ICT-5405

PROJECT MANAGEMENT AND QUALITY ASSURANCE

05 Configuration Management

Why Software Configuration Management ?

- The problem:

  Multiple people have to work on software that is changing

  More than one version of the software has to be supported:

  - **Released systems**
  - **Custom configured systems (different functionality)**
  - **System(s) under development**

  Software must run on different machines and operating systems

⇨ *Need for coordination*

- **Software Configuration Management**
  - **manages evolving software systems**
  - **controls the costs involved in making changes to a system**

What is Software Configuration Management?

A set of management disciplines within the software engineering process to develop a baseline.

- Configuration management (CM) is concerned with the policies, processes, and tools for managing changing software systems.
- You need to manage evolving systems because it is easy to lose track of what changes and component versions have been incorporated into each system version.
- Versions implement proposals for change, corrections of faults, and adaptations for different hardware and operating systems.
- Several versions may be under development and in use at the same time.

If you don't have effective **configuration management** procedures in place, you may waste effort modifying the wrong version of a system, delivering the wrong version of a system to customers, or forgetting where the software source code for a particular version of the system or component is stored

Software Configuration Management is a Project Function

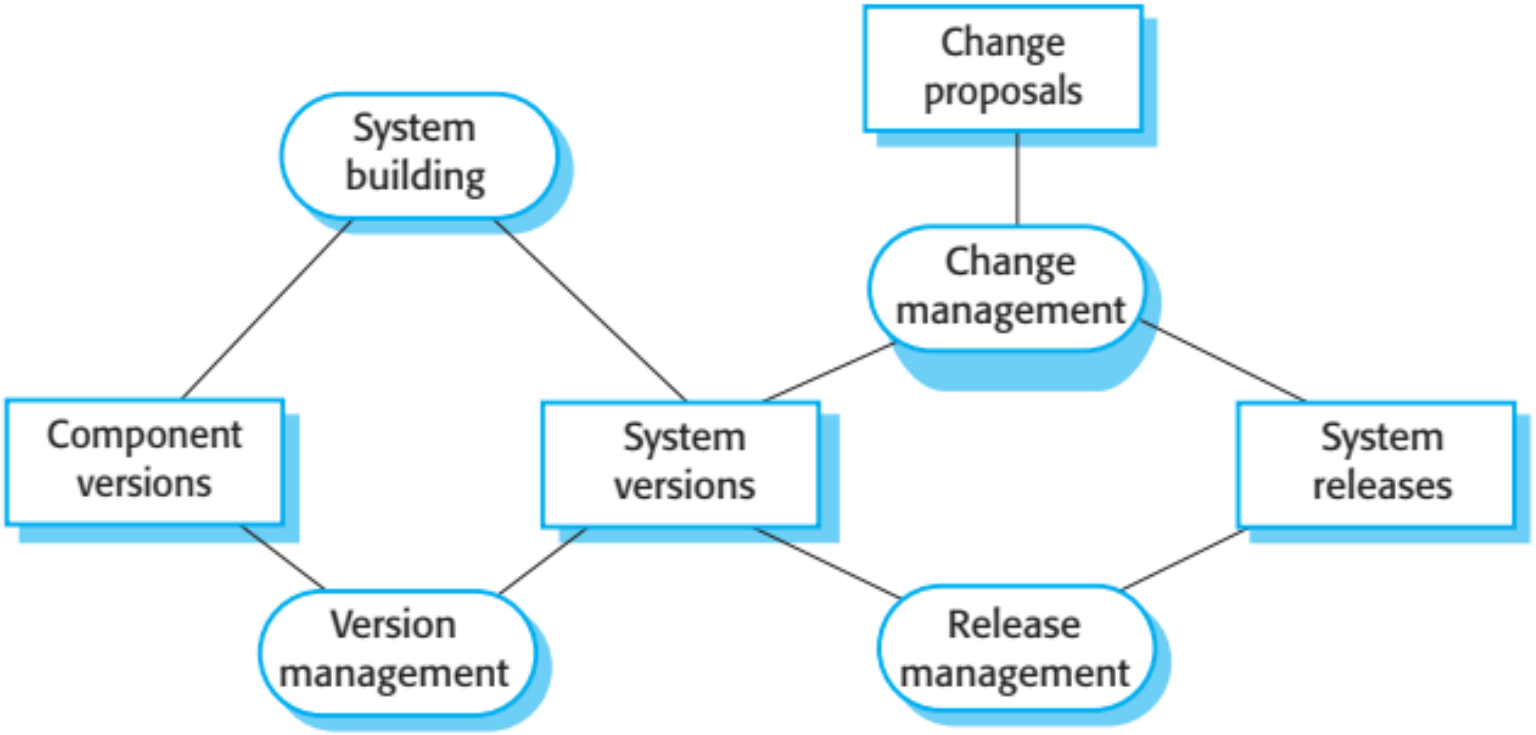
- SCM is a Project Function (as defined in the SPMP) with the goal to make technical and managerial activities more effective.
- Software Configuration Management can be administered in several ways:
  - A single software configuration management team for the whole organization
  - A separate configuration management team for each project
  - Software Configuration Management distributed among the project members
  - Mixture of all of the above

Configuration Management Activities

The configuration management of a software system product involves four closely related activities

- **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.
- **System building** This is the process of assembling program components, data, and libraries, then compiling and linking these to create an executable system.
- **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.
- **Release management** This involves preparing software for external release and keeping track of the system versions that have been released for customer use.

Configuration Management Activities



CM in Agile Development

- Agile development, where components and systems are changed several times a day, is impossible without using **CM tools**.
- The definitive versions of components are held in a shared project repository, and developers copy them into their own workspace.
- They make changes to the code and 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. This makes the modified components available to other team members.

Configuration Management Roles

- Configuration Manager

  Responsible for identifying configuration items. The configuration manager can also be responsible for defining the procedures for creating promotions and releases
- Change control board member

  Responsible for approving or rejecting change requests
- Developer

  Creates promotions triggered by change requests or the normal activities of development. The developer checks in changes and resolves conflicts
- Auditor

  Responsible for the selection and evaluation of promotions for release and for ensuring the consistency and completeness of this release

Terminology

- We will define the following terms
  - Configuration Item
  - Baseline
  - SCM Directories
  - Version
  - Revision
  - Release

Terminology: Configuration Item

"An aggregation of hardware, software, or both, that is designated for configuration management and treated as a single entity in the configuration management process."

- Software configuration items are not only program code segments but all type of documents according to development, e.g
  - all type of code files
  - drivers for tests
  - analysis or design documents
  - user or developer manuals
  - system configurations (e.g. version of compiler used)
- In some systems, not only software but also hardware configuration items (CPUs, bus speed frequencies) exist!

Finding Configuration Items

- Large projects typically produce thousands of entities (files, documents, data ...) which must be uniquely identified.
- Any entity managed in the software engineering process can potentially be brought under configuration management control
- But not every entity needs to be under configuration management control all the time.
- Two Issues:
  - What: Selection of Configuration Items
    - What should be under configuration control?
  - When: When do you start to place entities under configuration control?

Terminology: Version

- The initial release or re-release of a configuration item associated with a complete compilation or recompilation of the item. Different versions have different functionality.
- Versions always have a unique identifier, which is often composed of the configuration item name plus a version number.

Terminology: Baseline

"*A specification or product that has been formally reviewed and agreed to by responsible management, that thereafter serves as the basis for further development, and can be changed only through formal change control procedures*." -- The IEEE (IEEE Std. No. 610.12-1990)

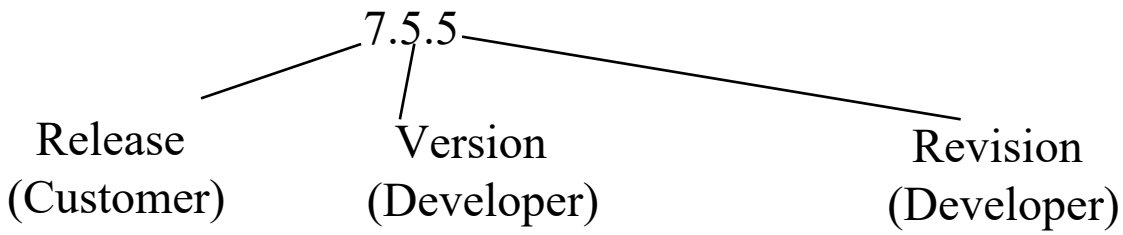
- A baseline is a milestone in the development of software that is marked by the delivery of one or more software configuration items and the approval of these SCIs that is obtained through a formal technical review

Examples:

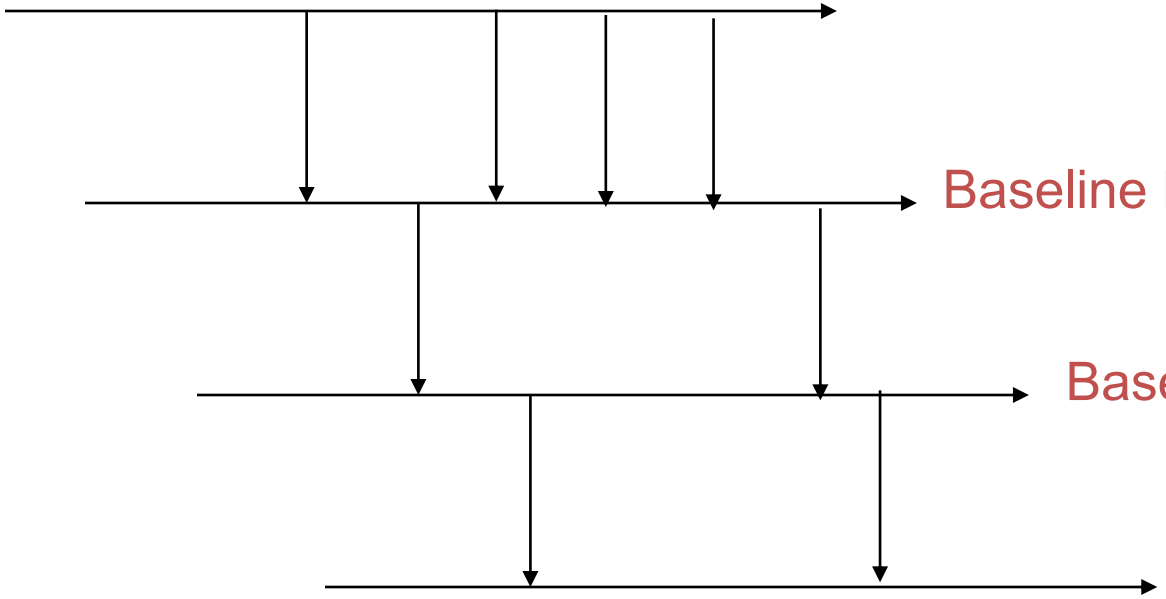
Baseline A: All the API have completely been defined; the bodies of the methods are empty.
Baseline B: All data access methods are implemented and tested.
Baseline C: The GUI is implemented.

More on Baselines

- As systems are developed, a series of baselines is developed, usually after a review (analysis review, design review, code review, system testing, client acceptance, ...)
  - *Developmental baseline* (RAD, SDD, Integration Test, ...)
    - Goal: Coordinate engineering activities.
  - *Functional baseline* (first prototype, alpha release, beta release)
    - Goal: Get first customer experiences with functional system.
  - *Product baseline* (product)
    - Goal: Coordinate sales and customer support.
- Many naming scheme for baselines exist (1.0, 6.01a, ...)
- A 3 digit scheme is quite common:



Baselines in SCM



Baseline A (developmental)

Baseline B (functional, first prototype)

Baseline C (functional, beta test)

Official Release

How do we manage changes in the baselines?

Time

Change management

- Change management is the handling of change requests
  A change request leads to the creation of a new release
- **General change process**
  - ❖ The change is requested (this can be done by anyone including users and developers)
  - ❖ The change request is assessed against project goals
  - ❖ Following the assessment, the change is accepted or rejected
  - ❖ If it is accepted, the change is assigned to a developer and implemented
  - ❖ The implemented change is audited.
- The complexity of the change management process varies with the project. Small projects can perform change requests informally and fast while complex projects require detailed change request forms and the official approval by one more managers.

Factors 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

Change management and agile methods

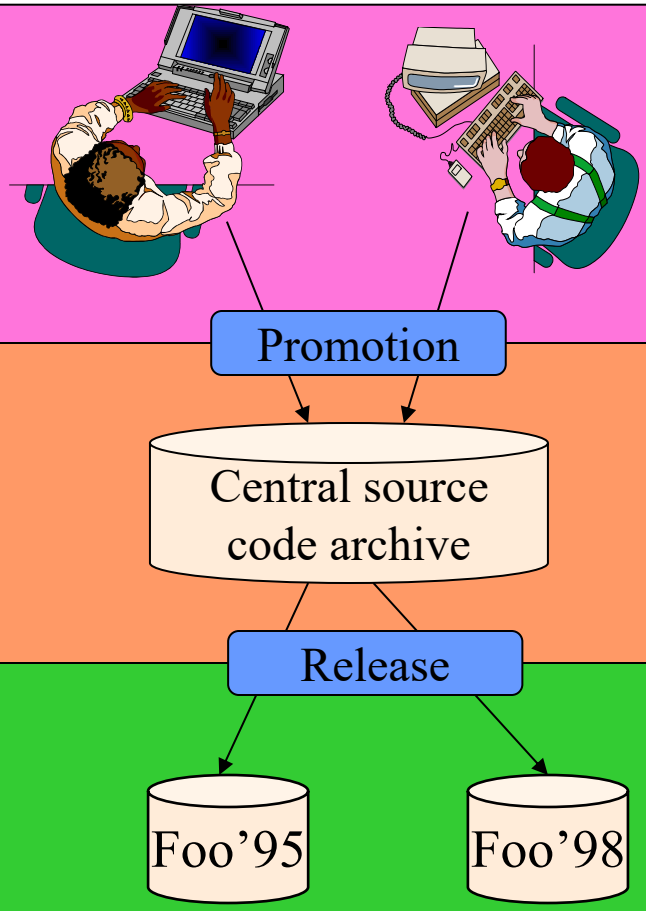
- In the most popular agile methods, customers are directly involved in change management.
- They propose a change to the requirements and work with the team to assess its impact and decide whether the change should take priority over the features planned for the next increment of the system.
- Changes to improve 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.

Terminology: SCM Directories

- Programmer's Directory (IEEE: Dynamic Library)
  - Library for holding newly created or modified software entities.
  - The programmer's workspace is controlled by the programmer only.
- Master Directory (IEEE: Controlled Library)
  - Manages the current baseline(s) and for controlling changes made to them.
  - Entry is controlled, usually after verification.
  - Changes must be authorized.
- Software Repository (IEEE: Static Library)
  - Archive for the various baselines released for general use.
  - Copies of these baselines may be made available to requesting organizations.

Standard SCM Directories

- Programmer's Directory
  (IEEE Std: "Dynamic Library")
  Completely under control of one programmer.
- Master Directory
  (IEEE Std: "Controlled Library")
  Central directory of all promotions.
- Software Repository
  (IEEE Std: "Static Library")
  Externally released baselines.



Terminology: Version vs. Revision vs. Release

- Version:
  An *initial* release or re-release of a configuration item associated with a *complete compilation* or recompilation of the item. Different versions have different functionality.

Question: Is Windows98 a new version or a new revision compared to Windows95 ?

- Revision:
  Change to a version that corrects only errors in the design/code, but does not affect the documented functionality.

- Release:
  The *formal distribution* of an approved version.

CM terminology

Term	Explanation
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.
Configuration 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.
Version	An instance of a configuration item that differs, in some way, from other instances of that item. Versions always have a unique identifier, which is often composed of the configuration item name plus a version number.
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 should always be possible to recreate a baseline from its constituent components.
Codeline	A codeline is a set of versions of a software component and other configuration items on which that component depends.

CM terminology

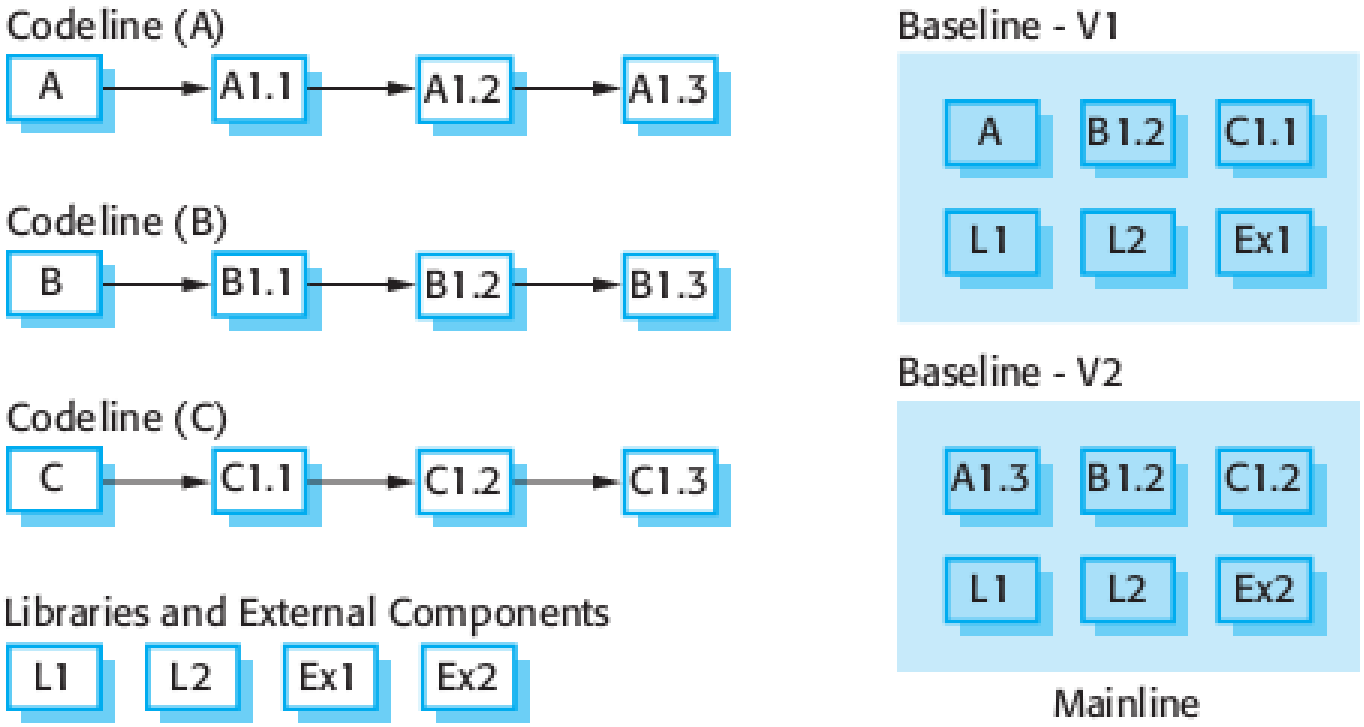
Term	Explanation
Mainline	A sequence of baselines representing different versions of a system.
Release	A version of a system that has been released to customers (or other users in an organization) for use.
Workspace	A private work area where software can be modified without affecting other developers who may be using or modifying that software.
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.
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.
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 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.


Codeline (A)
A
A1.1
A1.2
A1.3
Codeline (B)
B
B1.1
B1.2
B1.3
Codeline (C)
C
C1.1
C1.2
C1.3
Libraries and External Components
L1
L2
Ex1
Ex2
Baseline - V1
A
B1.2
C1.1
L1
L2
Ex1
Baseline - V2
A1.3
B1.2
C1.2
L1
L2
Ex2
Mainline

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.

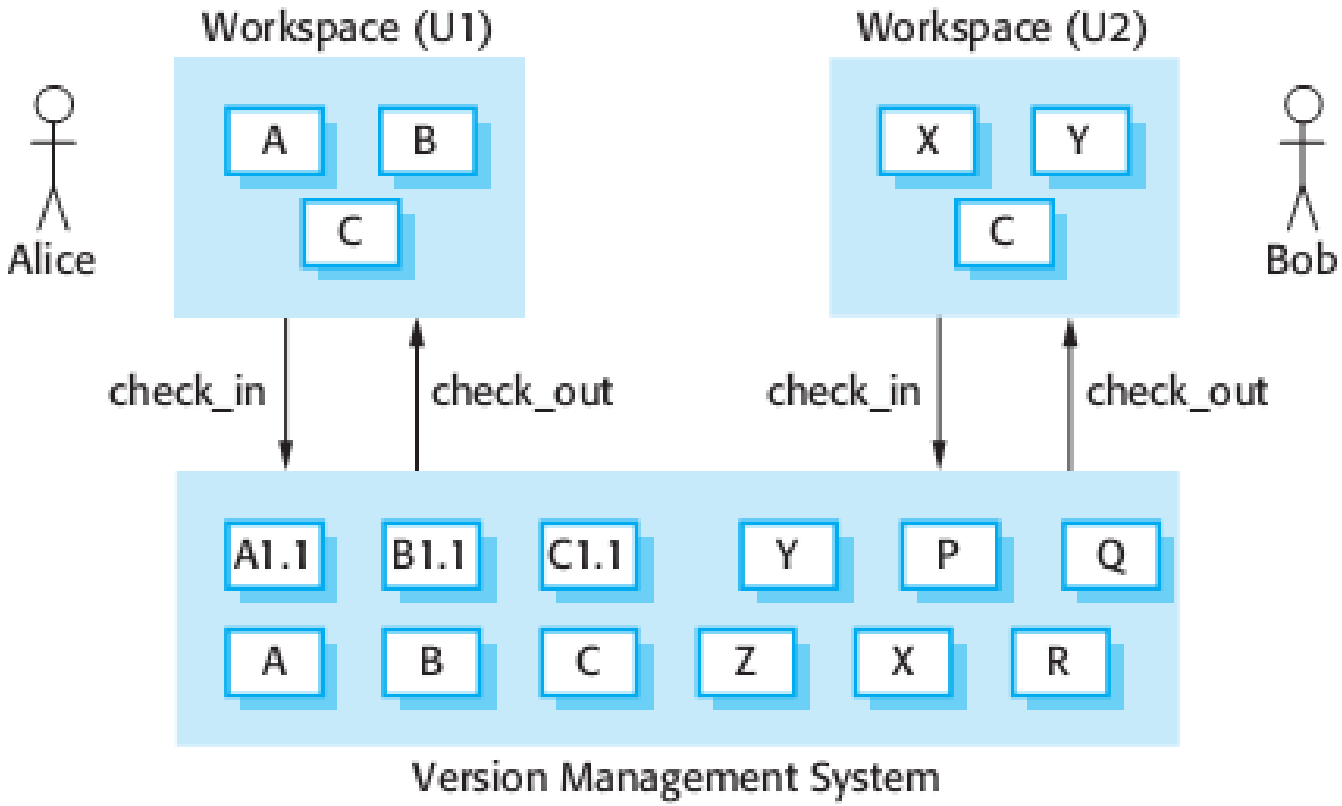
  For example, a product line may be instantiated so that there are individual system versions for different customers. You may have to recreate the version delivered to a specific customer if, for example, that customer reports bugs in their system that have to be repaired.

Key Features of Version management systems

Version management is the process of keeping track of different versions of software components and the systems in which these components are used

- Version and release identification
  Managed versions are assigned identifiers when they are submitted to the system.
- Storage management
  Rather than maintain separate copies of all versions of a component, the version control system may use efficient mechanisms to ensure that duplicate copies of identical files are not maintained. Where there are only small differences between files, the VC system may store these differences rather than maintain multiple copies of files. A specific version may be auto-matically re-created by applying the differences to a master version.
- Change history recording
  All of the changes made to the code of a system or component are recorded and listed.
- Independent development
  The version management system keeps track of components that have been checked out for editing and ensures that changes made to a component by different developers do not interfere.
- **Project support**
  A version management system may support the development of several projects, which share components.

Check-in and check-out from a version repository

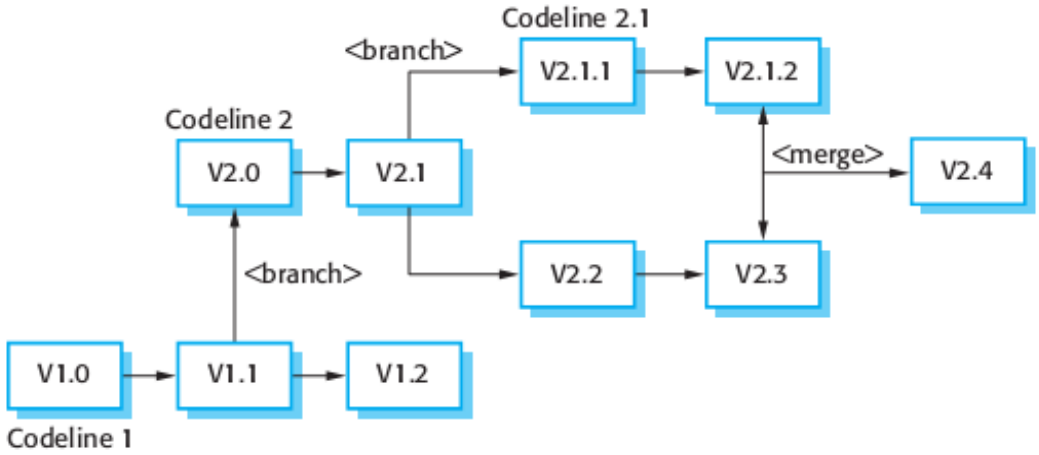


Codeline branches

- 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

- This is also shown in Figure 25.8, where component versions 2.1.2 and 2.3 are merged to create version 2.4. If the changes made involve completely different parts of the code, the component versions may be merged automatically by the version control system by combining the code
- changes. This is the normal mode of operation when new features have been added. These code changes are merged into the master copy of the system. However, the changes made by different developers sometimes overlap. The changes may be incompatible and interfere with each other. In this case, a developer has to check for clashes and make changes to the components to resolve the incompatibilities between the different versions.



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.

System building

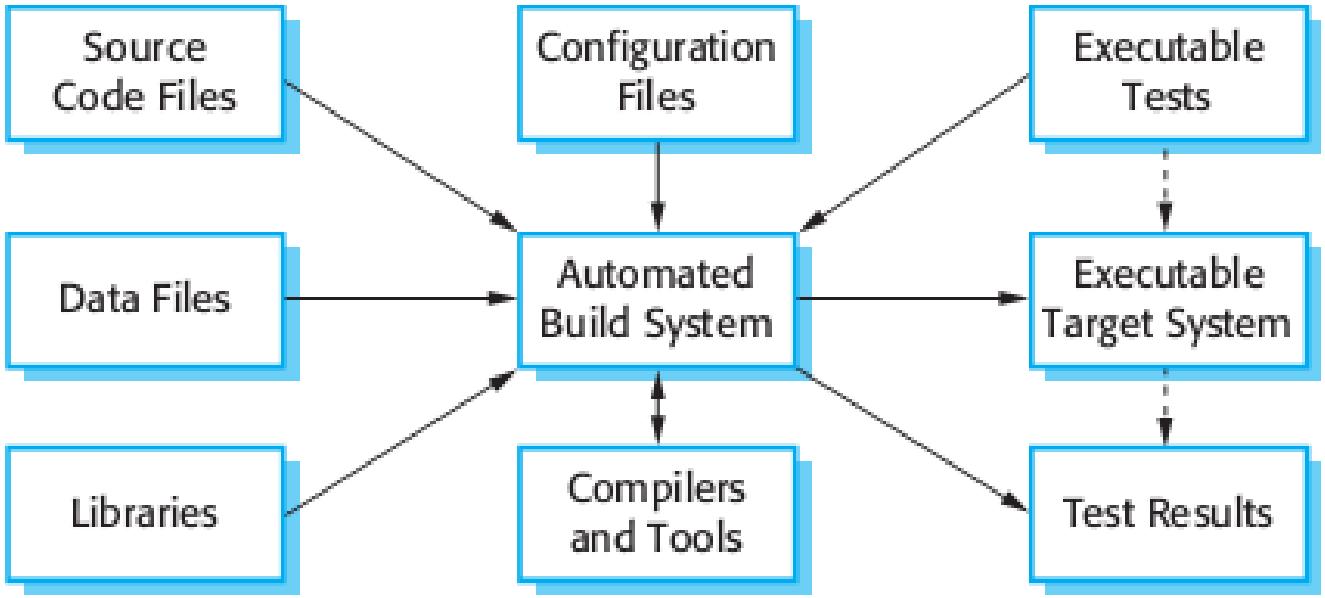


Figure :System building

Tools for system integration and building

1. **Build script generation** The build system should analyze the program that is being built, identify dependent components, and automatically generate a build script (configuration file). The system should also support the manual creation and editing of build scripts.
2. **Version control system integration** The build system should check out the required versions of components from the version control system.
3. **Minimal recompilation** The build system should work out what source code needs to be recompiled and set up compilations if required.

Tools for system integration and building

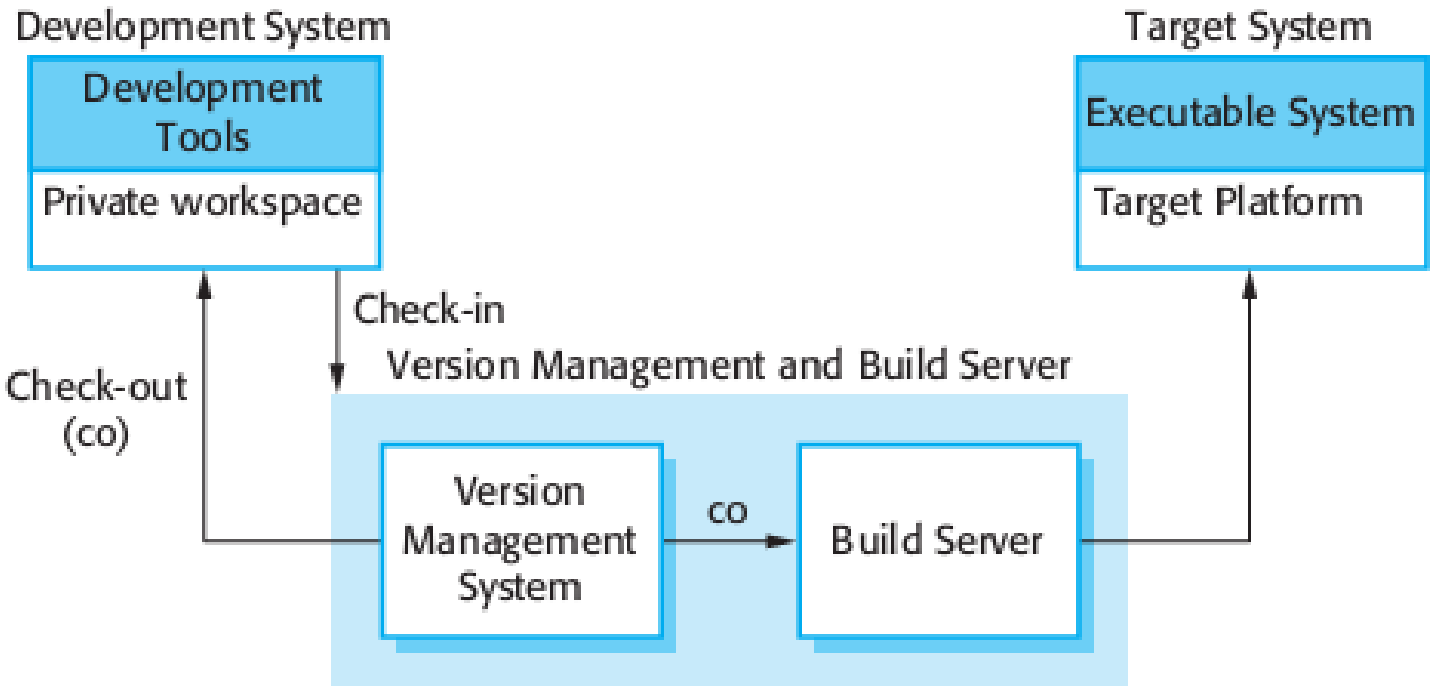
4. **Executable system creation** The build system should link the compiled object code files with each other and with other required files, such as libraries and configuration files, to create an executable system.
5. **Test automation** Some build systems can automatically run automated tests using test automation tools such as JUnit. These check that the build has not been “broken” by changes.
6. **Reporting** The build system should provide reports about the success or failure of the build and the tests that have been run.
7. **Documentation generation** The build system may be able to generate release notes about the build and system help pages.

Build platforms

Building is a complex process, which is potentially error-prone, as three different system platforms may be involved

- **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. This may be the same type of computer that is used for the development and build systems

Development, build, and target platforms



Agile building (Continuous Integration)

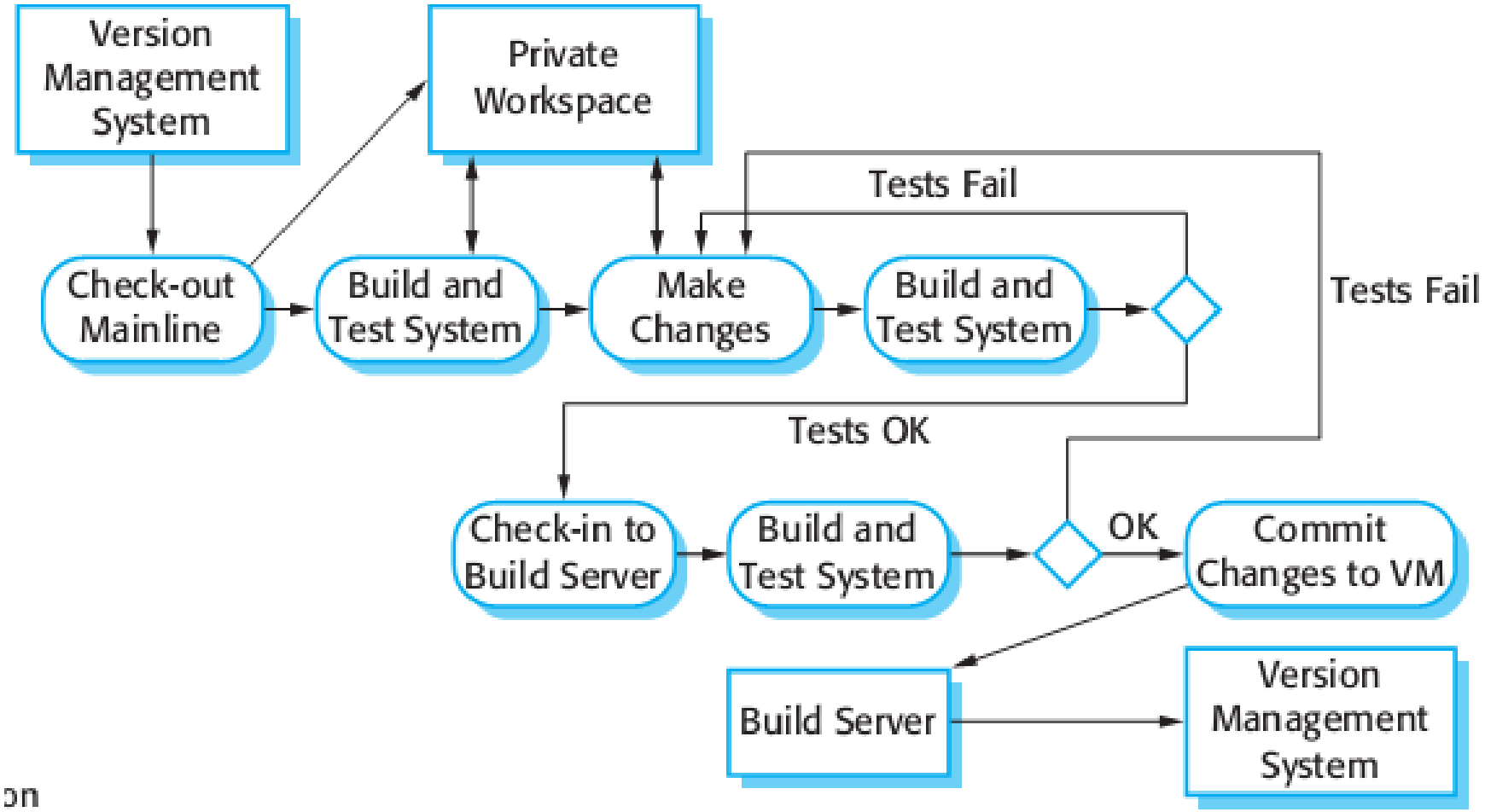
Agile methods recommend that very frequent system builds should be carried out, with automated testing used to discover software problems. Frequent builds are part of a process of continuous integration. **The steps in continuous integration** are:

1. Check out the mainline system from the version management system into the developer's private workspace.
2. 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.
3. Make the changes to the system components.
4. Build the system in the private workspace and rerun system tests. If the tests fail, continue editing.

Agile building (Continuous Integration)

5. Once the system has passed its tests, check it into the build system but do not commit it as a new system baseline.
6. 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.
7. 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



Daily building

For large systems or for systems where the execution platform is not the same as the development platform, continuous integration is usually impossible. In those circumstances, frequent system building is supported using a daily build system:

1. 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. Components may be incomplete but should provide some basic functionality that can be tested.
2. A new version of the system is built from these components by compiling and linking them to form a complete system.
3. This system is then delivered to the testing team, which carries out a set of predefined system tests.
4. 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.

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 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.

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. For marketing reasons, it may be necessary to include new features in a system so that users can be persuaded to upgrade from a previous release.
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 that affect the way in which many customers use the system, it may be necessary to correct them in a new system release. Minor system faults may be repaired by issuing patches, distributed over the Internet, which can be applied to the current release of the system.