Configuration Management

Lecture 1a – Part II

Agenda

- Configuration Management
- Version Management
- System Building
- Change Management
- Release Management

Configuration Management

Configuration Management (CM)

- Software systems are **constantly changing** during development and use.
  - **Bugs** are discovered and must be fixed.
  - System **requirements** change, and you must implement these changes in a new version of the system.
  - New versions of hardware and system **platforms** are released, and you must adapt your systems to work with them.
  - Competitors introduce new **features** in their system that you must **match**.

Configuration Management (CM)

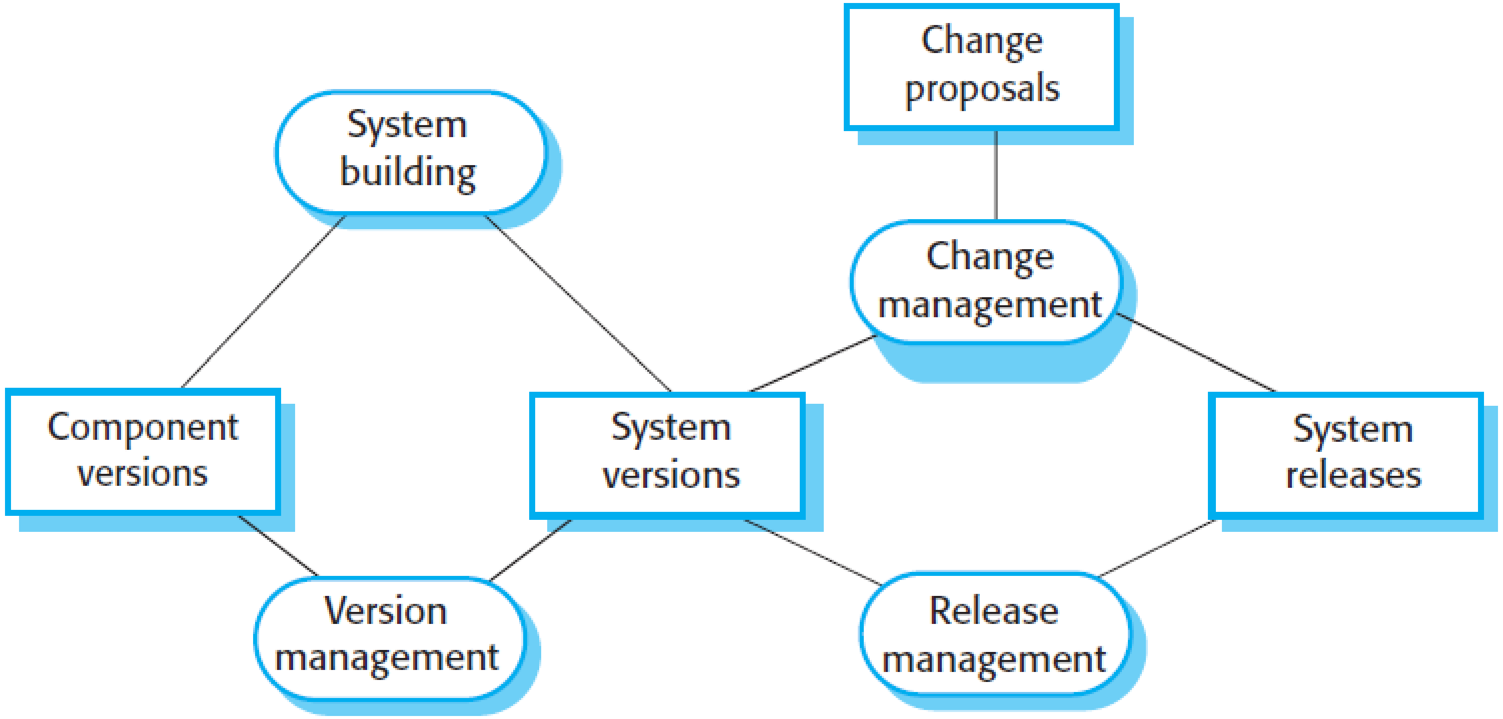
- Configuration Management is concerned with the **policies, processes, and tools for managing changing software systems**.
- As **changes** are made to the software, a **new version** of a system is created.
- A system can be **considered as a set of versions**, each of which may have to be maintained and managed.

Configuration Management

- The configuration management of a software system product involves **four** closely related **activities**:
    1. Version management
    2. System building
    3. Change management
    4. Release management

Configuration Management

- Configuration management activities:



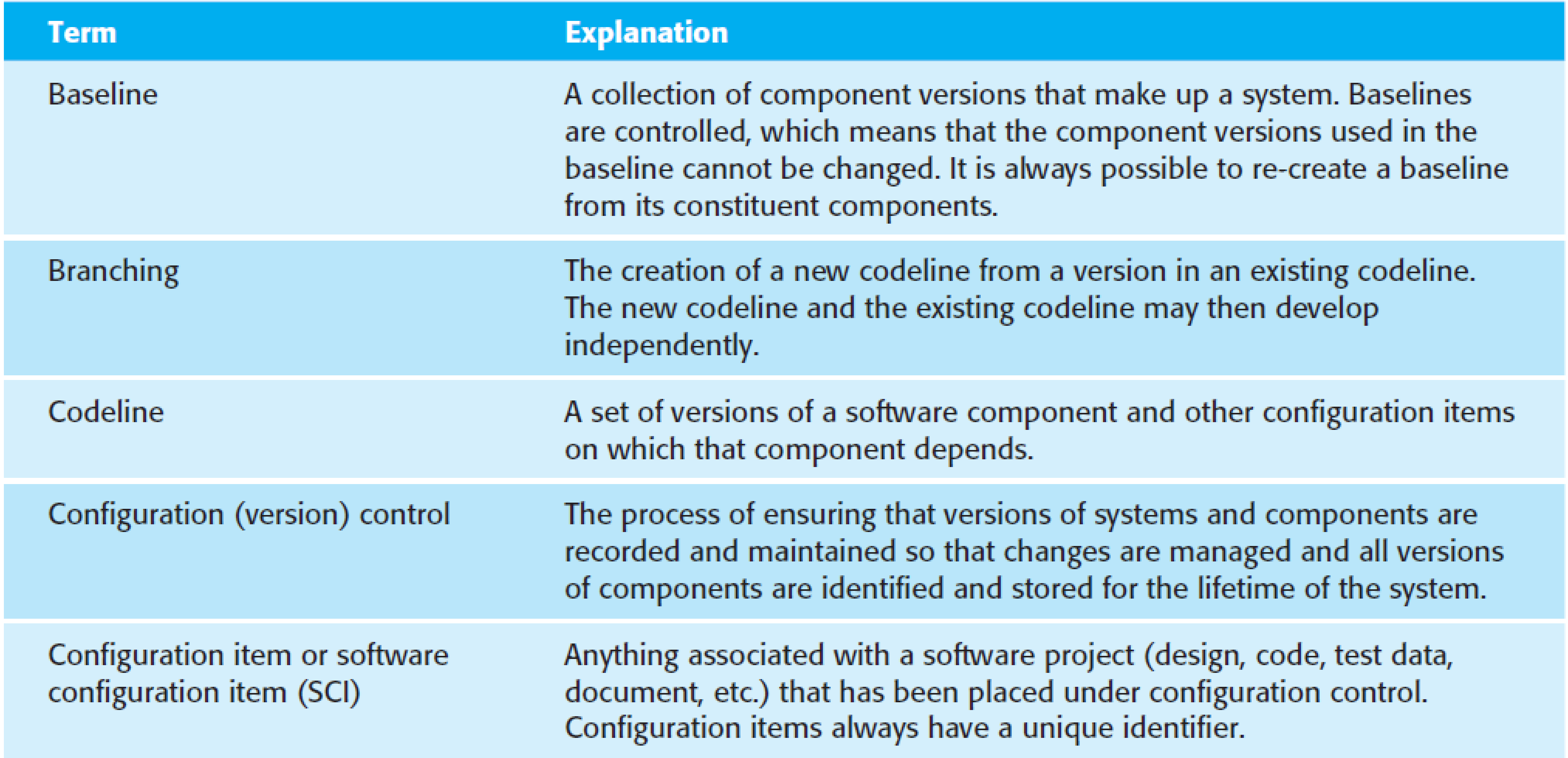
Configuration Management Activities

1. ***Version management:*** This involves keeping **track of the multiple versions** of system **components** and ensuring that changes made to components by different developers **do not interfere** with each other.
2. ***System building:*** This is the process of **assembling** program components, data, and libraries, then **compiling** and **linking** these to **create an executable system**.

Configuration Management Activities

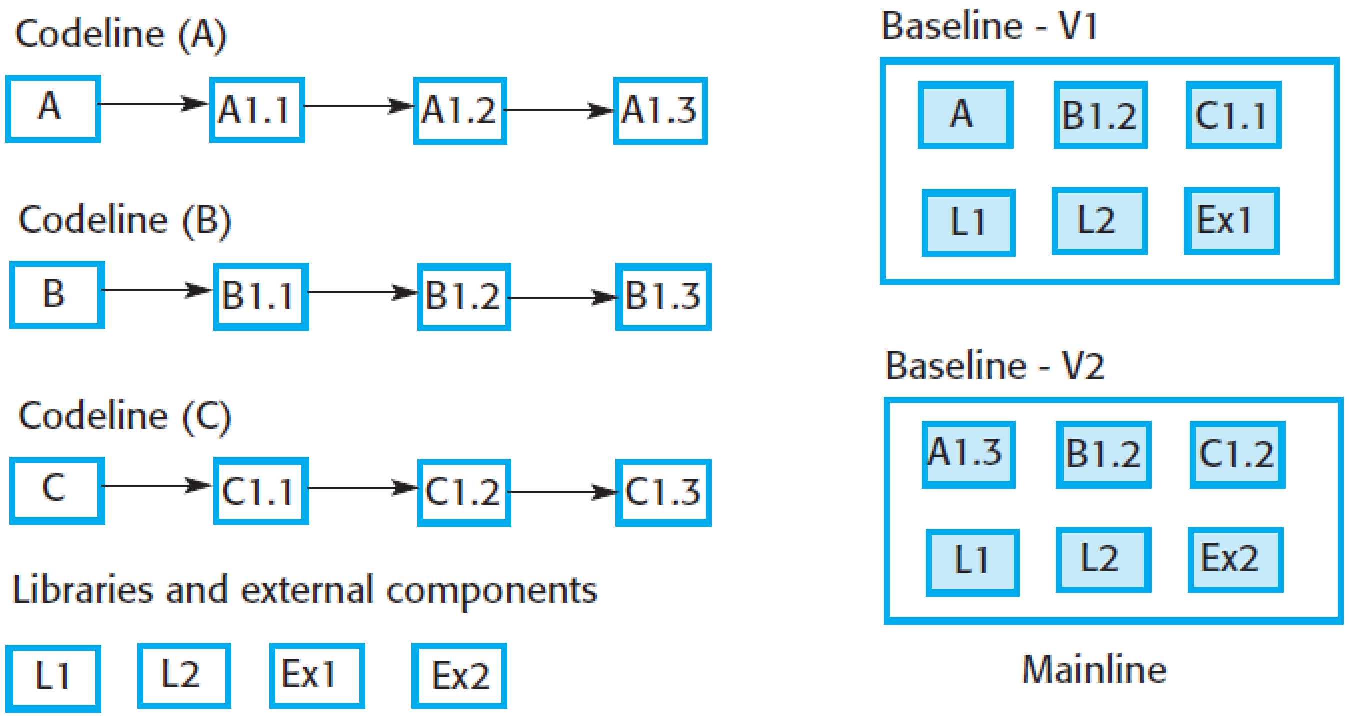
3. ***Change management:*** This involves keeping **track of requests for changes** to delivered software from customers and developers, working out the **costs** and **impact** of making these changes, and deciding **if and when** the changes should be implemented.
4. ***Release management:*** This involves preparing software for **external release** and keeping **track** of the **system versions** that have been released for customer use.

Configuration Management Terminology

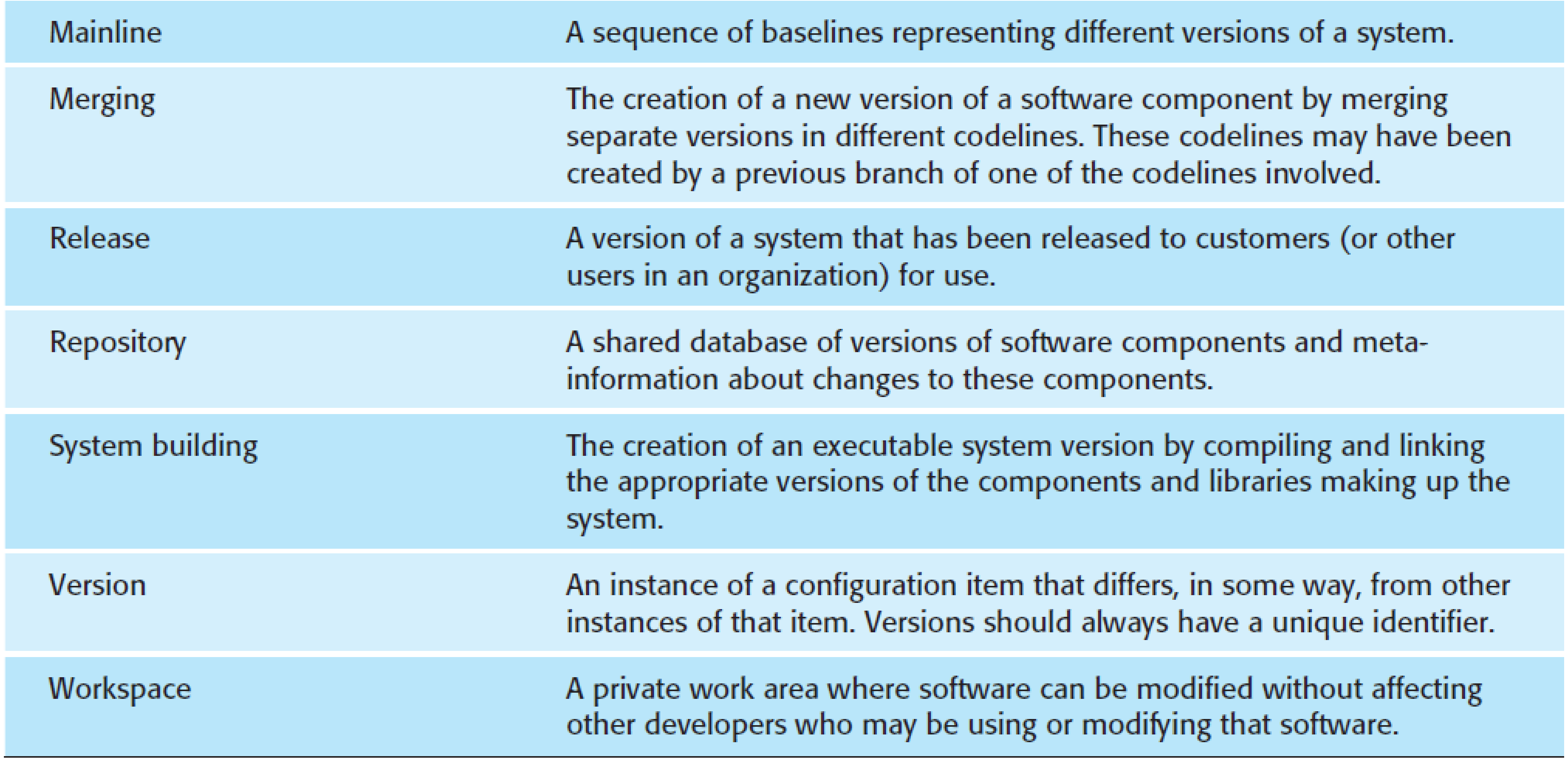


Term	Explanation
Baseline	A collection of component versions that make up a system. Baselines are controlled, which means that the component versions used in the baseline cannot be changed. It is always possible to re-create a baseline from its constituent components.
Branching	The creation of a new codeline from a version in an existing codeline. The new codeline and the existing codeline may then develop independently.
Codeline	A 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. Configuration items always have a unique identifier.

Codelines and Baselines



Configuration Management Terminology



Mainline	A sequence of baselines representing different versions of a system.
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 should 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 is the process of keeping **track** of different **versions** of software **components** 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**.
- In other words, version management is the process of managing codelines and baselines.
- **To support version management, we need Version Control (VC) systems.**

Features of Version Control (VC) Systems

- **Version and release identification**: managed versions of a component are assigned unique identifiers when they are submitted to the system.
- **Change history recording**: A version control system keeps records of the changes that have been made to create a new version of a component from an earlier version.
- **Independent development**: different developers may be working on the same component at the same time.

Features of Version Control (VC) Systems

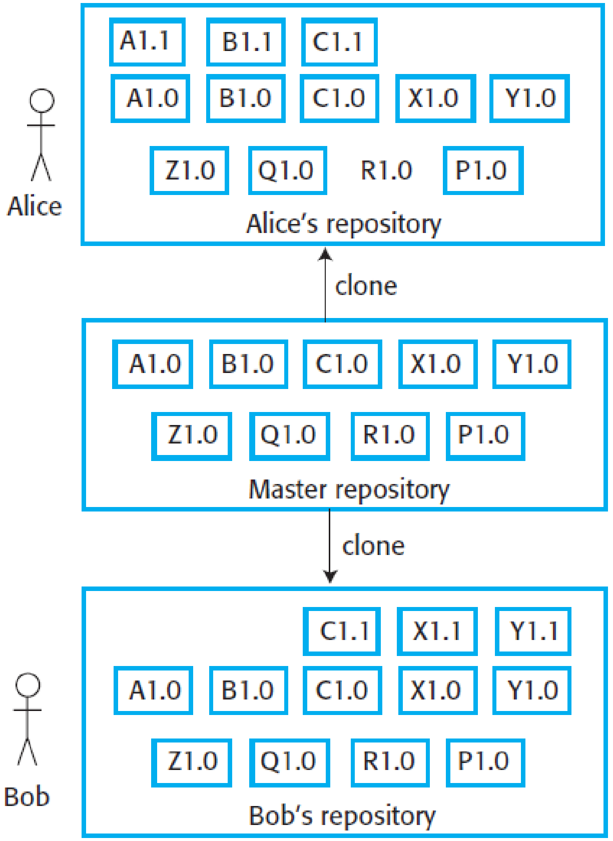
- **Project support**: A version control system may support the development of several projects, which share components.
- **Storage management:** Rather than maintain separate copies of all versions of a component, the version control system may use efficient mechanisms to ensure that duplicate copies of identical files are not maintained.

Types of Version Control Systems

- VC systems identify, store, and control access to the different versions of components. There are two types of modern version control systems:
  1. **Centralized systems**, where a **single** master **repository** maintains all versions of the software components that are being developed. Subversion is a widely used example of a centralized VM system.
  2. **Distributed systems**, where **multiple** versions of the component **repository** exist at the same time. Git, is a widely used example of a distributed VM system.

Distributed Version Control Systems

- Repository cloning

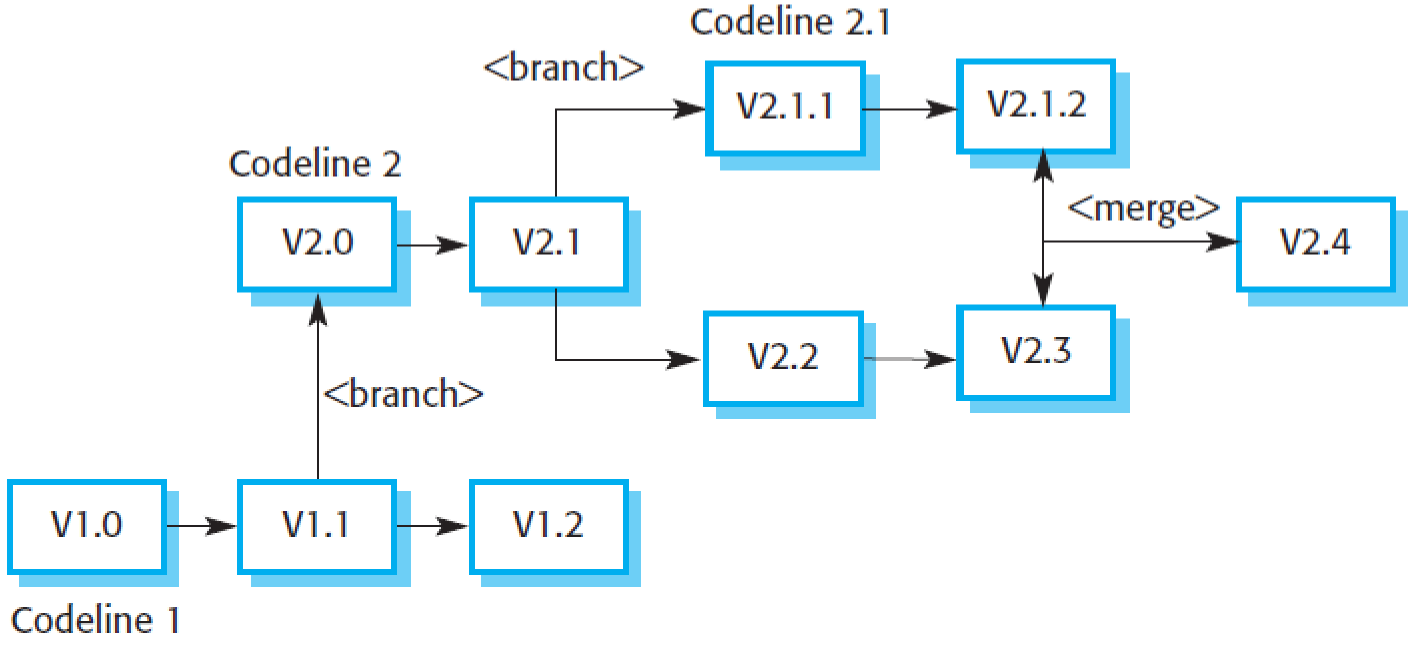


Advantages of Distributed over Centralised VC Systems

- It has a number of advantages:
  - 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 **offline 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 they have made.

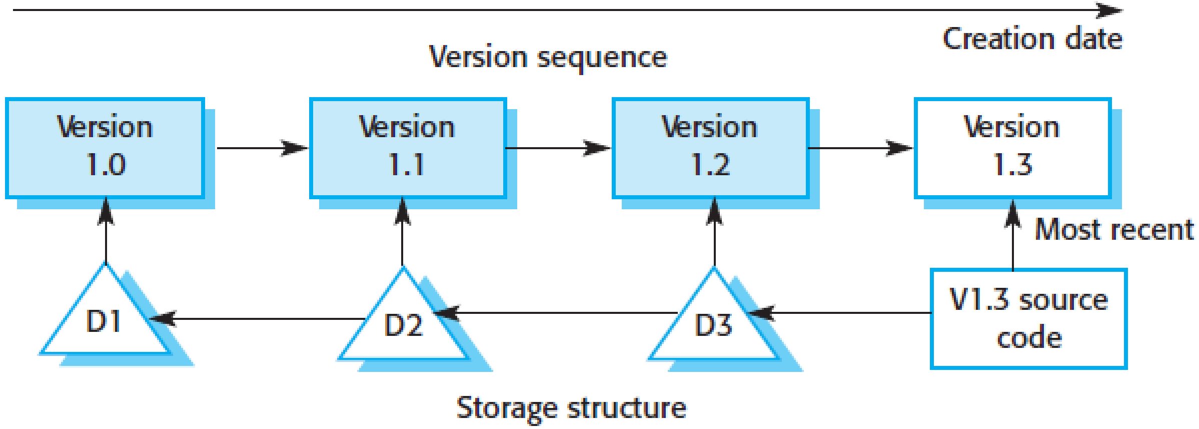
Branching and Merging in Distributed VC Systems

- Codelines may branch due to independent development of the same component.
- At some stage, it may be necessary to merge codeline branches to create a new version.



Storage Management in Distributed VC Systems

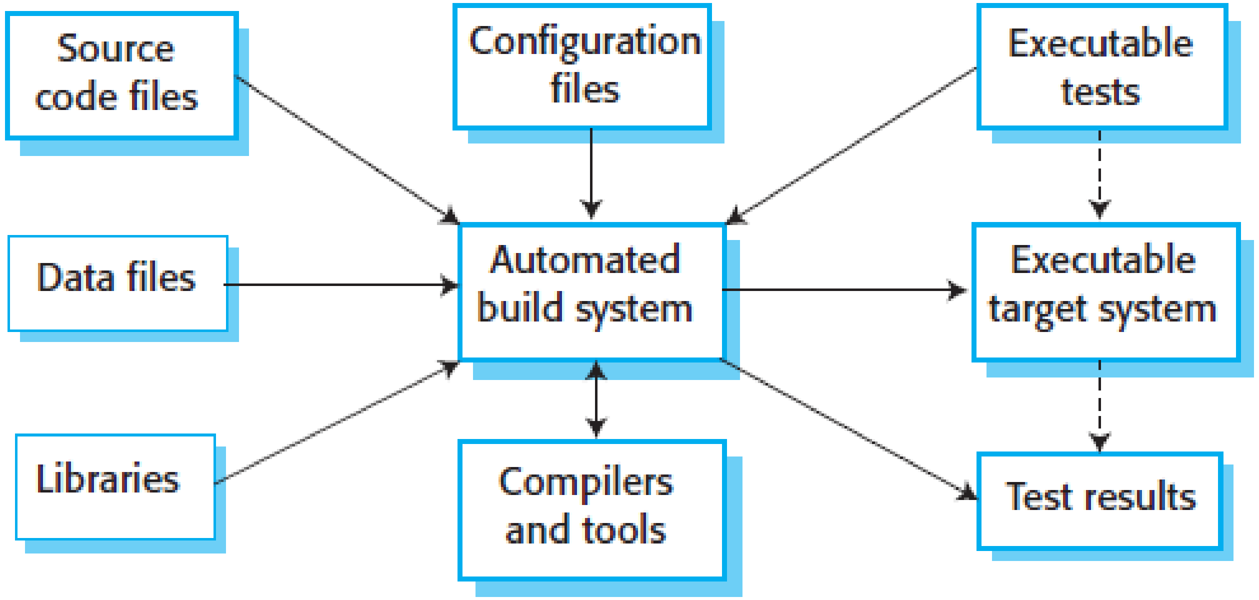
- When a new version is created, the system simply stores a delta, **a list of differences**, **between** the **new** version and the **older** version used to create that new version.
- Deltas are usually stored as lists of changed lines, and, by applying these automatically, one version of a component can be created from another.
  - Disadvantage: a long time to apply all the deltas.
  - Git uses deltas within *packfiles* (smaller files combined into an indexed single file) to further reduce their size.



System Building

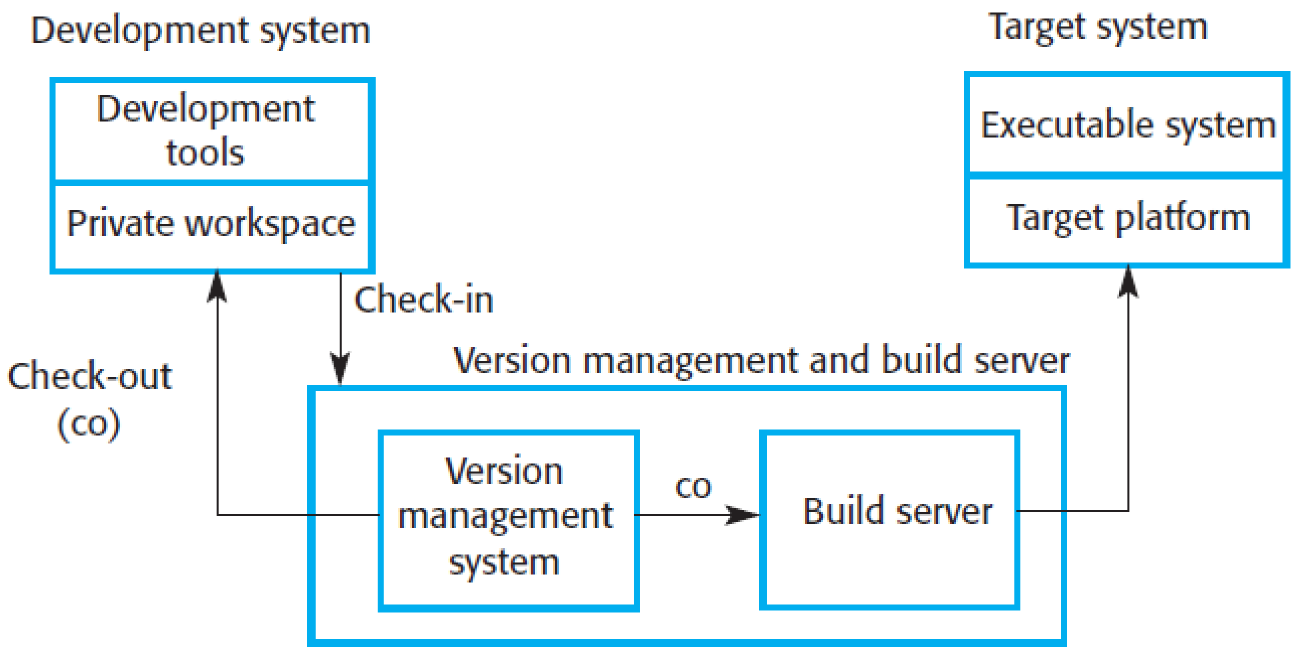
System Building

- System building is the process of **creating** a complete, **executable** system by compiling and linking the system components, external libraries, configuration files, and other information.



System Building

- In the building process, three different system platforms may be involved:
  1. The development system
  2. The build server
  3. The target environment

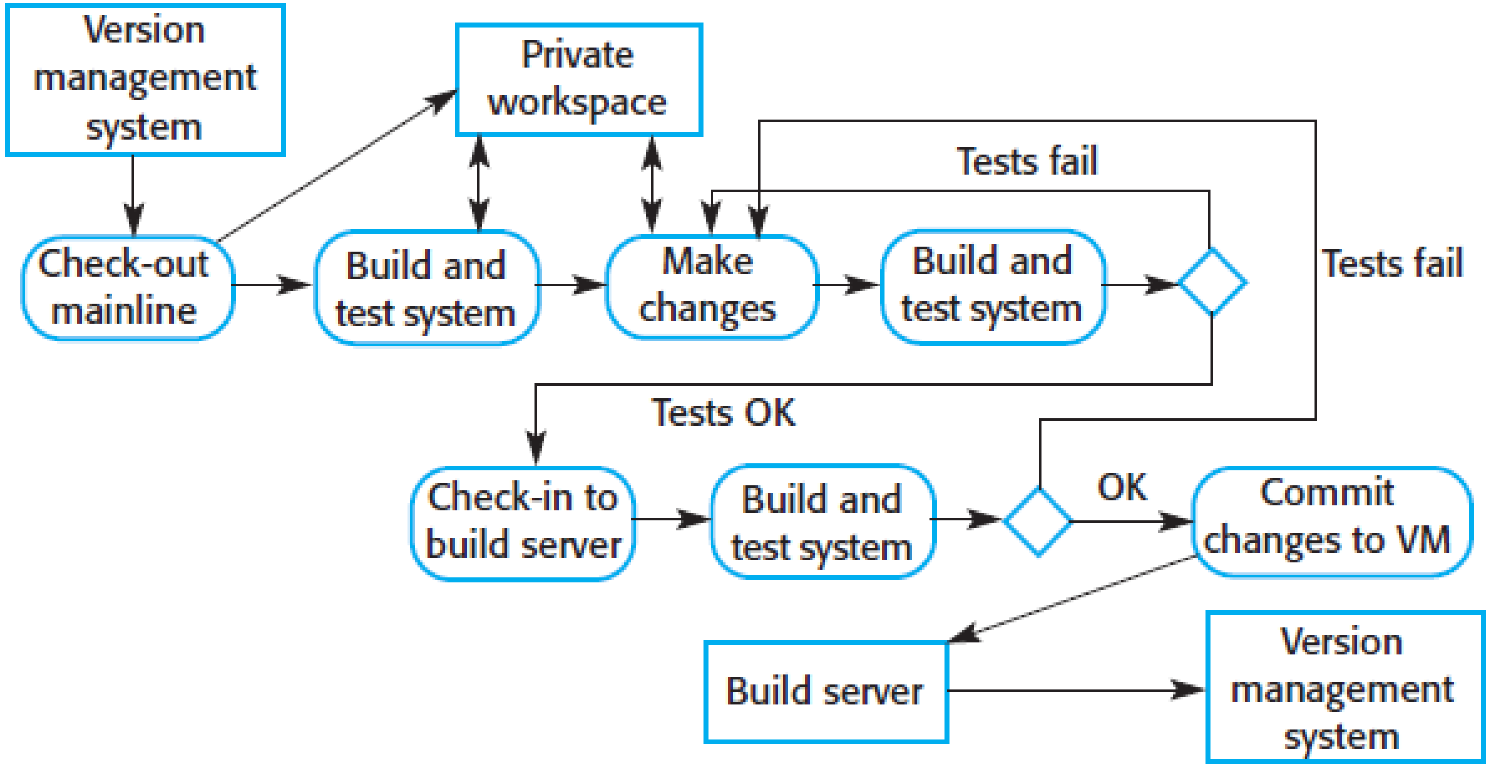


System Building

- 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**.

System Building

- We will see continuous integration in a coming lecture.



System Building

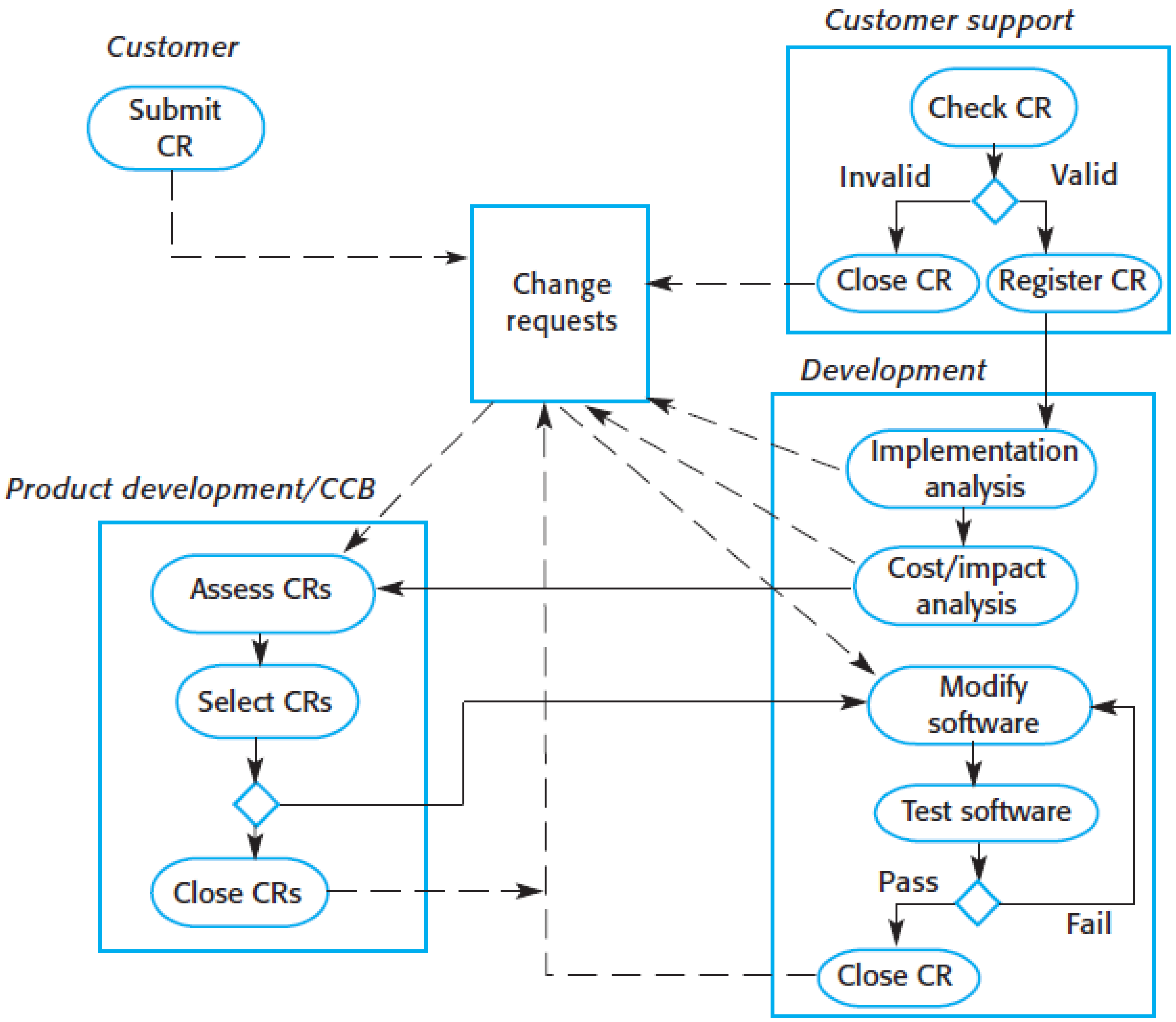
- Ideally, you should be able to build a complete system with a single command or mouse click.
- Tools for system integration and building include some or all the following features:
  1. Build **script generation**
  2. **Version control** system integration
  3. Minimal recompilation
  4. **Executable** system **creation**
  5. **Test automation**
  6. Reporting
  7. Documentation generation

Change Management

Change Management

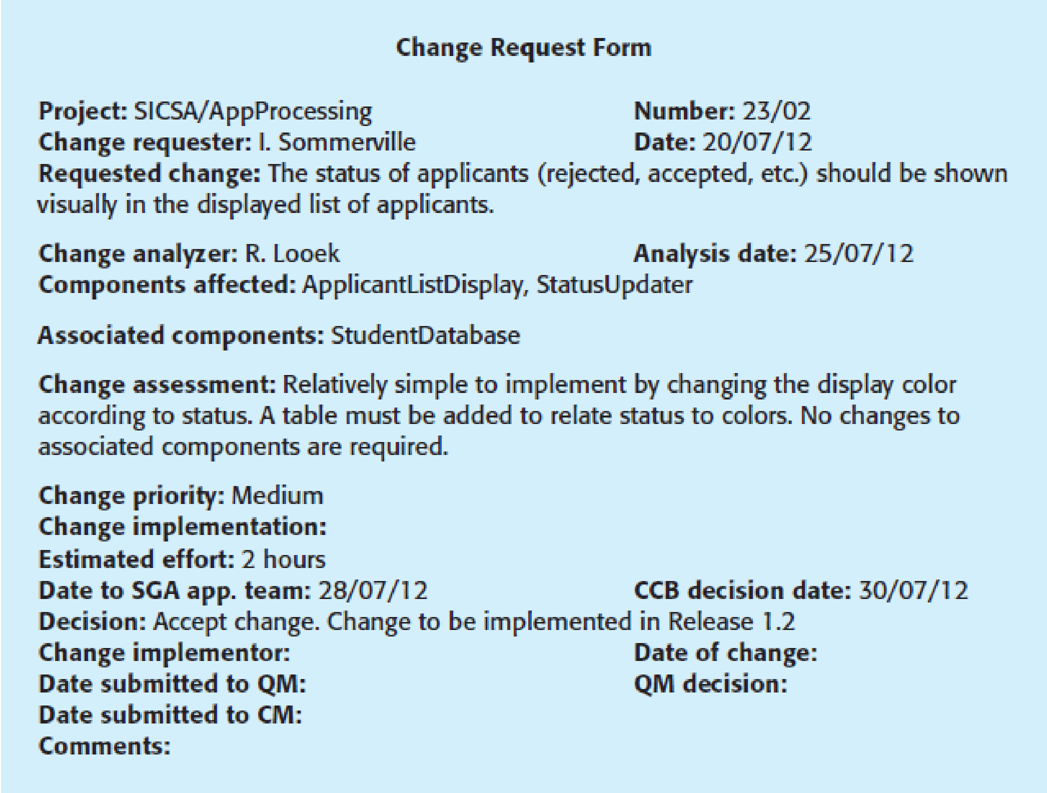
- Organizational needs and **requirements** change during the lifetime of a system, **bugs** must be repaired, and systems must **adapt** to changes in their environment.
- Change management is intended to ensure that the **evolution** of the system is **controlled**, and that the most urgent and cost-effective changes are **prioritized**.
- Change management is the process of analysing the **costs and benefits** of proposed changes, approving those changes that are **cost-effective**, and tracking which components in the system have been changed.

The Change Management Process



Change Management

- The change management process is initiated when a system stakeholder completes and submits a change request describing the change required to the system.



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:

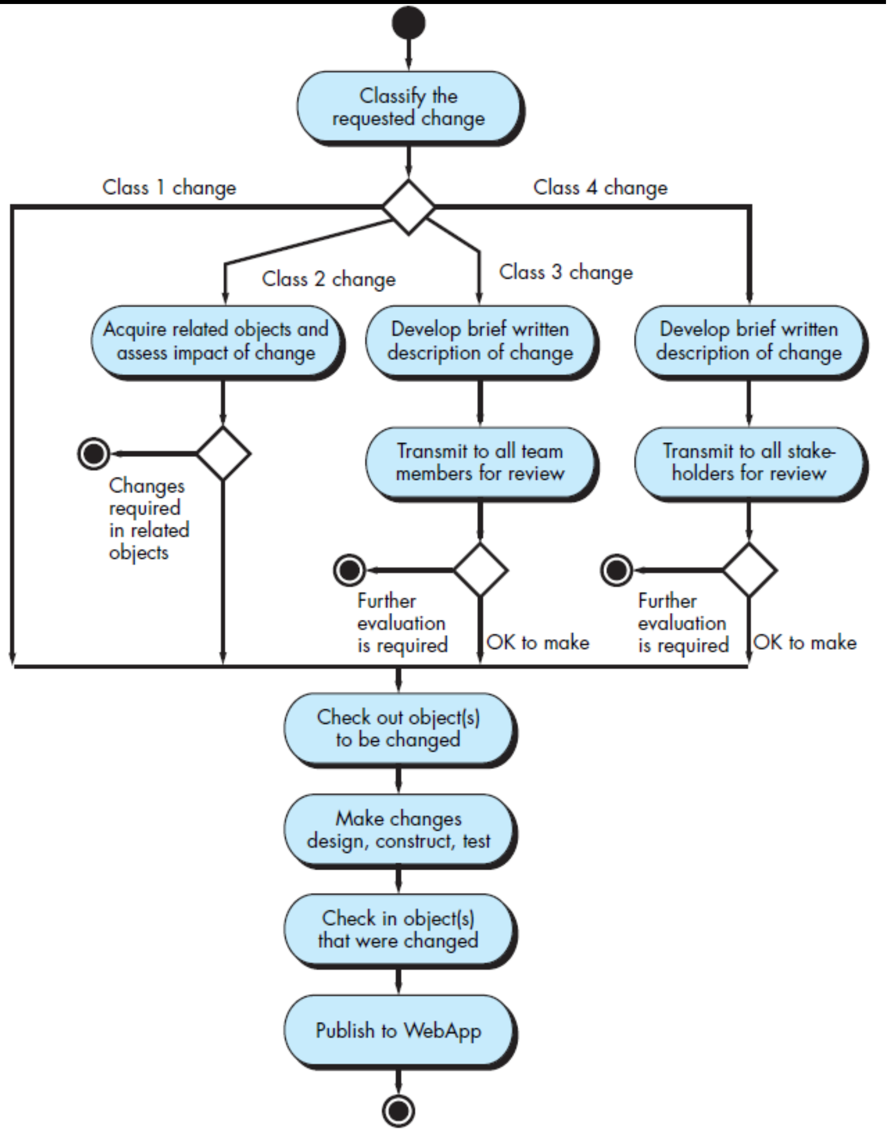
Change Management

- The factors that influence the decision on whether or not to implement a change include:
  1. The **consequences of not making** the change
  2. The **benefits** of the change
  3. The **number of users** affected by the change
  4. The **costs** of making the change
  5. The product **release cycle**

Change Management for a WebApp

- To speed up the change management in WebApp and mobile software development, each change may be categorized into one of four classes:
  - *Class 1:* A content or function change that **corrects an error or enhances** local content or functionality.
  - *Class 2:* A content or function change that has an **impact on other** content **objects** or functional **components**.
  - *Class 3:* A content or function change that has a **broad impact across** an app (e.g., major extension of functionality, significant enhancement or reduction in content, major required changes in navigation).
  - *Class 4:* A **major design change** (e.g., a change in interface design or navigation approach) that will be immediately noticeable to one or more categories of user.

Change Management for a WebApp


Classify the requested change
Class 1 change
Class 4 change
Class 2 change
Class 3 change
Acquire related objects and assess impact of change
Develop brief written description of change
Develop brief written description of change
Transmit to all team members for review
Transmit to all stake-holders for review
Changes required in related objects
Further evaluation is required
OK to make
Further evaluation is required
OK to make
Check out object(s) to be changed
Make changes design, construct, test
Check in object(s) that were changed
Publish to WebApp

Release Management

Release Management

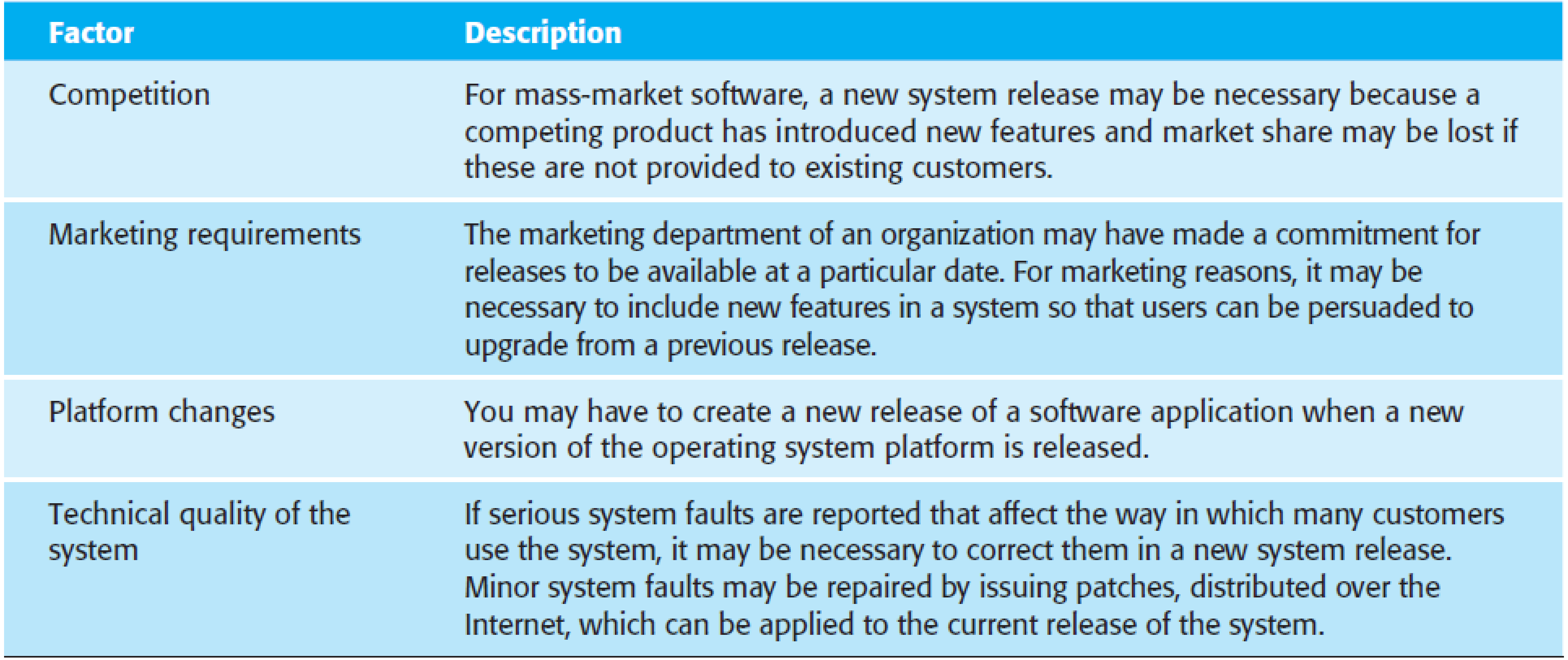
- A system release is a **version of a software system** that is **distributed to customers**.
- A software product release is **not just the executable** code of the system. The release may also include:
  - **configuration files** defining how the release should be configured for installations;
  - **data files**, such as files of error messages in different languages, 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 Management

- To document a release, you must **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.

Release Management

- **Factors that influence** a systems **release** include:



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.

Next On

Take Away Points

- Why is software configuration management important?
- Which are the activities involved in software configuration management?
    - Version management
    - System building
    - Change management
    - Release management
- Remember some configuration management terminology.

Further Reading

- Ian Sommerville, “Software Engineering”
  - Chapter 25 Configuration Management
- Pressman and Maxin, “Software Engineering: A Practitioner’s Approach”
  - Chapter 29 Software Configuration Management

Next Lecture

- Software testing: basic definitions.