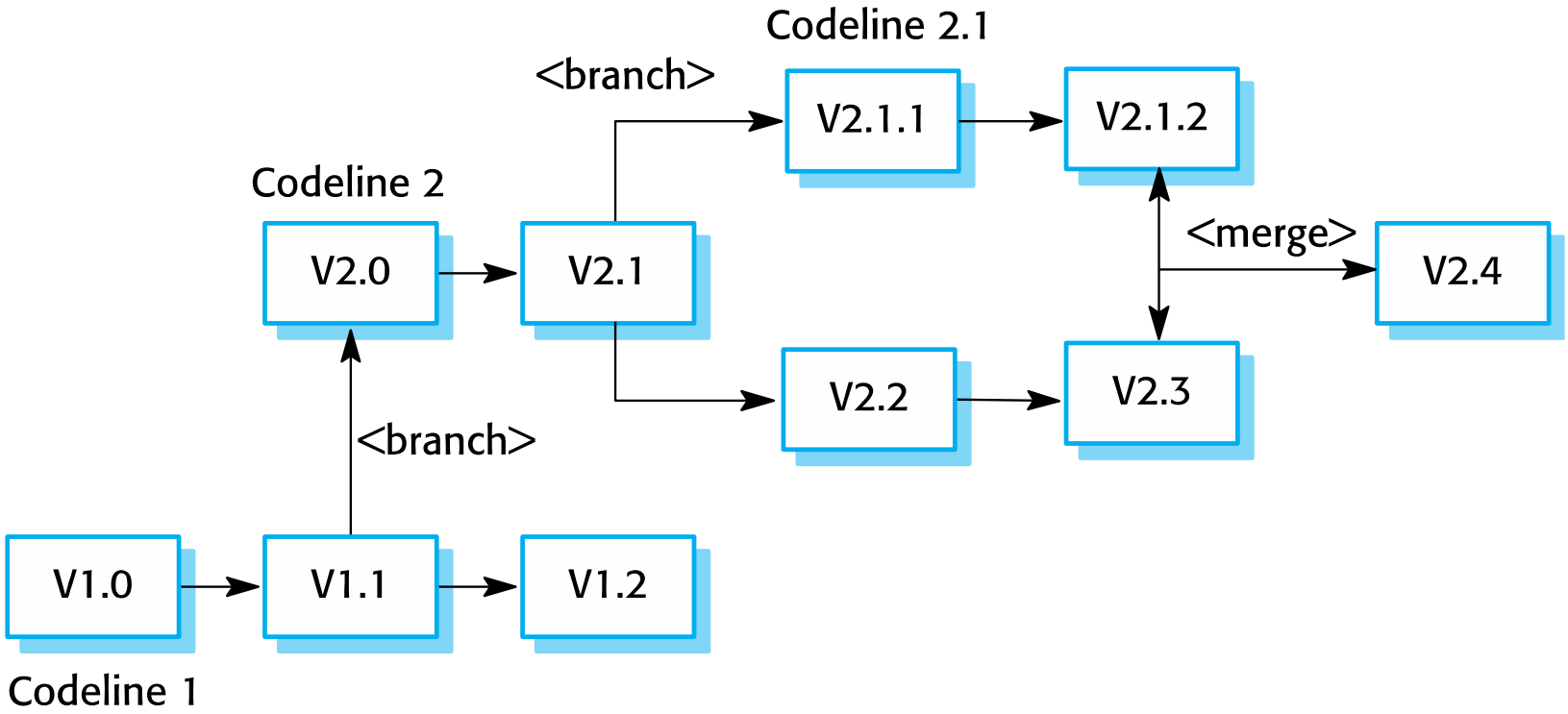
Open-source development



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 (**branches**) 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.

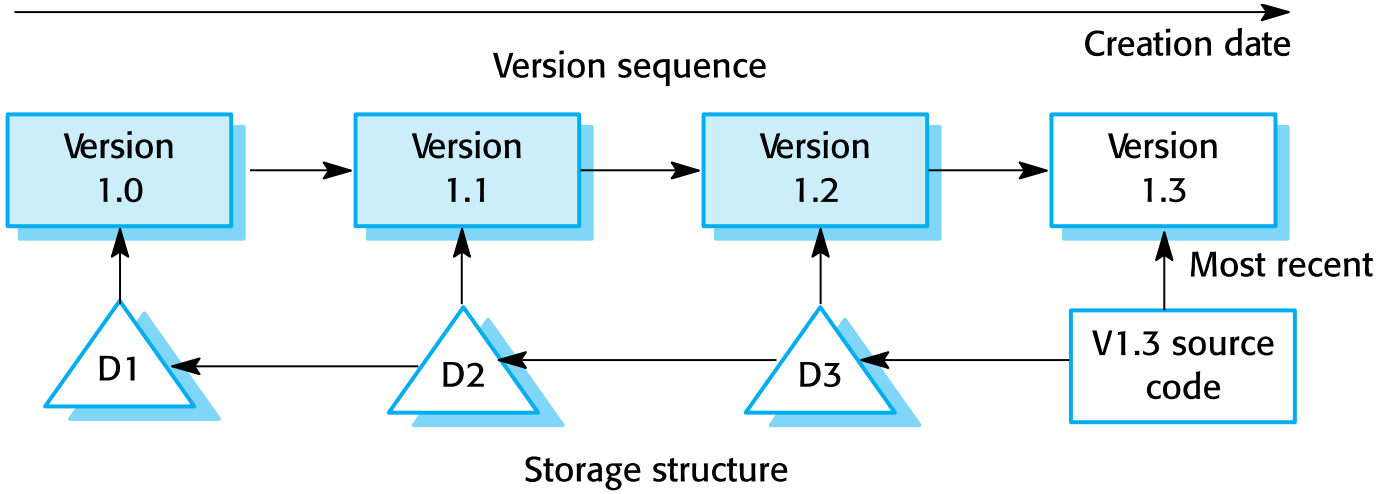
Branching and merging



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.
- One **disadvantages** is that it can take a long time to apply all of the deltas

Storage management using deltas



Storage management in Git

- 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.
- Deltas are used within packfiles to further reduce their size

System building

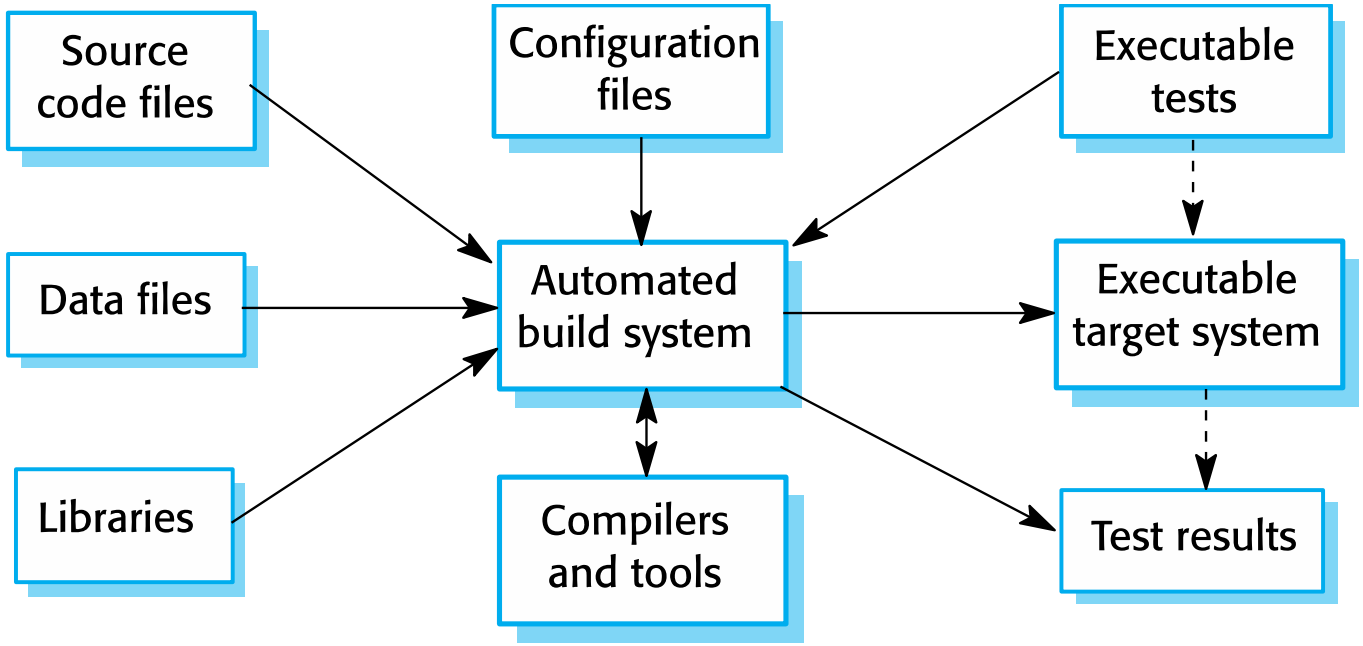
System building

- 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 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 platforms

- The development system, which 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.
- The build server, which is 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.
- The target environment, which is the platform on which the system executes.

System building



Build system functionality

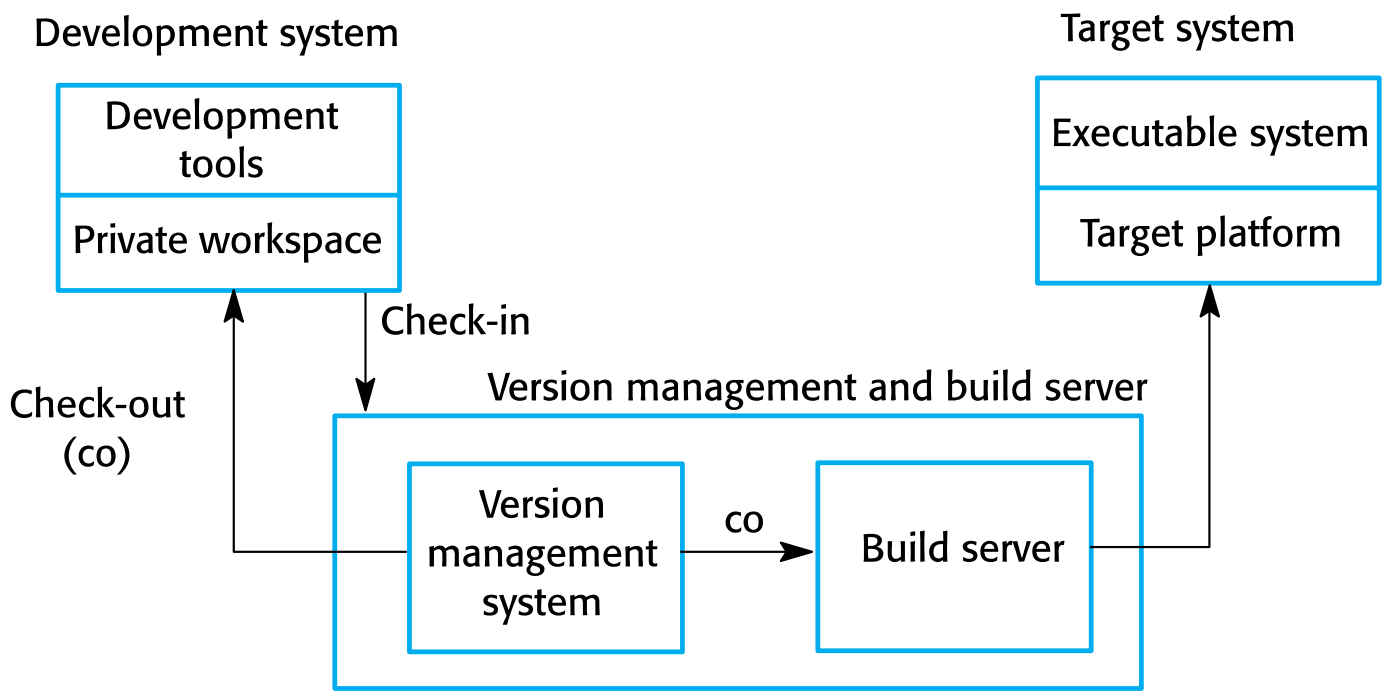
✧ Tools for system integration and building

- Build script generation
  - Makefile, ant file, maven file, Gradle file
    - may also include some static code analysis tools, such as CheckStyle, FindBugs, and PMD
- Version management system integration
- Minimal re-compilation
- Executable system creation
  - Link compiled object code files, libraries, configuration files
- Test automation
  - JUnit
- Reporting
  - Report the results regrading the build and test
- Documentation generation
  - Generate release notes about the build and system help pages

System platforms

- Building is a complex process, which is potentially error-prone, as three different platforms may be involved
  - The development system, which includes development tools such as compilers, source code editors, etc.
  - The build server, which is used to build definitive, executable versions of the system. This server maintains the definitive versions of a system.
  - The target environment, which is 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)
    - For large systems, the target environment may include databases and other application systems that cannot be installed on development machines

Development, build, and target platforms



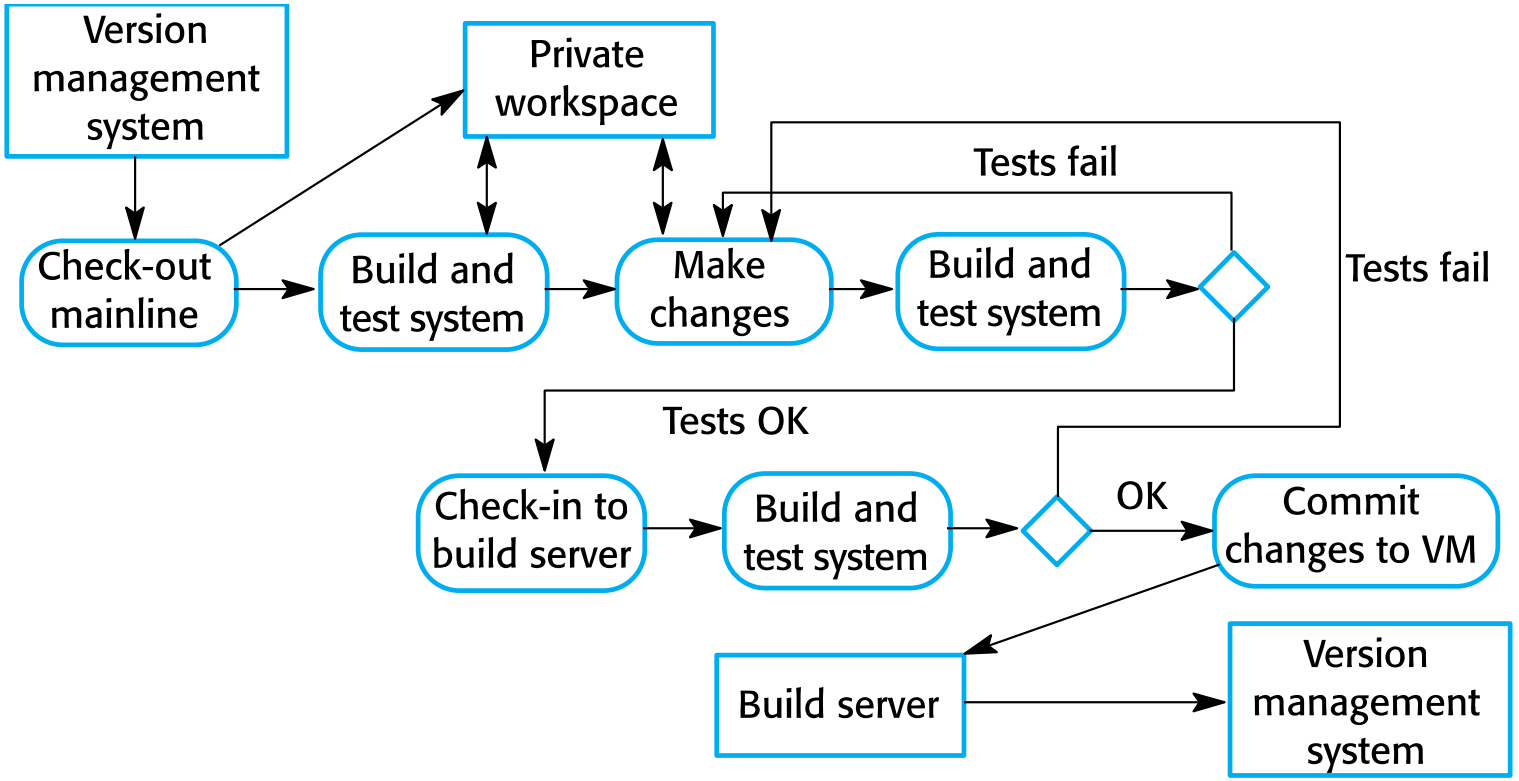
Agile building (Frequent build and automated testing are used to discover software problems)

- Check out the mainline system from the version management system into the developer's private workspace.
- Build the system and run automated tests to ensure that the built system passes all tests. If not, the build is broken and you should inform whoever checked in the last baseline system. They are responsible for repairing the problem.
- Make the changes to the system components.
- Build the system in the private workspace and rerun system tests. If the tests fail, continue editing.

Agile building (Frequent build and automated testing are used to discover software problems)

- Once the system has passed its tests, check it into the build system but **do not commit it** as a new system baseline.
- Build the system on the build server and run the tests. You need to do this in case others have modified components since you checked out the system. If this is the case, check out the components that have failed and edit these so that tests pass on your private workspace.
- If the system passes its tests on the build system, then **commit the changes** you have made as a new baseline in the system mainline.

Continuous integration



Pros and cons of continuous integration

✧ Pros

- The advantage of continuous integration is that it allows problems caused by **the interactions between different developers** to be discovered and repaired as soon as possible.
- The most recent system in the mainline is the **definitive working system**.

✧ Cons (can employ **daily build** instead of continuous integration)

- If the system is very large, it may take a long time to build and test, especially if integration with other application systems is involved.
- If the development platform is different from the target platform, it may not be possible to run system tests in the developer's private workspace.

Daily building

- 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.

Minimizing recompilation

- Tools to support system building are usually designed to minimize the amount of compilation that is required.
- They do this by checking if a compiled version of a component is available. If so, there is no need to recompile that component.
- A unique signature identifies each source and object code version and is changed when the source code is edited.
- By comparing the signatures on the source and object code files, it is possible to decide if the source code was used to generate the object code component.

File identification

✧ Modification timestamps

- The signature on the source code file is the time and date when that file was modified. If the source code file of a component has been modified **after** the related object code file, then the system assumes that recompilation to create a new object code file is necessary.

✧ Source code checksums

- The signature on the source code file is a checksum calculated from data in the file. A checksum function calculates a unique number using the source text as input. If you change the source code (even by 1 character), this will generate a different checksum. You can therefore be confident that source code files with different checksums are actually different.

Timestamps vs checksums

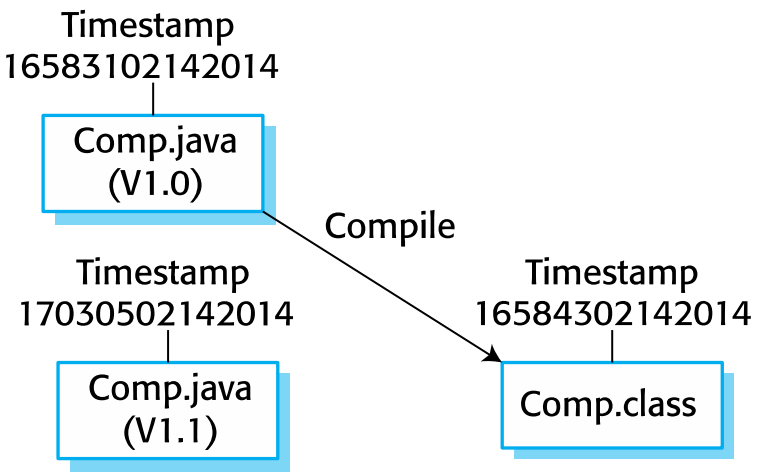
✧ Timestamps

- Because source and object files are linked by name rather than an explicit source file signature, it is not usually possible to build different versions of a source code component into the same directory at the same time, as these would generate object files with the same name. (i.e., **only most recently compiled version would be available**)

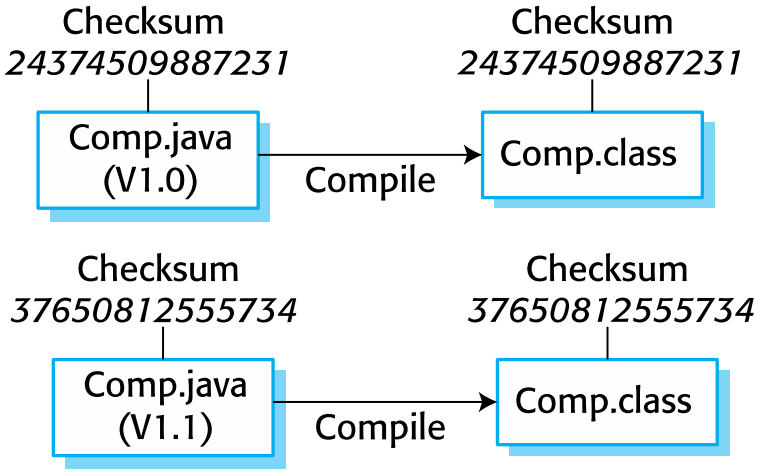
✧ Checksums

- When you recompile a component, it does not **overwrite** the object code, as would normally be the case when the timestamp is used. Rather, it generates a new object code file and tags it with the source code signature. Parallel compilation is possible and different versions of a component may be compiled at the same time. (i.e., **allow many different versions of object code of a component to be maintained at the same time**)

Linking source and object code



Time-based identification

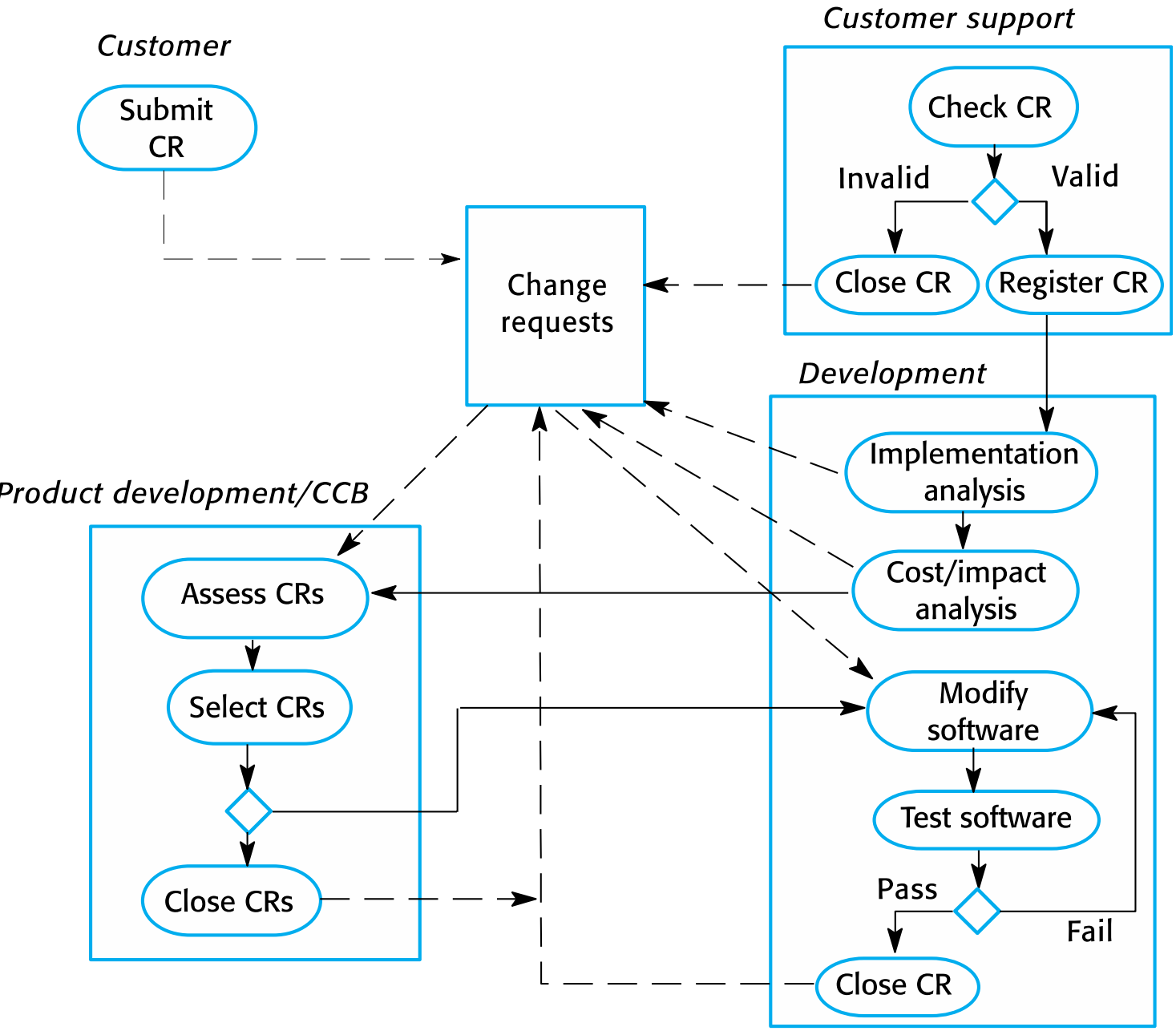


Checksum-based identification

Change management

Change management (change is **a fact of life** for large software systems)

- 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.
  - This process should come into effect when the software is handed over for release to customer or for deployment within an organization



The change management process

Change management process

- The change management process can be varied depending on
  - the types of software, such as a custom system, a product line, or an off-the-shelf product;
  - the size of company – small companies use a less formal process than large companies
- Tools to support change management may be relatively simple issue or bug tracking systems or software that is integrated with a configuration management package for large-scale systems, such as IBM Rational Clearcase
- The change management process is initiated when a system stakeholder completes and submits a change request
  - Can be a bug report or a request for additional functionality to be added; Some companies handle bug report and new requirements **separately**
  - Change requests may be submitted using a change request from (**CRF**)

A partially completed change request form used in a large complex system engineering project (a)

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

The **degree of formality** in the **CRF** varies depending on the **size and type of organization** that is developing the system

A partially completed change request form used in a large complex system engineering project (b)

Change Request Form

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:

Change management process

- After a change request has been submitted, it is checked to ensure that it is valid
- For **valid change requests**, the next stage of the change management process is change assessment and costing
  - **Identify all of the components affected** by the changes;
  - **Access the required changes** to the system modules;
  - **Estimate the cost** of making the changes
- Following change assessment and costing analysis, a separate group (called **change control board (CCB)** or **product development group**) decides if it is cost-effective for the business to make the change to the software
  - The CCB or product development group considers the impact of the change from a **strategic** or **organizational** rather than a **technical** point of view. It decides whether the change in question is economically justified, and it prioritizes accepted changes for implementation

Factors in change analysis

- The factors that influence the decision on whether or not to implement a change include
  - The consequences of not making the change
    - E.g., if the change is associated with a system failure, take into account the seriousness of that failure
  - The benefits of the change
  - The number of users affected by the change
    - Only a few users are affected by the change (low priority)
    - The majority of users have to adapt to the change (inadvisable)
  - The costs of making the change
    - The change affects many system components and/or takes a lot of time to implement
  - The product release cycle

Change management for software products

- Change management for software products, rather than customer systems specially developed for a certain customer, are handled in a different way
  - Customer is not directly involved in decisions about system evolution
  - Change requests for the products come from the
    - Customer support team (submit change requests associated with bugs that have been discovered and reported by customers)
    - The company marketing team (suggest changes that should be included to make it easier to sell a new version of system)
    - Developer themselves (good ideas about new features that can be added to the system)

Change management and agile methods

- In some agile methods, customers are directly involved in change management.
- The propose a change to the requirements and work with the team to assess its impact and decide whether the change should take priority over the features planned for the next increment of the system.
- Changes that involve the software improvement are decided by the programmers working on the system.
- Refactoring, where the software is continually improved, is not seen as an overhead but as a necessary part of the development process.

Derivation history (a record of the changes made to each component)


```
// 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
```

Release management

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

- As well as the 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.

Release planning

- Preparing and distributing a system release for mass-market products is an expensive process
- 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.

Factors influencing system release planning

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.

Release creation

- **Release creation** is the process of creating the collection of files and documentation that include all components of the system release. The process involves several steps:
  - 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 military systems and 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.
  - It may be necessary to keep copies of older operating systems and other support software because they may still be in operational use
  - Older operating systems can run in a virtual machine
- You should also record the versions of the operating system, libraries, compilers and other tools used to build the software.
- Accordingly, you may have to store copies of the platform software and the tools used to create the system in the version control system, along with the source code of the target system

Software as a service

- When planning the installation of new system releases, you cannot assume that customers will always install new system release
  - New releases of system cannot, therefore, rely on the installation of previous releases
    - Some customers may go directly from release 1 to release 3, skipping release 2
- Delivering **software as a service (SaaS)** reduces the problems of release management.
- It simplifies both release management and system installation for customers.
- The software developer is responsible for replacing the existing release of a system with a new release and this is made available to all customers at the same time.

Key points

- **Configuration management** is the management of an evolving software system. When maintaining a system, a CM team is put in place to ensure that changes are incorporated into the system in a controlled way and that records are maintained with details of the changes that have been implemented.
- The main configuration management processes are concerned with version management, system building, change management, and release management.
- Version management involves keeping track of the different versions of software components as changes are made to them.

Key points

- System building is the process of assembling system components into an executable program to run on a target computer system.
- Software should be frequently rebuilt and tested immediately after a new version has been built. This makes it easier to detect bugs and problems that have been introduced since the last build.
- Change management involves assessing proposals for changes from system customers and other stakeholders and deciding if it is cost-effective to implement these in a new version of a system.
- **System releases** include executable code, data files, configuration files and documentation. **Release management** involves making decisions on system release dates, preparing all information for distribution and documenting each system release.