

Conceitos Básicos

Engenharia de Software 2020-1

Moisés Gomes de Carvalho

Facets of SCM

The aspects of SCM are:

- Configuration item identification
- Promotion management
- Release management
- Branch management
- Variant management
- Change management

No fixed rules:

- SCM functions are usually performed in different ways (formally, informally) depending on the project type and life-cycle phase (research, development, maintenance).

SCM Activities

- Configuration item identification
 - modeling of the system as a set of evolving components
- Promotion management
 - is the creation of versions for other developers
- Release management
 - is the creation of versions for the clients and users
- Branch management
 - is the management of concurrent development
- Variant management
 - is the management of versions intended to coexist
- Change management
 - is the handling, approval and tracking of change requests

SCM 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 (*Olimpic Element*)
 - 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 and Methodology

- What are
 - Configuration Items
 - Baselines
 - SCM Directories
 - Versions, Revisions and Releases
- The usage of the terminology presented here is not strict but varies for different configuration management systems. We will see for example that the configuration management system used for this class uses different names than those mentioned in the IEEE standards.

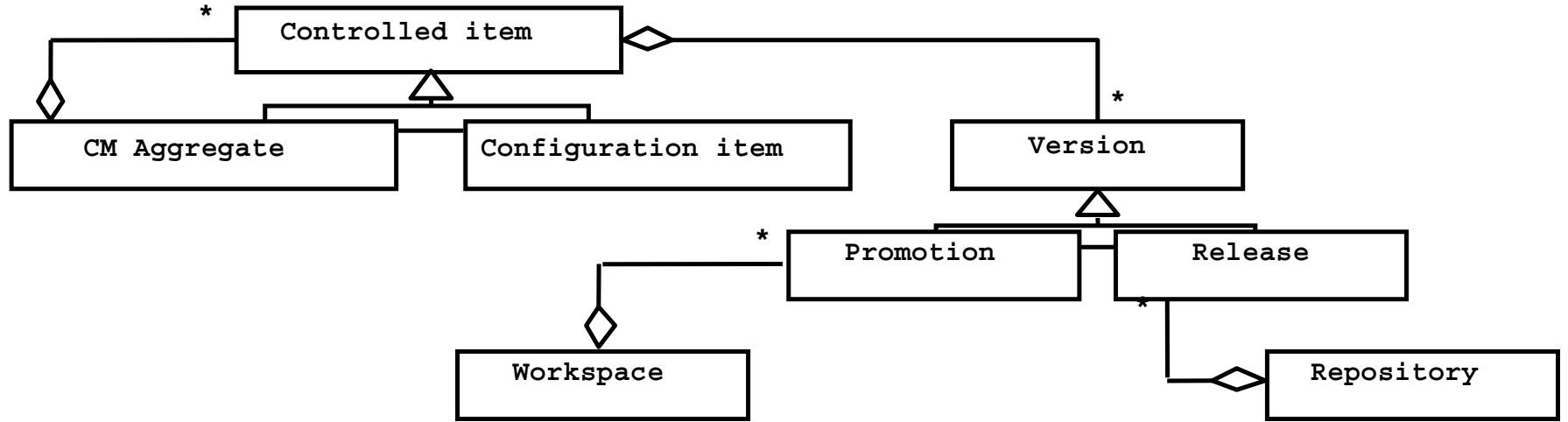
Terminology: Configuration Item

“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!

Configuration management concepts (UML class diagram).



Finding Configuration Items (CIs)

- Large projects typically produce thousands of entities (files, documents, ...) which must be uniquely identified.
- But not every entity needs to be configured all the time. Issues:
 - What: Selection of CIs (What should be managed?)
 - When: When do you start to place an entity under configuration control?
- Starting too early introduces too much bureaucracy
- Starting too late introduces chaos

Configuration Identification...

- Configuration identification is the process of establishing a baseline from which system changes are made – allows for control.
- So what needs to be under SCM?
 - Items where changes need to be tracked and controlled
 - If in doubt, add it into the inventory of items under SCM
- Common items under SCM are:
 - Source code, documentation, hardware/OS configuration

Finding Configuration Items (continued)

- Some of these entities must be maintained for the lifetime of the software. This includes also the phase, when the software is no longer developed but still in use; perhaps by industrial customers who are expecting proper support for lots of years.
- An entity naming scheme should be defined so that related documents have related names.
- Selecting the right configuration items is a skill that takes practice
 - Very similar to object modeling
 - Use techniques similar to object modeling for finding CIs

Terminology Review – 1

- **Baseline** - A product that has been formally approved, and consists of a well-defined set of consistent configuration items

Terminology Review - 2

“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.”

Examples:

Baseline A: The API of a program is completely defined; the bodies of the methods are empty.

Baseline B: All data access methods are implemented and tested; programming of the GUI can start.

Baseline C: GUI is implemented, test-phase can start.

Baseline Types

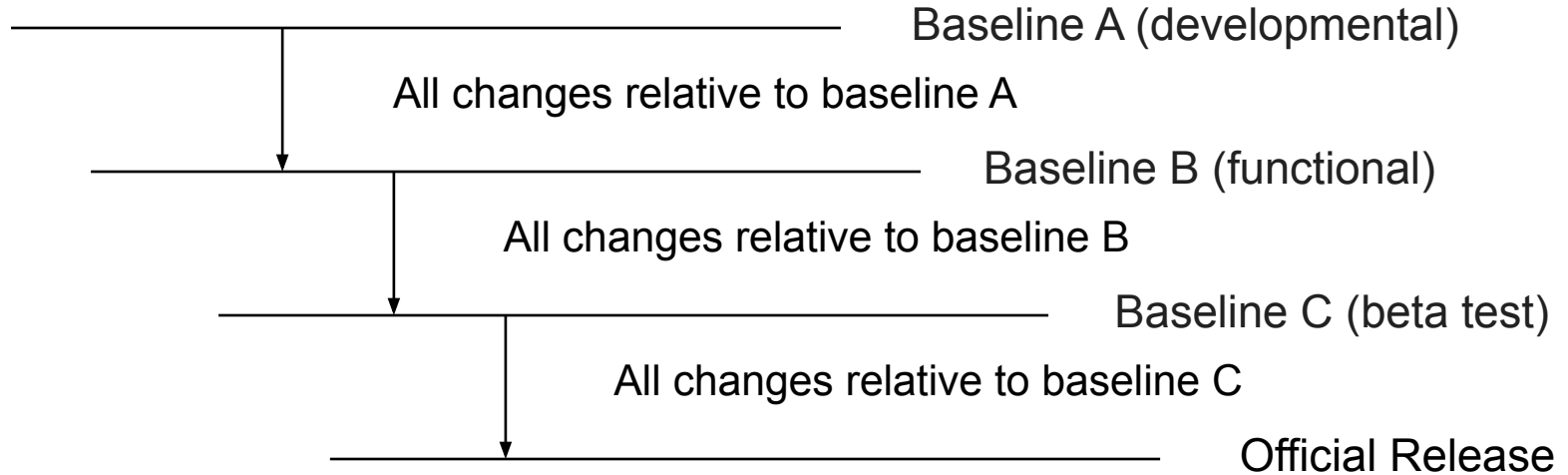
- As the system is developed a number of baselines are created:
 - *Developmental baseline* (RAD, SDD, integration test, ...).
 - Goal: coordinate engineering activities.
 - *Functional baseline* (prototype, technology preview, alpha, beta release).
 - Goal: obtain customer experiences with functional system.
 - *Product baseline* (GA with a version - win95, word 2000).
 - Goal: coordinate sales and customer support.

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, ...)
- 3 digit scheme:



Baselines in SCM



Three digit version identification scheme

<u>MUE.0.0.1:Release</u>	Alpha test release
<u>MUE.1.0.0:Release</u>	First major release
<u>MUE.1.2.1:Release</u>	Second minor release with bug fixes
<u>MUE.2.0.3:Release</u>	Second major release with three series of bug fixes

Three-digit version identification scheme

<version> ::= <configuration item name>.<major>.<minor>.<revision>

<major> ::= <nonnegative integer>

<minor> ::= <nonnegative integer>

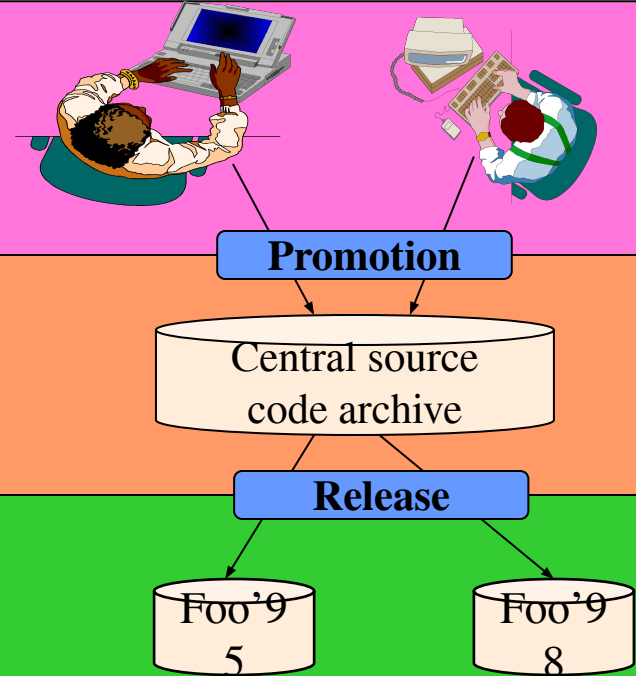
<revision> ::= <nonnegative integer>

Managed 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.
- Repository (IEEE: static library).
 - Archive for the various baselines released for general use. Copies of these baselines may be made available to requesting organizations.

Managed 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.



Configuration Identification

- A few notes...
 - Starting too early can add too many items that may really not require full configuration management.
 - Starting too late will result in a disaster
 - It is common to have 1000+ items under SCM
- A good start
 - Place all documents under SCM
 - Add code as it starts to be available
 - Remove older items and archive them
 - Remove items where the changes are minor, rare and need not be under the purview of a complete SCM

Version Allocation...

- Once a configuration item (CI) has been identified – a proper version number must be allocated
- The best option is to start with a major-minor versioning scheme
 - Major version numbers are between 0 – n
 - Minor version numbers should be between 0 – 100

Version Allocation...

- Examples:
 - Report.Java (version 1.23)
 - Major version: 1.0
 - Minor version: 23 (indicative of number of revisions to this file)
 - Project plan (version 6.34d)
 - Major version: 6
 - Minor version: 34
 - The "d" is indicative of "draft"
- Versioning scheme is developed by the company to suite their needs

Version Allocation...

- Often many companies prefix the configuration item based on its type.
 - Documentation may be prefixed "doc"
 - Source code can be "src"
 - Example: doc-pmp-2.34
 - Project management plan document (version 2.34)

Version Allocation

- New versions of software can be:
 - Maintenance releases
 - Minor upgrades
 - Technology refresh or major upgrades
 - Technology insertion

Baseline Levels

- The software system can be tagged at various stages of its evolution with a baseline number
 - **Development baseline** “n” (where the “n” can be indicative of the 10% of functionality implemented)
 - **Testing baseline** (where a specific build is created for the specific purpose of testing)
 - **Release baseline** (where the software is built for GA)
- There is no rule on when to baseline – but a good guideline is to have one a week

Terminology Review – 3

- **Version** – an *initial* release or re-release of a configuration item (*ideally different versions should have different functionality*)
- **Revision** – minor *changes* to a version that correct errors in the design/code (*typically revisions do not affect expected functionality*)
- **Release** – the *formal distribution* of a **baseline**

Configuration Control and Change Management

- Review of change activity can highlight what is changing and what is not.
 - Impact of change can be measured over time
- Issues to consider are:
 - Keeping track of changes (deltas or separate files)
 - Allows for parallel development on a single item (many developers updating the same file)

Deltas Vs Separate Files

- After the initial baseline has been established – the item is said to be under SCM.
- Changes can be tracked as:
 - Deltas: *only the changed portion is stored*
 - Separate file: *changes are stored in a new file*
- Deltas work best for text files
- Separate files is a good idea for binary file formats.

Parallel Development

- Many members can often work on the same item:
 - Two developers update the same code file (working on different functions)
 - A number of engineers may be working on a single word document containing the specifications
- Changes are tracked by each user and often merged regularly to create a synchronized version:
 - Merge conflicts are resolved via normal channels of communications
 - Effective management can reduce merge conflicts

Change Management – 1

- For best results changes should be handled formally
 - A change control board (CCB) is necessary
- CCB consists of all key stakeholders
 - Customers
 - Developers
 - Designers and architects
 - Management
 - Business strategists and financiers

Change Request

- Changes are required because:
 - A problem is discovered (bug?)
 - An enhancement is required
- Once a change is required – a “change request” is raised
- A change request (CR) will outline:
 - Current operation, nature of problem/enhancement, expected operation after system is changed

Change Control Board – 1

- All change requests (CR) are reported to the CCB for review
- CCB discusses all open CRs at regular meetings (frequency is determined by nature of project)
- CCB determines if CR identifies a “problem” or an “enhancement”
 - This is done to identify who “pays” for the change

Change Control Board – 2

- Once the change has been categorized, it is discussed in detail,
 - Probable source of problem
 - Impact of the change
 - Time and resource requirements (estimates)
- CCB will assign a priority and severity for all CRs (CRs may also be rejected)
- The CRs are assigned to development management for further action

Impact Analysis

- Before changes are made often a deep or shallow impact analysis is performed
- Impact analysis makes full use of software metrics
- Managers often track
 - Increases in complexity measures as system evolved over a period
- Trend analysis is performed based on modules and change requests to ensure flexibility

Change Management – 2

- Development managers will assign a CR to a developer (or a team)
- The requested change is made as per the plan and a full regression test suite is executed
- Configuration manager reviews the changed system
 - Ensures that all required documentation is changed
 - Ensures that the impact does not exceed estimates (too much)

Configuration Management

- To ensure proper tracking the following information needs to be collected:
 - When was the change made
 - Who made the change
 - What was changed (items modified)
 - Who authorized the change and who was notified
 - How can this request be cancelled
 - Who is responsible for the change
 - Priority and severity
 - How long did the change take (vs estimate)

Change Management – 3

- Changes will need to be modified or cancelled
 - This is required if the team assigned the change request need more resources and time than estimated
- These decisions are delegated to CCB with the extra information added along with the CR.
- A CR can be *assigned, deferred, rejected, closed*.
- All changes are reviewed and modified

Change Control Board – 3

- The complexity of the change management process varies with the project
- Small projects can perform change requests informally
- Complex projects require detailed change request forms and the official approval by one or more managers
- For safety critical projects – change management is very rigorous
 - Example: software changes to an airplane avionics system
- Change request management is supported by robust software packages

Change Management – 4

- To ensure effective change management:
 - Each working version is assigned an identification code
 - As a version is modified, a revision code or number is assigned to each resulting changed component
 - Each component's version and status as well as a history of all changes are tracked

Configuration Auditing – 1

- Key philosophy is “trust by verify”
- Configuration auditing is a process to:
 - Verify that the baseline is complete & accurate
 - Check that changes made and recorded
 - Documentation reflects updates
- Audits can be rigorous, or on a random set of configuration items
- A regular audit is required to ensure that SCM is working efficiently:
 - Can reveal weaknesses in processes, tools, plans as well as resources

Configuration Auditing – 2

- The two main types of audit are:
 - **Physical audit:** *are all identified items have a correct version and revision, this helps us remove old and unnecessary items.*
 - **Functional audit:** *verifies that the items under SCM satisfy defined specifications.*

Status Accounting

- Simple record that can identify all items under SCM
 - Location of the component, who placed it under SCM etc
 - The current version
 - Revision history (change log)
 - Pending CRs
 - Impact analysis trends
- Status accounting is vital to enable control and effective management

Roles and Responsibilities...

- Configuration manager
 - Responsible for approving configuration items
 - Responsible for development and enforcement of procedures
 - Approves STM (ship to manufacture) level release
 - Responsible for monitoring entropy
- Change control board
 - Approves and prioritizes, or rejects change requests

Roles and Responsibilities...

- Software engineers
 - Responsible for identification and versioning of configuration items
 - Create promotions triggered by change requests or the normal activities of development.
 - Update the items to incorporate requested changes – they also resolve any merge conflicts

Standards

- Approved by ANSI:
 - IEEE 828: software configuration management plans
 - IEEE 1042: guide to software configuration management

Outline of the SCMP (IEEE 828-1990)

- 6 main sections:
 - Introduction
 - Management
 - Activities
 - Schedule
 - Resources
 - Maintenance

Conformance to the IEEE Standard 828-1990...

- A document titled “software configuration management plan” is required.
- Minimum required sections are:
 - Introduction
 - Management
 - Activities
 - Schedule
 - Resources
 - Plan maintenance

Conformance to the IEEE Standard 828-1990

- Quality guidelines:
 - All activities defined in the plan must be assigned
 - All configuration items should have a defined process for establishing the baseline and change control
- If these criteria are met, the SCMP can state:
"This SCMP conforms with the requirements of IEEE Std. 828-1990."

Tools

- A large number of tools are available
- Examples:
 - RCS – pessimistic locking control system, no support for parallel development
 - CVS - based on RCS, allows concurrent working using optimistic locking
 - Git - CVS evolution

Repositories

- All items under configuration are placed into a repository
- This repository is controlled by a tool like CVS or Git
- A Git repository can handle both binary and text files
 - Changes are stored as deltas for text files
 - Separate files are stored for binary files

Functions of a Repository

- Access to repository is controlled by a security policy (in GIt a username/password or SSH key)
- After a user is logged into the repository they can:
 - Check-out a file for use
 - Check-in a changed file back into the repository
 - Tag the repository at a certain date/time
 - Place a new file into the repository

File Check-Out

- Check-out:
 - The repository marks the file as checked-out
 - In concurrent systems (like CVS/Git), a list of all users that have checked out the file is maintained
 - In locking systems (like RCS), only one user can check-out a file at a time. In essence the file is locked for future use

File Check-In...

- Check-in:
 - The file is modified and the changed version is checked back into the repository
 - If the file was locked – then only the user that checked it out can unlock it by checking in the same file (verified on file name only)
 - In a concurrent system – the first user to check in will cause the system to enter a conflict resolution mode

File Check-In...

- In a concurrent versioning system:
 - Users can potentially work on the same file at the same time
 - If two users change the same sections of the file then when checking in, the system will flag a merge conflict
 - Merge conflicts have to be resolved before the file can be placed into the repository
 - This allows more flexibility, however it can be more dangerous as well when used without sufficient training

File Check-In

- Most companies have procedures that will outline the steps to take before a file is checked back into the repository
- All changes made to the file are documented in a “status log” during the check-in process
 - Some procedures require the change request number to be stated after a certain date

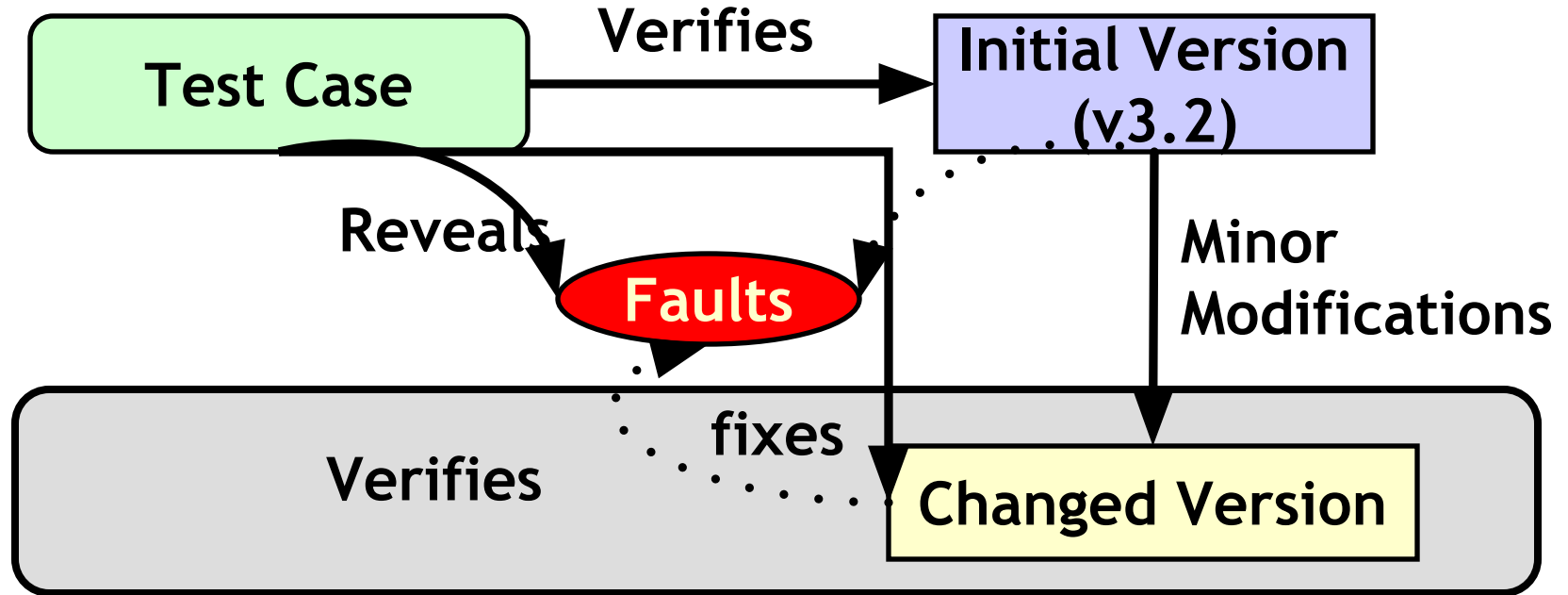
Ensuring Build Consistency

- When developers check-in source code with modifications – the changes may cause more bugs
- To ensure that new changes do not cause unexpected failures many techniques are used by developers:
 - Regression testing
 - Compile & link verification
 - Static audits
 - Metrics trends
- This aspect of development is one of the most important and requires careful monitoring

What is Regression Testing?

- Simply put, it is repetition of existing tests
 - Usually done after minor changes are made to code
 - It does not apply for enhancements
 - Before checking in the changes into a repository
- It can be seen as selective testing
- Intention is to show that modifications have not caused unintended effects
- Verifies that the system still complies with its specified requirements

Regression Testing



Regression Testing Limitations

- Regression test suite does not contain tests for new or changed capabilities
- When a primary test suite is promoted to become a regression test suite, it is no longer effective as a primary test suite
 - Once a version has passed all of its test cases, the test suite has revealed all the bugs that it can and must be changed to look for new changes

Compile and Link Verification

- This process is applied mainly to source code before checking it into a repository
 - The changed copy is built locally on the developers workspace
- The aim is to ensure that there are no compiler errors, warnings or link failures when integrated with the existing set of code as mirrored on the repository
- This is the simplest form of check, and increases confidence – this does not check for bugs.

Static Audits

- Applicable to source code
- Before check-in the changed code is passed through a static code verification tool
- Any violations or failure to meet company standards are picked up by this tool
- All issues are resolved before check-in
- This step can ensure that the overall quality of the code in the repository is higher
- Does not detect functional bugs or errors that can be caused at runtime.

Metrics Trends

- This step is another step to ensure that poor quality code does not enter the repository
- Company can define a global standard on the size of methods, allowed complexity etc...
 - If changes have caused a deviation from the norm, then it required approval from the team leader before being checked into the repository
- The code is analyzed over a period to ensure that over time the code does not deteriorate.

Tagging a Repository...

- As software evolves the files that make up the system are all at different version levels
- Example (from an O/S can be)
 - Text editor v3.0
 - Internet explorer v5.5
 - Kernel build 1885
 - Windowing system build 23
 - Etc...
- Before the above configuration is released – we need to note down these numbers. That process is known as tagging the repository

Tagging a Repository

- Once a repository is tagged
 - We can revert back to this level even if a large number of changes are made
 - The tag number can be issued to the quality and testing teams for verification.
- Repository is tagged at regular intervals,
 - Weekly is typical early in the life cycle
 - Daily towards the end or if the process calls for high-frequency integration

Keywords

- Configuration item
- Version
- Baseline
- Release
- Revision
- Change management
- Configuration management

Summary

- SCM is an insurance policy.
- Effective SCM can ensure:
 - Rework cost is reduced
 - Effort put into development is not wasted
 - There is no loss of control as software evolves

References

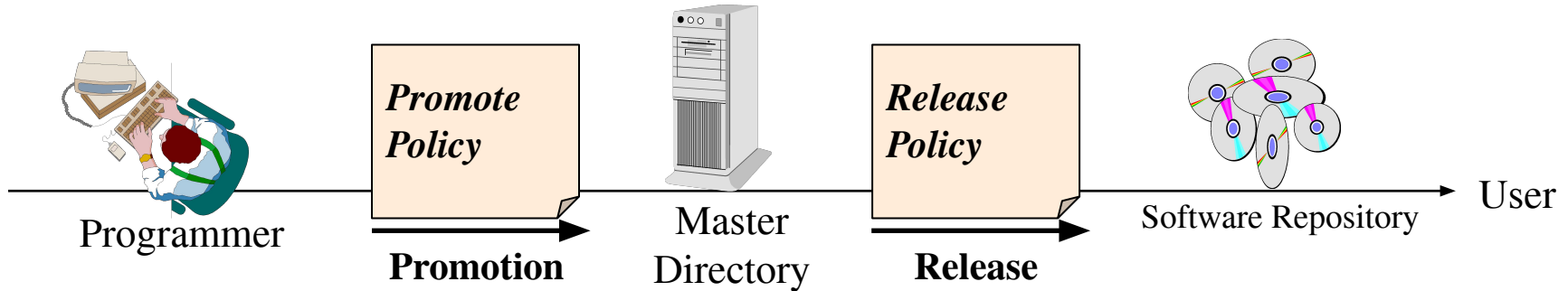
- Ghezzi, C., Jazayeri, M., and Mandrioli, D. (1991) *Fundamentals of Software Engineering*, Prentice Hall
- Horch, J. W. (1996) *Practical guide to software quality management*, Artech House.
- Bruegge, B., and Teubner, G., *Object-oriented software engineering: conquering complex and changing systems*
- IEEE standard 828-1990
- Lientz, B. P., and Swanson, E. B. (1981) "Problems in application software maintenance", *Communications of the ACM*, 24(11):763-769.

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.

Controlling Changes

- Two types of controlling change:
 - *Promotion*: The internal development state of a software is changed.
 - *Release*: A set of promotions is distributed outside the development organization.



- Approaches for controlling change to libraries (Change Policy)
 - Informal (good for research type environments)
 - Formal approach (good for externally developed CIs and for releases)

Change Policies

- Whenever a promotion or a release is performed, one or more policies apply. The purpose of change policies is to guarantee that each version, revision or release (see next slide) conforms to commonly accepted criteria.
- Examples for change policies:
 - “No developer is allowed to promote source code which cannot be compiled without errors and warnings.”
 - “No baseline can be released without having been beta-tested by at least 500 external persons.”

Version vs. Revision vs. Release

- Version:

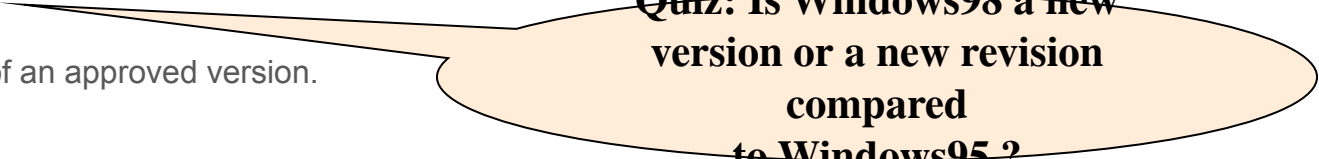
- An *initial* release or re-release of a configuration item associated with a *complete compilation* or recompilation of the item. Different versions have different functionality.

- Revision:

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

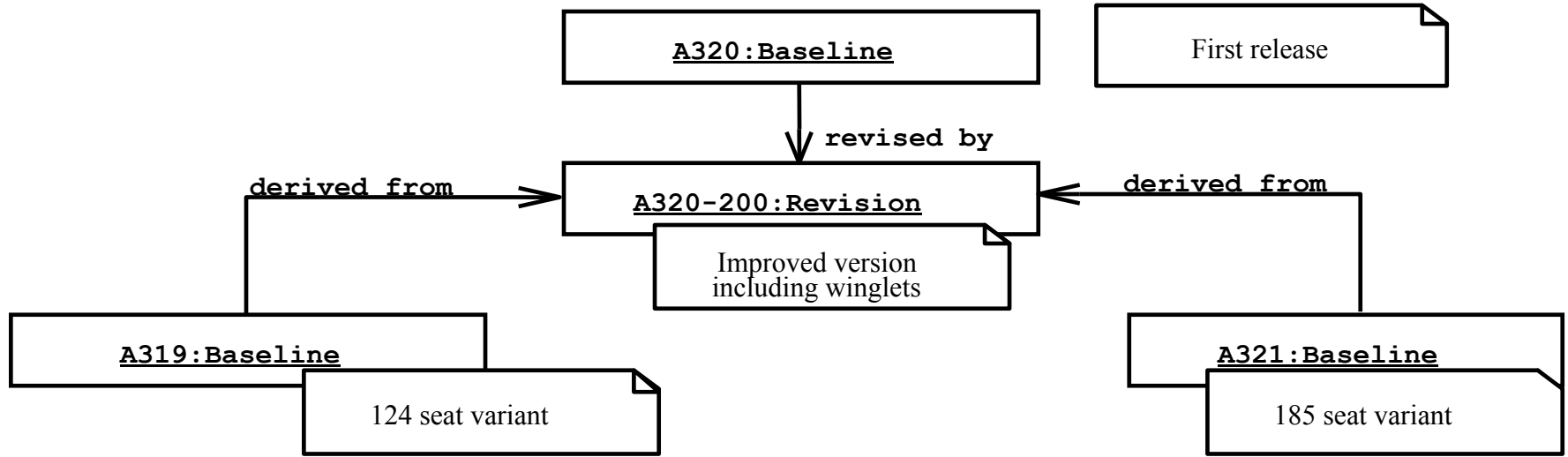
- Release:

- The *formal distribution* of an approved version.



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

Examples of baselines, revisions, and variants.

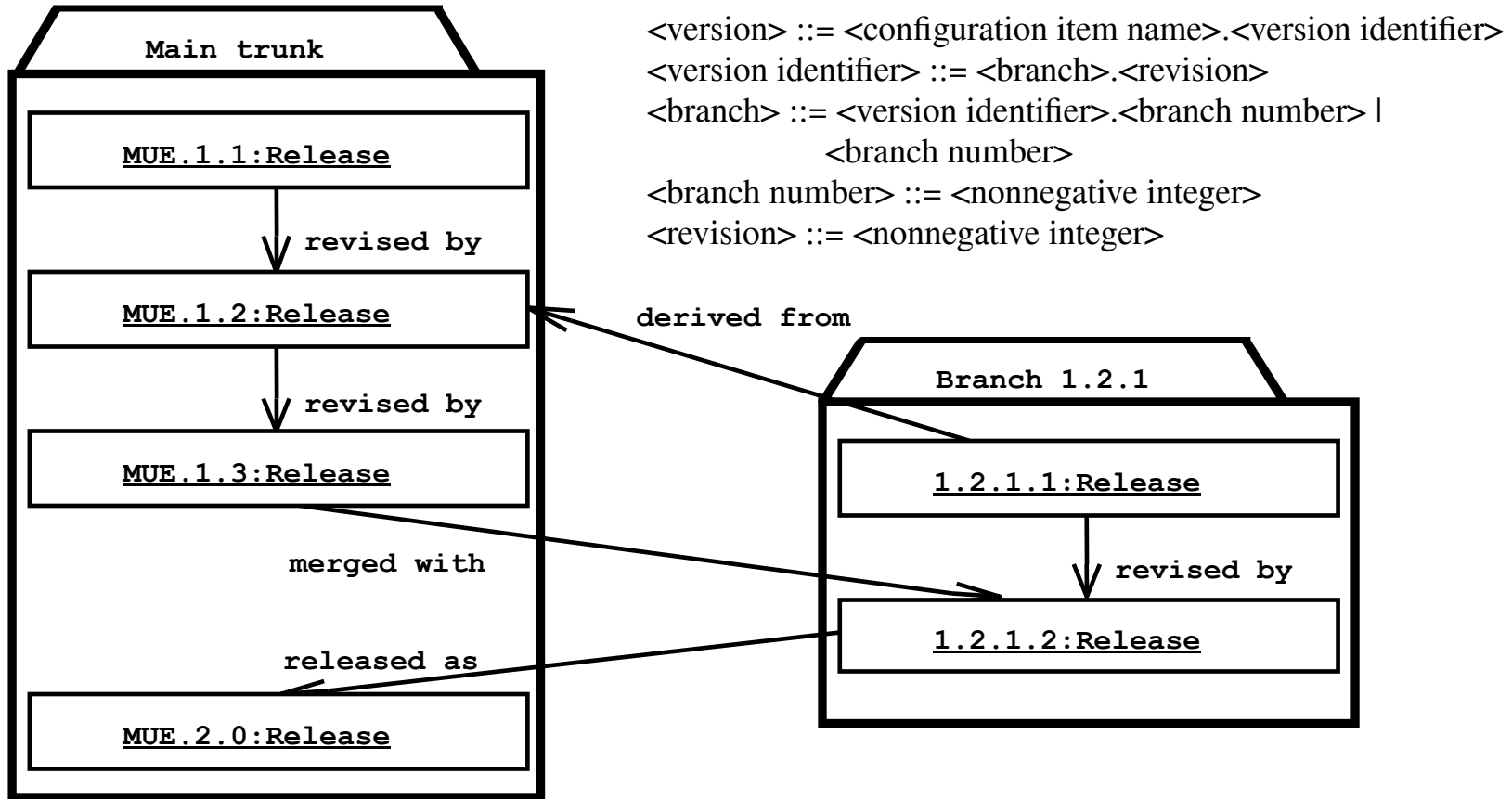


The A319, A320, and the A321 are all based on the same design. They vary mostly by the length of their fuselage.

Managing Concurrent Development

- In big projects, developers frequently want to change the same items
- Need to support released versions and new development concurrently.
- HOW?

Branches: CVS



SCM planning

- Software configuration management planning starts during the early phases of a project.
- The outcome of the SCM planning phase is the *Software Configuration Management Plan (SCMP)* which might be extended or revised during the rest of the project.
- The SCMP can either follow a public standard like the IEEE 828, or an internal (e.g. company specific) standard.

The Software Configuration Management Plan

- Defines the *types of documents* to be managed and a document naming scheme.
- Defines *who takes responsibility* for the CM procedures and creation of baselines.
- Defines *policies for change* control and version management.
- Describes the *tools* which should be used to assist the CM process and any limitations on their use.
- Defines the *configuration management database* used to record configuration information.

Outline of a Software Configuration Management Plan (SCMP, IEEE 828-1990)

- 1. Introduction
 - Describes purpose, scope of application, key terms and references
- 2. Management (WHO?)
 - Identifies the responsibilities and authorities for accomplishing the planned configuration management activities
- 3. Activities (WHAT?)
 - Identifies the activities to be performed in applying to the project.
- 4. Schedule (WHEN?)
 - Establishes the sequence and coordination of the SCM activities with project mile stones.
- 5. Resources (HOW?)
 - Identifies tools and techniques required for the implementation of the SCMP
- 6. Maintenance
 - Identifies activities and responsibilities on how the SCMP will be kept current during the life-cycle of the project.

Tailoring the SCMP

- The IEEE standard allows quite a bit flexibility for preparing an SCMP.
- To conform to the rest of the project, the SCMP may be
 - tailored upward:
 - to add information
 - to use a specific format
 - tailored downward
 - Some SCMP components might not apply to a particular project.
 - Instead of omitting the associated section, mention its applicability.
 - Information that has not been decided on at the time the SCMP is approved should be marked as “to be determined”.

Conformance to the IEEE Standard 828-1990

- Presentation format & Minimum information
 - A separate document or a section embedded in another document titled “Software Configuration Management Plan”.
 - 6 Sections: Introduction, Management, Activities, Schedules, Resources and Plan Maintenance
- Consistency Criteria:
 - All activities defined in the SCMP are assigned to an organizational unit or person and they are associated with resources to accomplish the activities.
 - All identified Configuration items have defined processes for baseline establishment and change control.
- If the above criteria are met, the SCMP can include the following sentence:
“This SCMP conforms with the requirements of IEEE Std 828-1990.”

Tools for Software Configuration Management

- Software configuration management is normally supported by tools with different functionality.
- Examples:
 - RCS
 - very old but still in use; only version control system
 - CVS
 - based on RCS, allows concurrent working without locking
 - Perforce
 - Repository server; keeps track of developer's activities
 - ClearCase
 - Multiple servers, process modeling, policy check mechanisms

Summary

- Software Configuration Management is an elementary part of the project management plan to manage evolving software systems and coordinate changes to them.
- SCM is performed by following a SCM plan. This plan can either follow a public standard (e.g. IEEE 828) or an internal standard.
- It is necessary to tailor a standard to a particular project:
 - Large projects need detailed plans to be successful
 - Small projects can't afford the bureaucracy of such plans
- SCM is supported by tools. Their functionality varies from simple version storage tools to very sophisticated systems with automated procedures for policy checks and support for the creation of SCM documents.

An example of change management process

