TUM

Project Management and Organization

Lecture 8: Change Management

27 May 2015

Bernd Bruegge

Technische Universität München

Institut für Informatik

Chair for Applied Software Engineering

http://wwwbruegge.in.tum.de

Outline of the Lecture

- Miscellaneous
  - Quiz
- Purpose of Software Configuration Management (SCM)
- Terminology: Baseline, Version, Revision, Release
- Software Configuration Management Activities
- Outline of a Software Configuration Management Plan
- Build and Release Management
- Continuous Integration

Quiz about 07 Contracting

- You have 7 min to answer 6 questions in Moodle

Outline of the Lecture

- ✓Miscellaneous
  - Quiz
- Purpose of Software Configuration Management (SCM)
- Terminology: Baseline, Version, Revision, Release
- Software Configuration Management Activities
- Outline of a Software Configuration Management Plan
- Build and Release Management
- Continuous Integration

Why Change Management?

- The problem:
  - Multiple people work on artifacts that are changing
  - More than one version of the artifact has to be supported:
    - Released software systems
    - Custom configured systems (different functionality)
    - Systems under development
    - Software running on different machines & 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?

- Definition Software Configuration Management:
  - A set of management disciplines within a software engineering process to develop a *baseline**
  - Software Configuration Management encompasses the disciplines and techniques of initiating, evaluating and controlling change to work products during and after a software project
- Standard:
  - IEEE 828-2012: IEEE Standard for Configuration Management in Systems and Software Engineering.

Baseline: A work product that can be changed only through a formal change control procedure.

Administering Software Configuration Management

- Software Configuration Management is a *project function* with the goal to make technical and managerial activities more effective
- Software Configuration Management can be administered in several ways:
  - Organization-wide
  - Project-specific
  - Distributed among the project members
  - Mixture of all of the above.

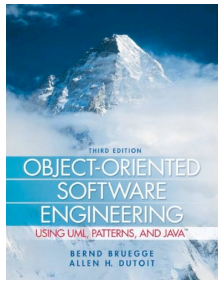
Configuration Management Roles

- Configuration Manager
  - Responsible for identifying configuration items
  - Also often 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.

Configuration Management Activities

- Configuration item identification
  - Modeling the system as a *set of evolving components*
- Promotion management
  - The creation of *versions for other developers*
- Release management
  - The creation of *versions for clients and users*
- Change management
  - The handling, approval & *tracking of change requests*
- Branch management
  - The management of *concurrent development*
- Variant management
  - The management of *coexisting versions*

This Lecture



Bruegge-Dutoit Ch13,p.551ff.

Configuration Item

Configuration Item: An aggregation of software, hardware, or both, designated for configuration management and treated as a single entity in the configuration management process.

- Software configuration items: source files, models, tests, binaries, documents, configurations
- Hardware configuration items: e.g. CPUs, bus speed frequencies, sensors, actuators

Configuration Item Identification

- 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?
    - In early days, it was an activity
    - Nowadays it should be a project function (from beginning to end of the project)

Configuration Item Identification

- Selecting the right configuration items is a skill that takes practice
  - Very similar to object modeling
  - Use techniques similar to object modeling for finding configuration items:
    1. Identify the configuration items
    2. Find relationships between the configuration items.

Terminology: Version

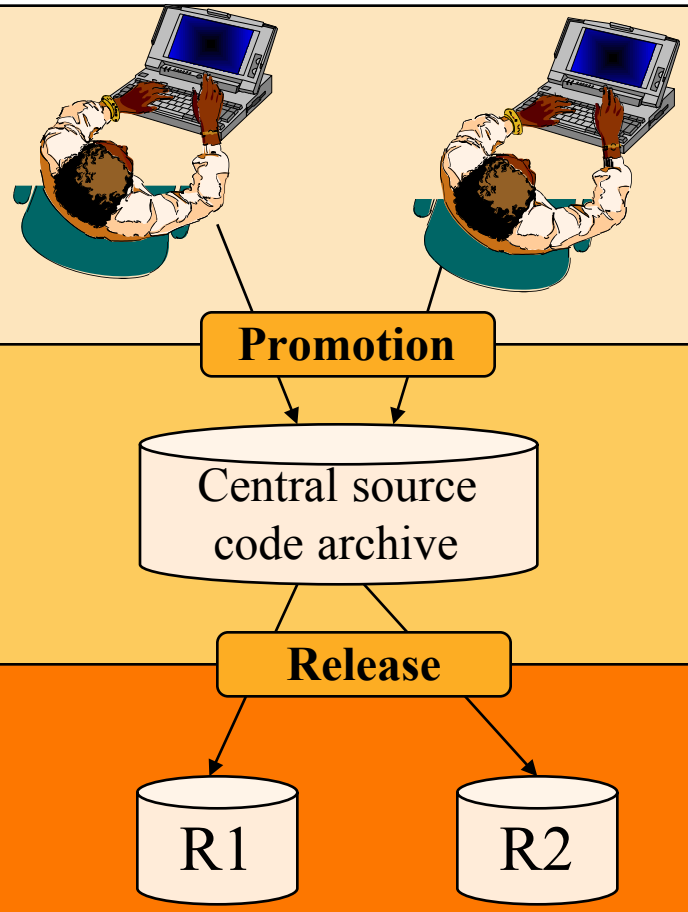
Version: The initial release or re-release of a configuration item associated with a complete compilation or recompilation of the item. Different versions have different functionality.

Terminology: SCM Directories

- Programmer's Directory (IEEE 1042: Dynamic Library)
  - Library for holding newly created or modified software entities
  - The programmer's workspace is controlled by the programmer only
- Master Directory (IEEE 1042: Controlled Library)
  - Manages the current baseline(s) and for controlling changes made to them
  - Changes must be authorized
- Software Repository (IEEE 1042: Static Library)
  - Archive for the various baselines released for general use
  - Copies of these baselines may be made available to requesting organizations.

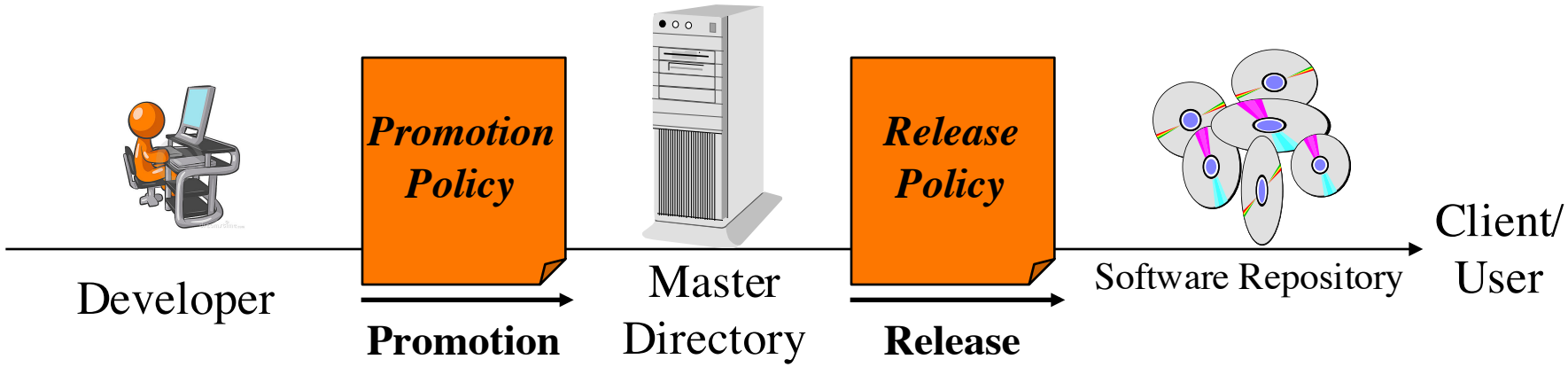
Standard SCM Directories in IEEE terminology

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



Promotion and Release Management

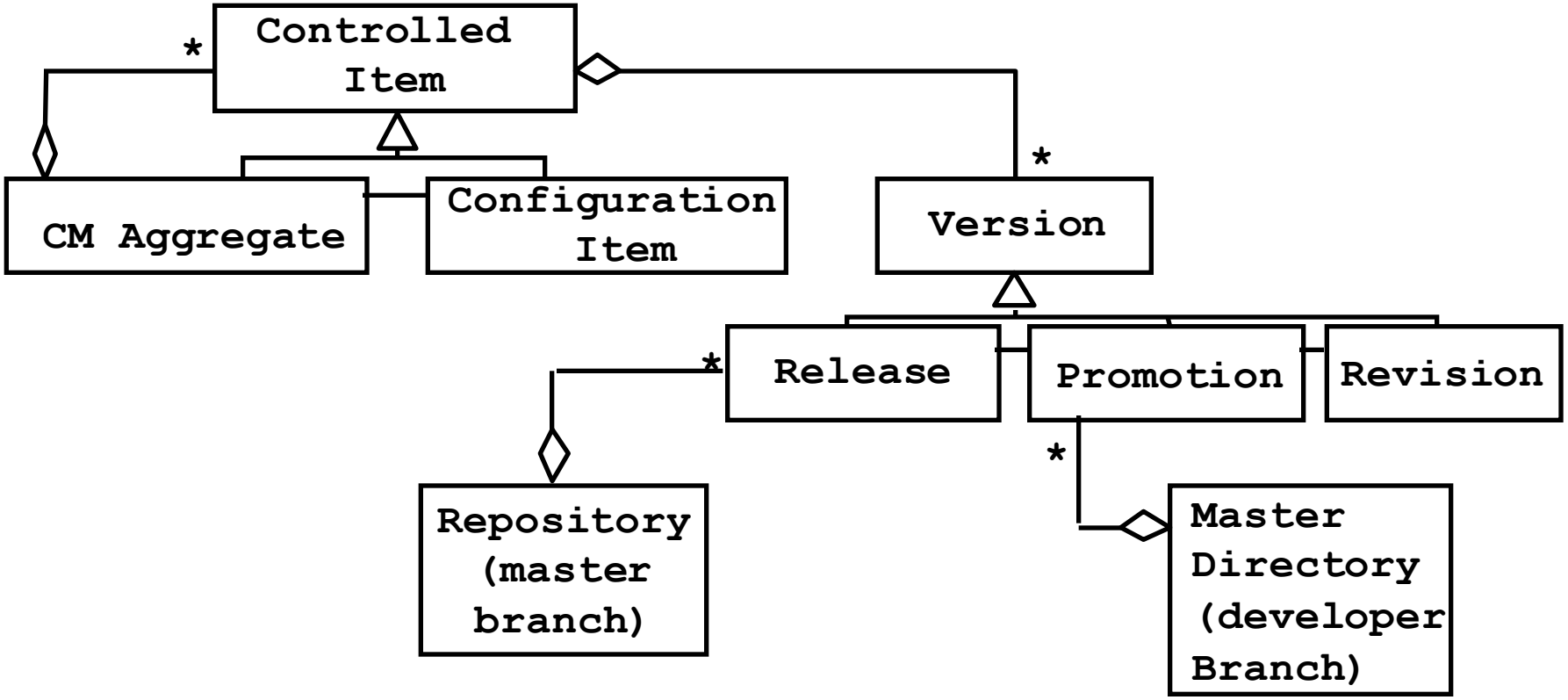
- *Promotion:* The internal development state of a software is changed
- *Release:* A changed software system is made visible outside the development organization



Terminology: Version vs. Revision

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

Object Model for Configuration Management (UML Class Diagram)



Terminology: Baseline

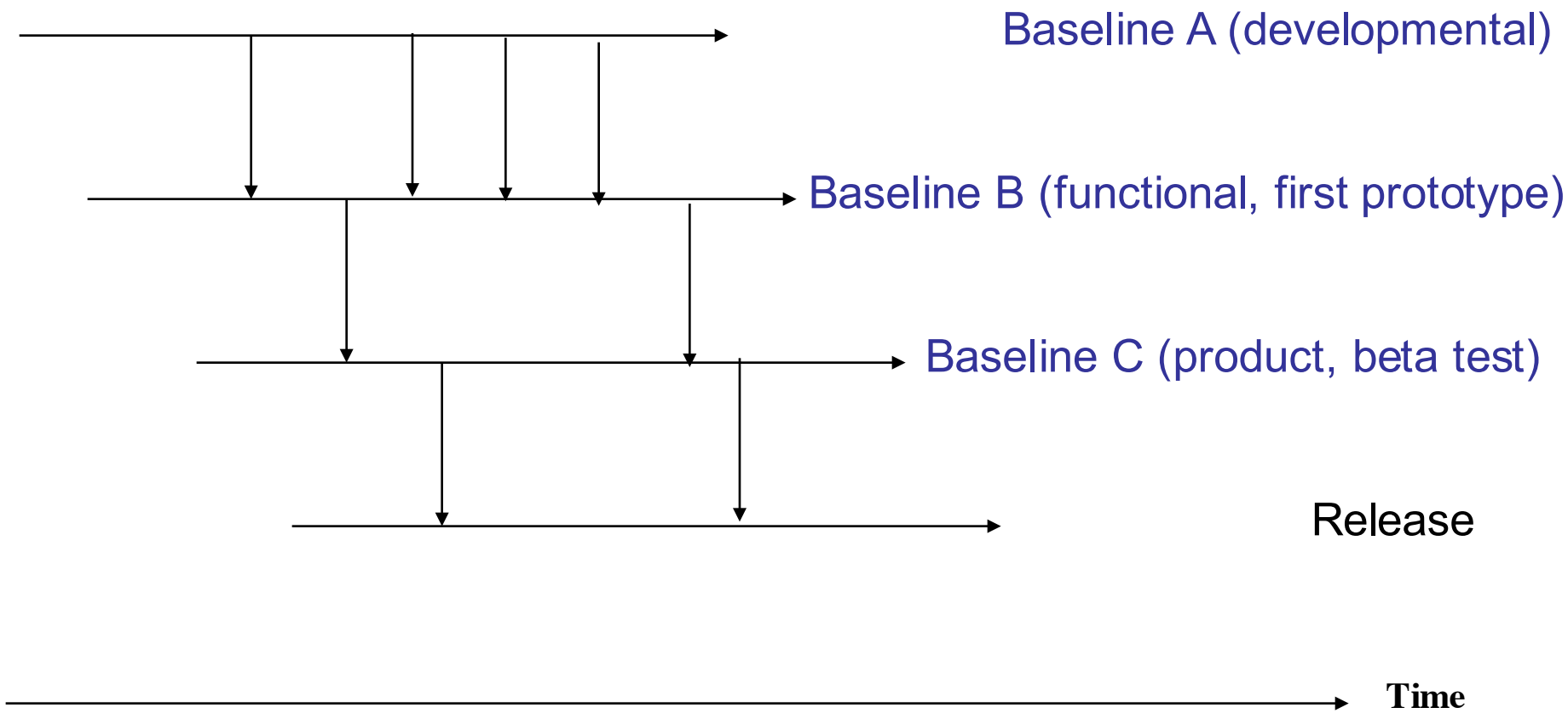
Baseline: A specification or product that has been formally reviewed and agreed to by responsible management, that thereafter serves as the basis for further development, and can *changed only be through formal change control procedures*.

- Examples:
  - *Baseline A:* The API has been 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.

Types of Baselines

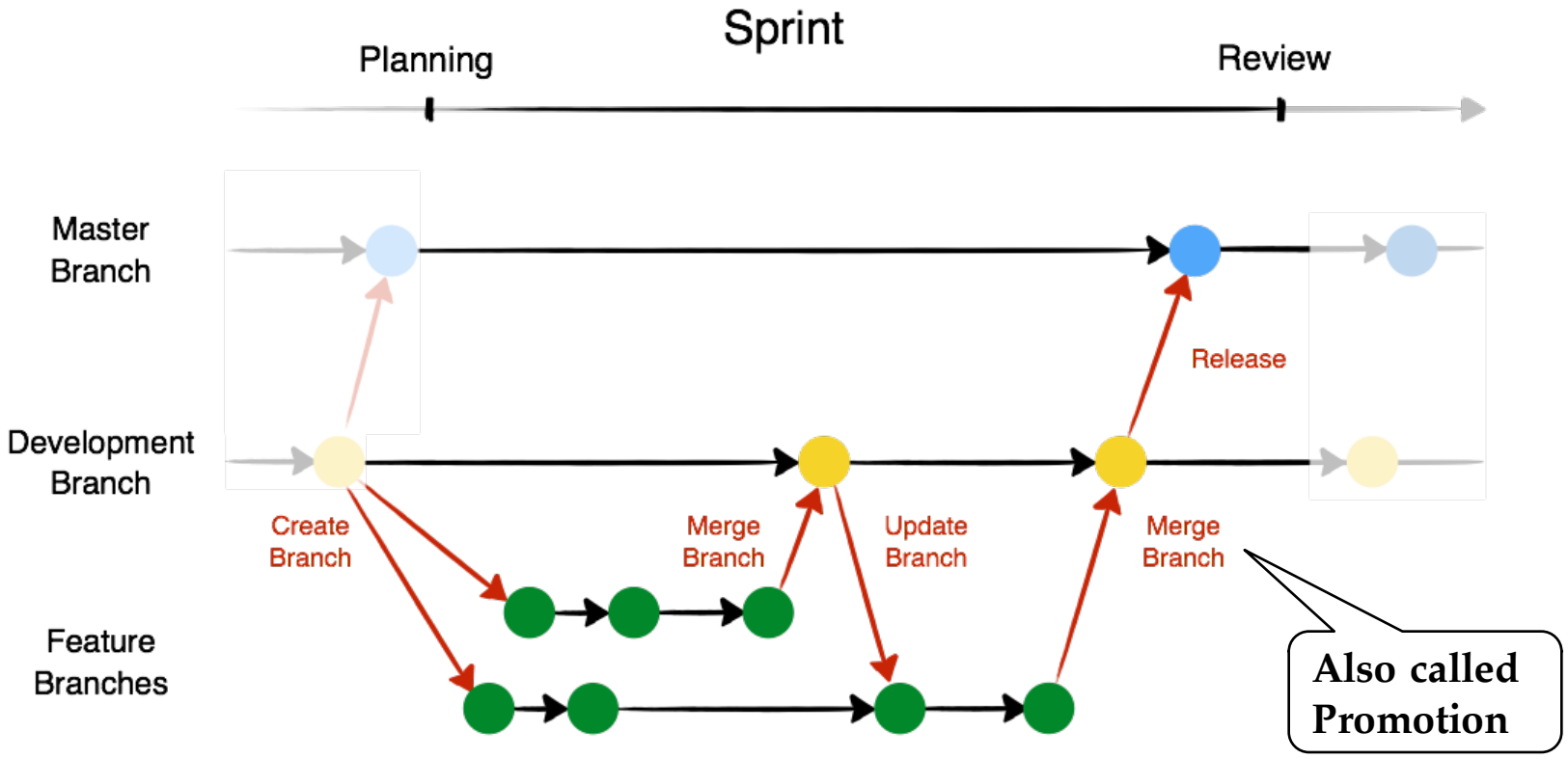
- As systems are developed, a series of baselines is developed, usually after a review (analysis review, design review, code review, system testing, ...)
  - Developmental baseline
  - Functional baseline
  - Product baseline
- Branch Management allows to transition between these baselines

Branch Management Example



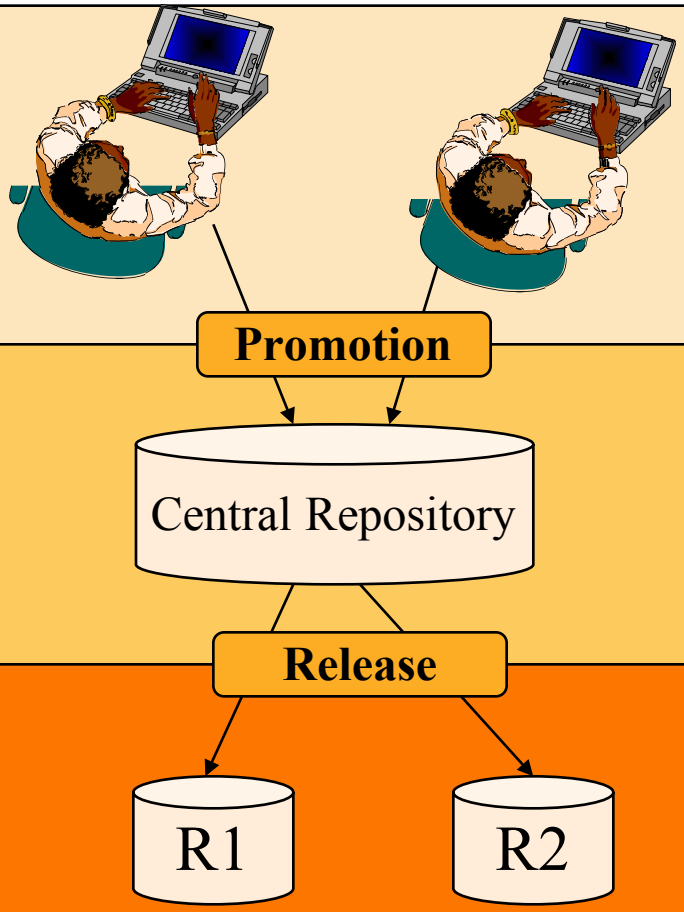
Branch Management Model

- Example of a Branching Model (e.g. in Git)
  - Master Branch: External Release (e.g. Product Increment)
  - Development Branch: Internal Release
  - Feature Branches: Incremental development and explorations



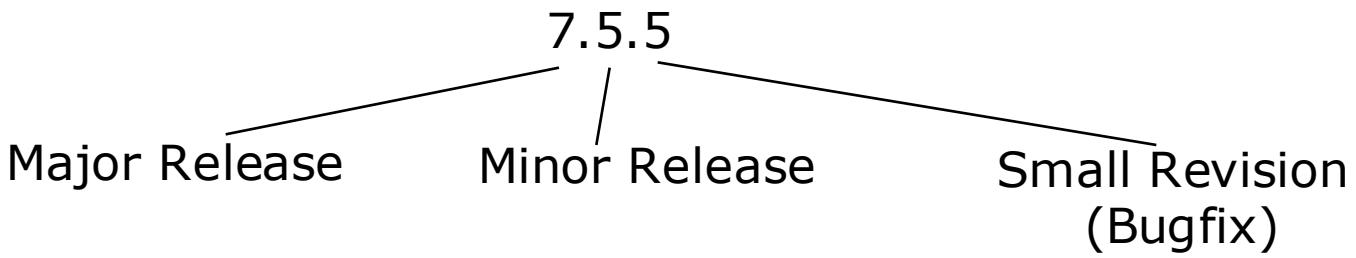
Promotions and Releases in a lightweight Branching Model

- Feature Branch
  - Completely under control of one programmer
- Development Branch
  - Central directory of all promotions
  - Candidates for releases
- Master Branch
  - External releases



Naming Schemes for Baselines (Tagging)

- Many naming scheme for baselines exist (1.0, 6.01a, ...)
- A 3 digit scheme is quite common:



History of Software Configuration Management Tools

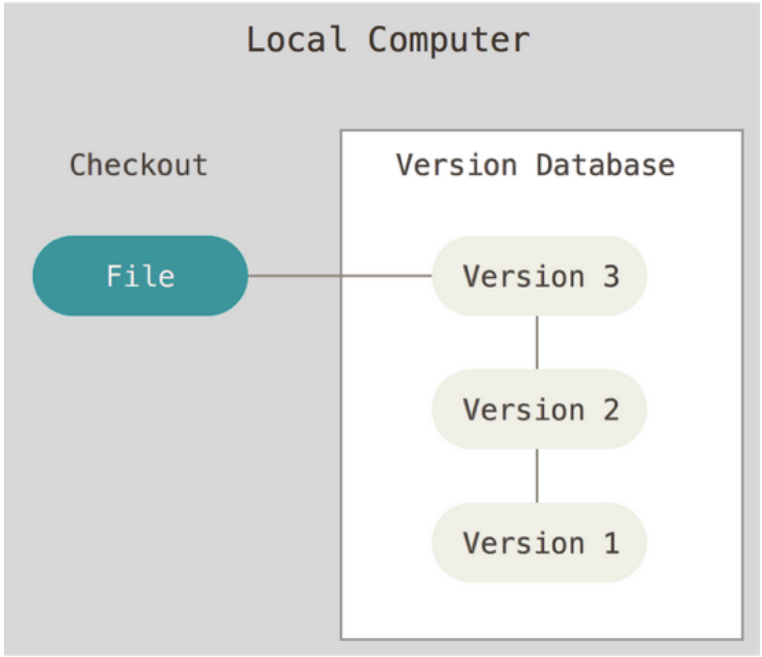
- **RCS**: The first on the block [Tichy 1975]
- **CVS** (Concurrent Version Control)
  - Based on RCS, allowed concurrent working without locking
  - http://www.cvshome.org/
- **Perforce**
  - Repository server; allows to keep track of developer's activities
  - http://www.perforce.com
- **ClearCase**
  - Multiple servers, process modeling, policy check mechanisms
  - http://www.rational.com/products/clearcase/
- **Subversion** (Slide 29)
- **Git** (Slide 31).

Architectural Styles for Version Control

- Monolithic (Standalone)
- Repository Style (Client-Server)
- Peer-to-peer

Monolithic Architecture for Version Control

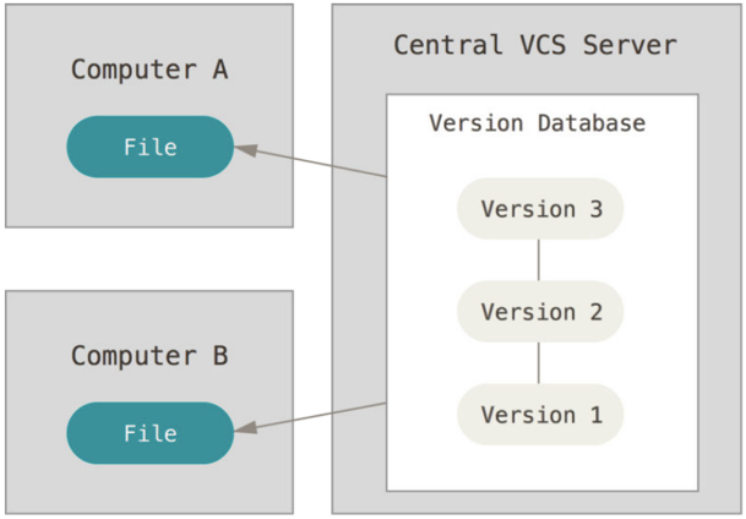
- Programmer has a simple local database that keeps all the changes to files under revision control.
- Example of Monolithic Version Control Architecture: RCS. Still distributed with many computers today.



Source: Chacon, Scott. *Pro git*. Apress, 2009

Repository Architecture for Version Control

- A single server contains all the versioned files
- Programmers check out files from to the server to their computer, change them and check them back into the server
- Administrators have fine-grained control over who can do what
- Problem:
  - Single point of failure in the Central VCS Server: Possibility of loosing all the versions and their history if the server crashes.



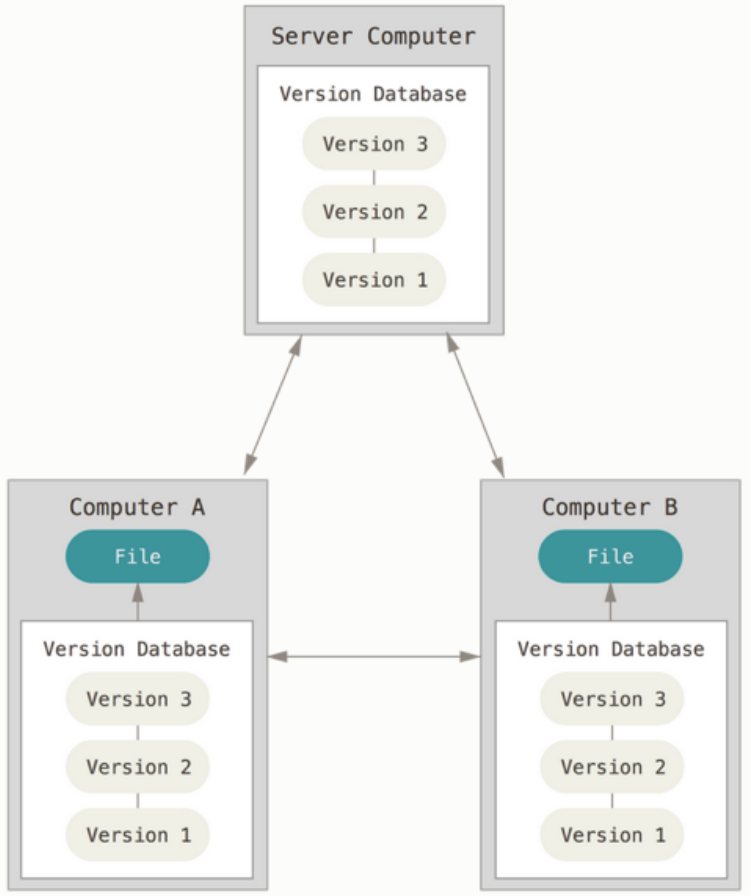
Source: Chacon, Scott. *Pro git*. Apress, 2009

Example of a Repository Architecture: Subversion

- Open Source Project (http://subversion.tigris.org)
- Based on CVS
  - Commands: `checkout, add, delete, commit, diff`
    - `Checkout: Check out a local copy from the master directory`
    - `Add: Add a local copy to the master directory`
    - `Delete: Delete a local copy in the programmer's directory`
    - `Commit: Commit local changes to master directory`
    - `Diff: Comparison between 2 versions`
- Server Options
  - Standalone installation
  - Integrated into the Apache webserver
- The time for branch management is independent of the size of the system (unlike CVS, which creates physical copies of the files, Subversion uses only tags).

Peer-to-Peer Architecture for Version Control

- Addresses the single point of failure problem
- Each programmer's directory (Computer A, Computer B,…) fully mirrors the master directory (Server Computer)
- Programmers can work offline on their own branches
- If the server dies and a programmer has a full copy of the master directory, it can be copied back to the Server computer
- **Example:** Git.



Source: Chacon, Scott. *Pro git*. Apress, 2009

Example of a Peer-to-Peer Architecture Git

- Open Source Project (http://git-scm.com)
  - Supports light-weight local branching
  - Commands: `clone, commit, push, fetch, merge, pull`
    - `Clone: creates a copy of the master directory`
    - `Commit: Commit the change to the programmer's directory`
    - `Push: Push local changes to the master directory`
    - `Fetch: Imports changes from the master directory into the programmers' directory`
    - `Merge: Merge two directories`
    - `Pull: Fetch followed by merge`
- Differences to Subversion
  - Multiple branches in each of the SCM directories, Subversion supports only single repositories
  - Branches are light-weight (no copy of the change history)
- Additional information:
  - http://git-scm.com/documentation

Configuration Management Activities

- ✓Configuration item identification
  - Modeling the system as a *set of evolving components*
- ✓Promotion management
  - the creation of *versions for other developers*
- ✓Release management
  - the creation of *versions for clients and users*
- Change management
  - the handling, approval & *tracking of change requests*
- Branch management
  - the management of *concurrent development*
- Variant management
  - the management of *coexisting versions*

This Lecture

Bruegge-Dutoit Ch13,p.551ff.

Change management

- Change management is the handling of change requests
- The general change management process:
  - The change is requested
  - The change request is assessed against requirements and project constraints
  - Following the assessment, the change request is accepted or rejected
  - If it is accepted, the change is assigned to a developer and implemented
  - The implemented change is audited.

Change Policies

- The purpose of a change policy is to guarantee that each promotion or release conforms to commonly accepted criteria

- Examples for change policies:

  "No developer is allowed to promote source code which was compiled with errors or warnings."

  "No baseline can be released without having been beta-tested by at least 500 external persons."

Change Management Activities and Responsibilities

- Configuration Control: Managing a Change Request
- Configuration Status Accounting
- Configuration Audits and Reviews
- Interface Control

Configuration Control

- Define a change request form
- Define management procedures for:
  - Identification of the need for a change request
  - Analysis and evaluation of a change request
  - Approval or disapproval of a change request
  - Implementation, verification and release of the change

Define Activities and Responsibilities

- ✓Configuration Control
- ➢Configuration Status Accounting
- ➢Configuration Audits and Reviews
- Interface Control

Configuration Status Accounting

- Answers the following questions:
  - What elements are to be tracked and reported for baselines and changes?
  - What types of status accounting reports are to be generated? What is their frequency?
  - How is information to be collected, stored and reported?
  - How is access to the configuration management status data controlled?

Configuration Audits and Reviews

- Identifies audits and reviews for the project
  - An audit determines for each configuration item if it has the required physical and functional characteristics
  - A review is a management tool for establishing a baseline.

Configuration Audits and Reviews (cont'd)

- For each audit or review the software configuration management plan (SCMP) has to define:
  - Objectives
  - The Configuration Items under review
  - The schedule for the review
  - Procedures for conducting the review
  - Participants by job title
  - Required documentation
  - Procedure for recording deficiencies and how to correct them
  - Approval criteria.

Configuration Management Activities

- ✓Configuration item identification
  - Modeling the system as a *set of evolving components*
- ✓Promotion management
  - the creation of *versions for other developers*
- ✓Release management
  - the creation of *versions for clients and users*
- ✓Change management
  - the handling, approval & *tracking of change requests*
- Branch management
  - the management of *concurrent development*
- Variant management
  - the management of *coexisting versions*

This Lecture

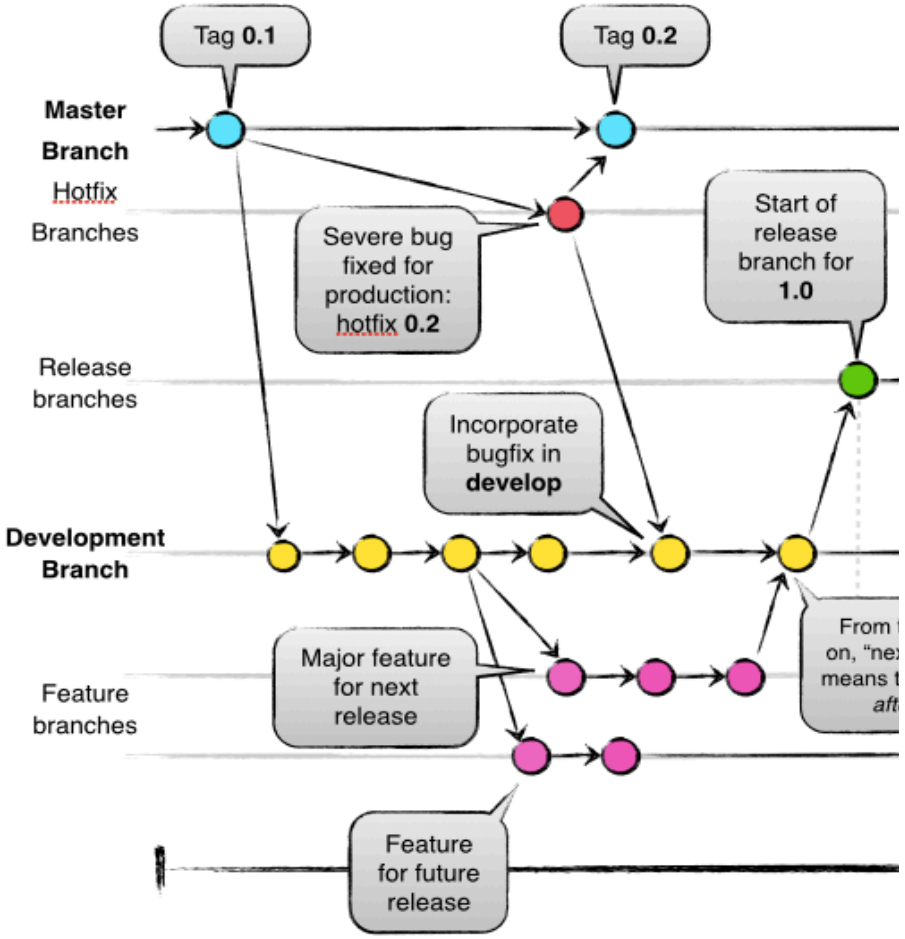
Bruegge-Dutoit Ch13,p.551ff.

Branch Management

A branching model controls the concurrent development and defines rules when branches are created and merged (promoted)

Example: Git-flow Branching Model

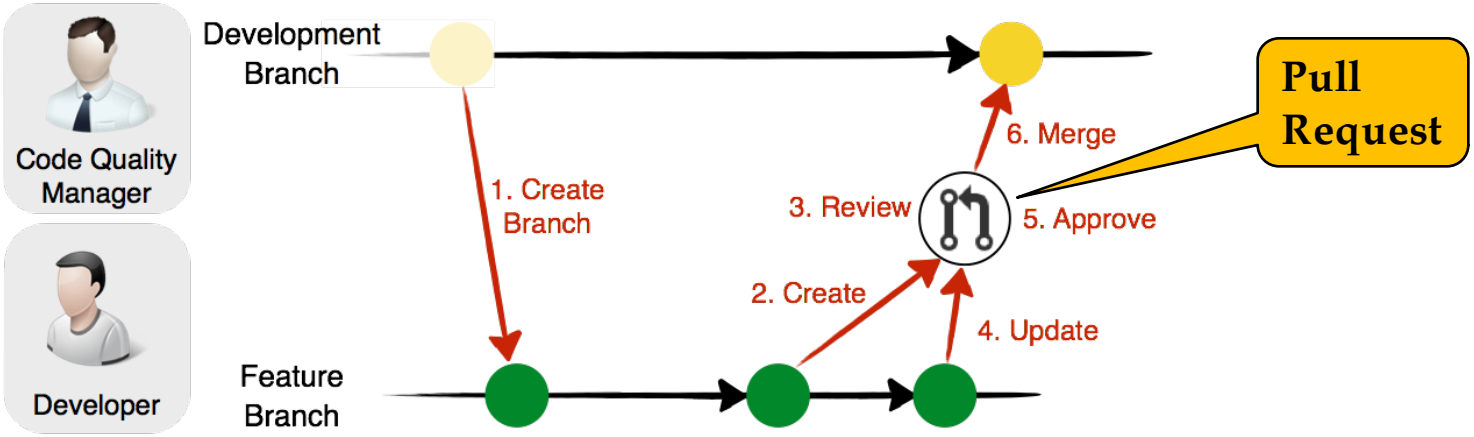
- The **Master Branch** contains external release candidates (Potential Product Increment)
- **Hotfix Branches** include revisions
- **Release Branches** contain external releases (Product Increment)
- The **Development Branch** contains internal release candidates
- **Feature Branches** address incremental developments and explorations



Source: http://nvie.com/posts/a-successful-git-branching-model

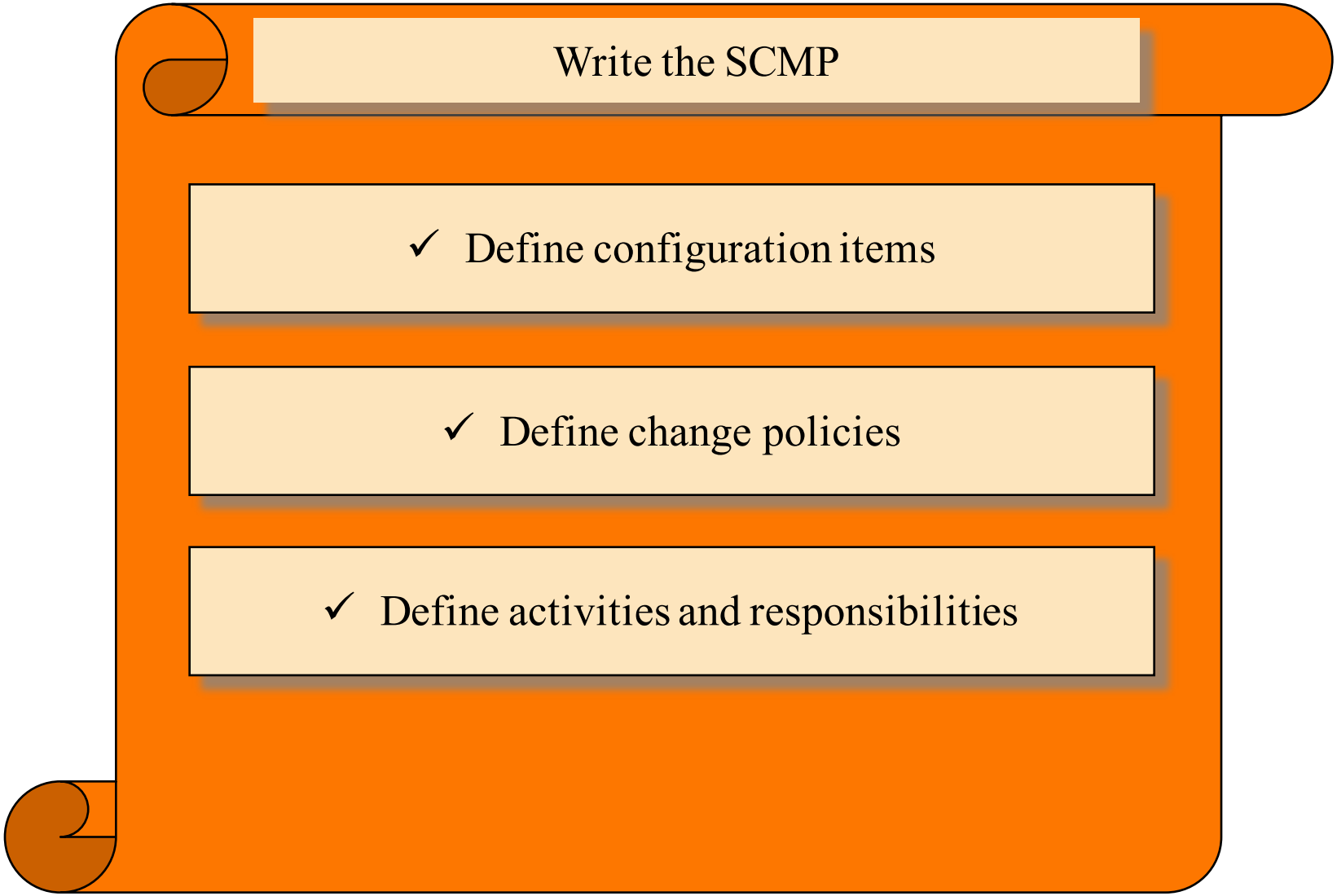
Merge Management with Pull Requests and Code Reviews

- Before the changes of a Feature Branch are merged (promoted) into the Development Branch, the code quality manager reviews the changes
- The Code Quality Manager only approves them if they follow specified quality requirements (e.g. regression tests have passed, coding guidelines are being followed)



→ More about this in Exercise 05 Branch and Merge Management

Tasks for Configuration Managers



Outline of the Lecture

- ✓Purpose of Software Configuration Management (SCM)
- ✓Some Terminology
- ✓Software Configuration Management Activities
- Outline of a Software Configuration Management Plan
- Build and Release Management
- Continuous Integration

Software Configuration Management 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 (SCMP)

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

Outline of a SCMP (IEEE 828-2012)

1. Introduction
   - Describes the Plan's purpose, scope of application, key terms, and references
2. SCM management (WHO?)
   - Identifies the responsibilities and authorities for managing and accomplishing the planned SCM activities
3. SCM activities (WHAT?)
   - Identifies all activities to be performed in applying to the project
4. SCM schedule (WHEN?)
   - Establishes required coordination of SCM activities with other activities in the project
5. SCM resources (HOW?)
   - Identifies tools and physical and human resources required for the execution of the Plan
6. SCM plan maintenance
   - Identifies how the Plan will be kept current while in effect

Tailoring the SCMP

- The IEEE standard allows quite a bit of flexibility for preparing the SCMP
- The SCMP can be
  - tailored upward:
    - to add information
    - to use a specific format
  - tailored downward
    - Some SCMP components might not apply to a particular project.
- Always state the reasons for diverting from the standard in the Introduction
- It is not possible to omit any of the six major classes of information.

Requirements for Build Management

- Large and distributed software projects need to provide a development infrastructure with an integrated build management that supports:
  - Regular builds from the master directory
  - Automated execution of tests
  - E-mail notification
  - Determination of code metrics
  - Automated publishing of the applications and test results (e.g. to a website)
- Tools for Build Management:
  - Unix's Make
  - Ant
  - Maven

Activities in Build Management

- The transition from source code to the executable application contains many mechanical (boring) activities:
  - Settings required paths and libraries
  - Compiling source code
  - Copying source files (e.g. images, sound files, start scripts)
  - Setting of file permissions (e.g. to executable)
  - Packaging of the application (e.g. zip, tar, dmg)
- Executing these steps manually is time-consuming and the chance of introducing failures is high
- Automating these steps has its origins in Unix

Outline of the Lecture

- ✓ Purpose of Software Configuration Management (SCM)
- ✓ Terminology
- ✓ Software Configuration Management Activities
- ✓ Outline of a Software Configuration Management Plan
- ✓ Build Management
- Continuous Integration
- Continuous Delivery

Reasons for Continuous Integration

- Risk #1: The later integration occurs in a project, the bigger is the risk that unexpected faults occur
- Risk #2: The higher the complexity of the software system, the more difficult it is to integrate its components
- Continuous integration addresses these risks by building as early as possible and frequently
- Additional Advantages:
  - There is always an executable version of the system
  - Team members have a good overview of the project status

Definition Continuous Integration

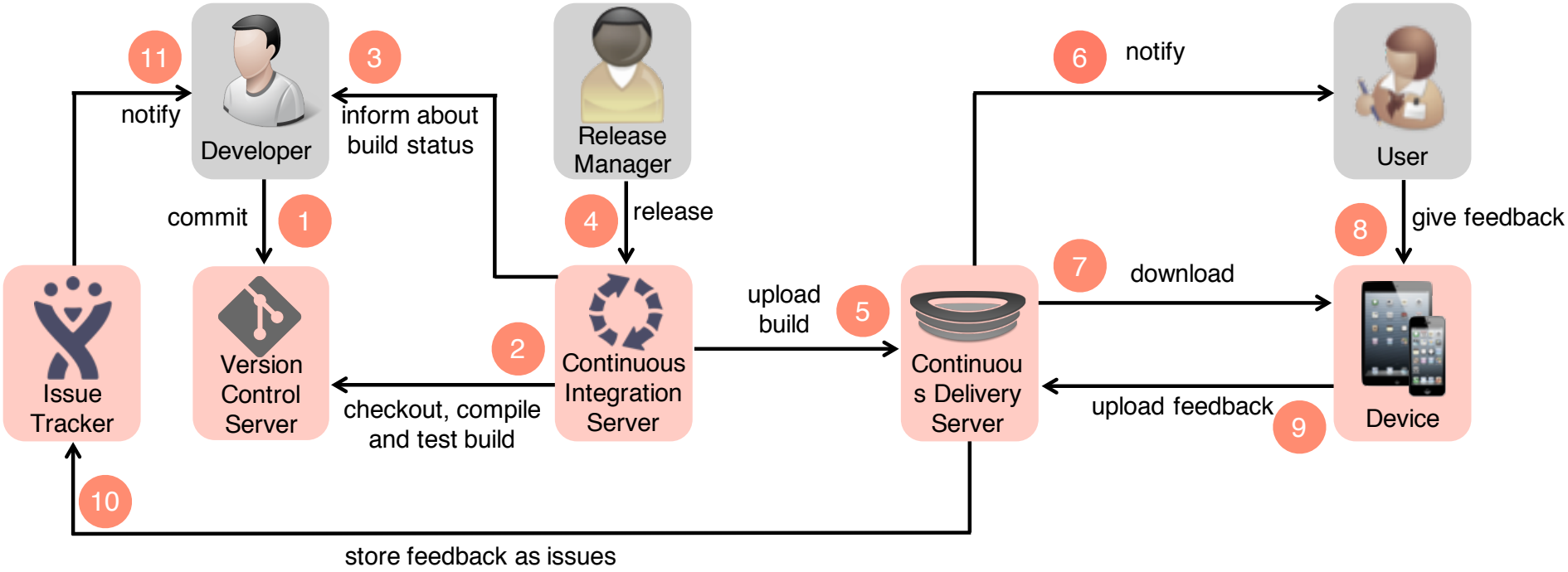
Continuous Integration: A software development method where members of a team *integrate* their work *frequently*, usually each person integrates at least daily, leading to multiple integrations per day.
Each integration is verified by an *automated build including the execution of tests* to detect integration errors as quickly as possible.

Continuous Integration can regularly answers these Questions

- Do all the software components work together?
- How much code is covered by automated tests?
- Where all tests successful after the latest change?
- What is my code complexity?
- Is the team adhering to coding standards?
- Where there any problems with the last deployment?

Continuous Delivery

- Definition: Continuous Delivery is a software development discipline where teams build software in such a way that the software (or any change to it) can be released to production at any time.



Sources:

1) Humble, Jez, and David Farley. Continuous delivery: reliable software releases through build, test, and deployment automation. Pearson Education, 2010.
2) Fowler, Martin. "Continuous Delivery." URL http://martinfowler.com/bliki/ContinuousDelivery.html (2013)
3) Stephan Krusche, Lukas Alperowitz, Bernd Bruegge and Martin Wagner, Rugby: An Agile Process Model Based on Continuous Delivery, 1st International Workshop on Rapid Continuous Software Engineering (RCoSE'14), ACM. Hyderabad - India, June 2014

Benefits of Continuous Delivery

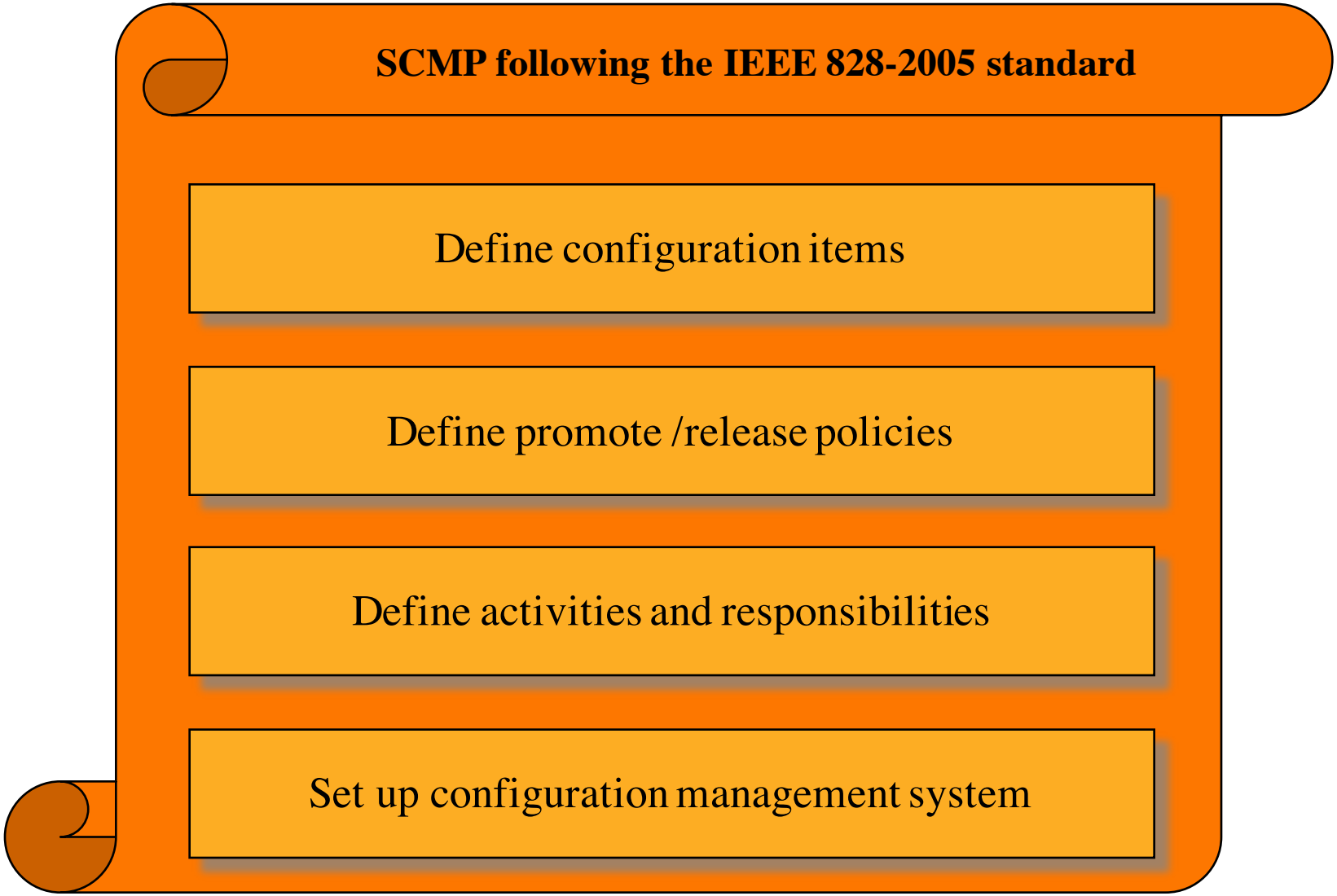
- **Accelerated Time to Market**: CD lets an organization deliver the business value inherent in new software releases to customers more quickly
- **Building the Right Product**: Frequent releases let the application development teams obtain user feedback more quickly
- **Improved Productivity and Efficiency**: Significant time savings for developers, testers, operations engineers, etc. through automation
- **Reliable Releases**: The risks associated with a release have significantly decreased, and the release process has become more reliable
- **Improved Product Quality**: The number of open bugs and production incidents has decreased significantly
- **Improved Customer Satisfaction**: A higher level of customer satisfaction is achieved

Source: Chen, Lianping (2015). "Continuous Delivery: Huge Benefits, but Challenges Too". IEEE Software 32 (2): 50

Summary

- Software Configuration Management:
  - A set of management disciplines within a software engineering process to develop a *baseline*
- Promotions and Releases
- A SCMP needs to be tailored to meet the project requirements
- Git allows lightweight branching and merging
- Continuous Integration and Continuous Delivery are emerging as central aspects in modern software projects

Summary: Tasks for Configuration Managers



References

- IEEE Standards (PDF in Moodle)
  - http://standards.ieee.org/findstds/standard/828-2012.html
- Version Control Systems
  - Subversion: http://subversion.tigris.org
  - Git: http://git-scm.com
  - Atlassian Stash: Git Enterprise Manager http://www.atlassian.com/software/stash
- P.M. Duvall, S. Matyas, A. Glover, Continuous Integration: Improving Software Quality and Reducing Risk, Addison-Wesley, 2007.
- Humble, Jez, and David Farley. Continuous delivery: reliable software releases through build, test, and deployment automation. Pearson Education, 2010.
- Fowler, Martin. "Continuous Delivery" http://martinfowler.com/bliki/ContinuousDelivery.html (2013)
- Stephan Krusche, Lukas Alperowitz, Bernd Bruegge and Martin Wagner, Rugby: An Agile Process Model Based on Continuous Delivery, 1st International Workshop on Rapid Continuous Software Engineering (RCoSE'14), ACM. Hyderabad - India, June 2014
- Chen, Lianping (2015). "Continuous Delivery: Huge Benefits, but Challenges Too". IEEE Software 32 (2): 50