Managing Changes, Risks, and Quality

Goals of the Unit

- Requests for changes and changes will occur in your project
- The goal of this unit is understanding:
  - The importance of keeping a project under scope
  - How request for changes can positively or negatively influence your project
  - The techniques to manage changes

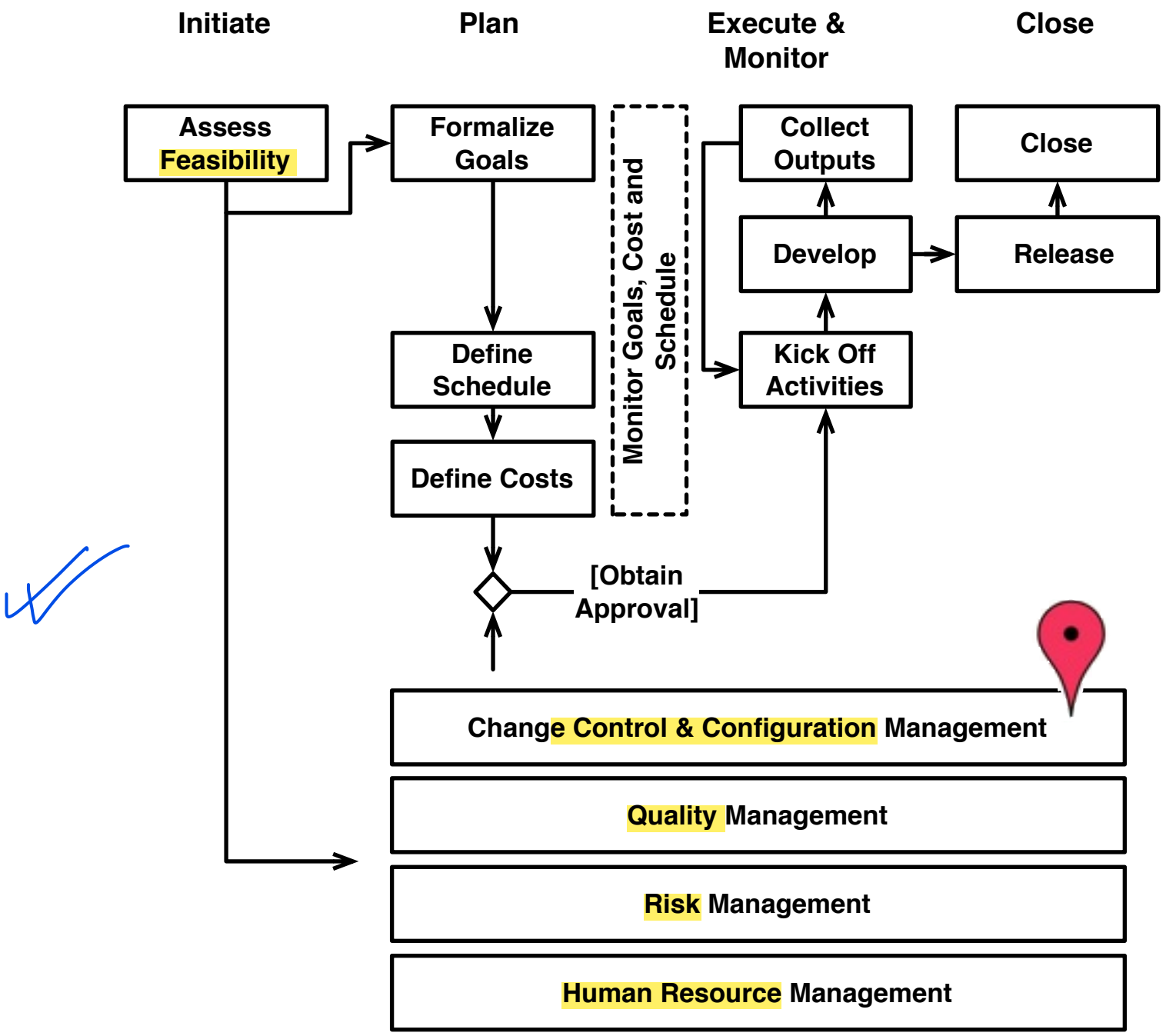
The Framework

- The scope document formalizes the goals of a project
- Ideally, once the goals are fixed, the project should move on to the design/implementation phase and achieve the project goals, through a progressive refinement
- Any deviation from such course of action is a perturbation (it changes goals, plans, costs, outputs, work to be performed, …)
- Changes, however, are inevitable
- The goal of a sound project management, therefore, is ensuring that the change process is properly managed

Fundamental Concepts

- **Change Control** is the set of practices to ensure request for changes are properly taken care of
- **Configuration Management** is the set of practices to ensure project outputs remain coherent over time

- Change Control and Configuration Management span over the lifecycle of the project outputs
- In software projects artifacts are extremely simple to change (e.g., editing a file)
- In software projects, connection with bug reporting/bug lifecycle

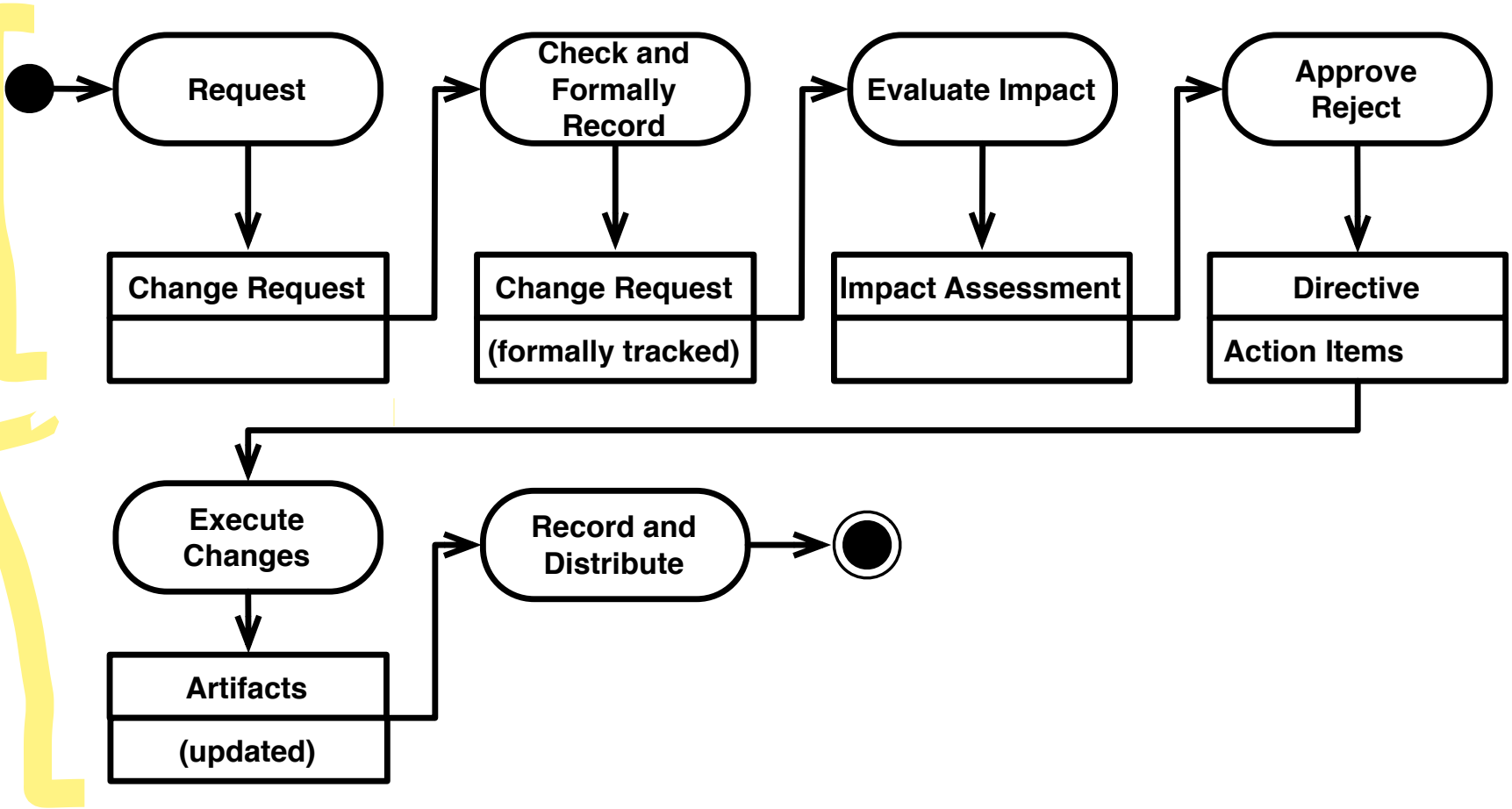

Initiate
Plan
Execute & Monitor
Close
Assess Feasibility
Formalize Goals
Define Schedule
Define Costs
Monitor Goals, Cost and Schedule
Collect Outputs
Develop
Kick Off Activities
Close
Release
[Obtain Approval]
Change Control & Configuration Management
Quality Management
Risk Management
Human Resource Management

Change Control

Causes of Request for Changes

- **Incompleteness or incoherencies** in the project requirements or in the description of work
- A **better comprehension of the system to be developed**
- A **technical opportunity**
- A **technical challenge**
- A **change in the external environment**
- **Non-compliance** of a project deliverable

A Change Control Process



- It runs in parallel to the other PM activities throughout the project

Comments

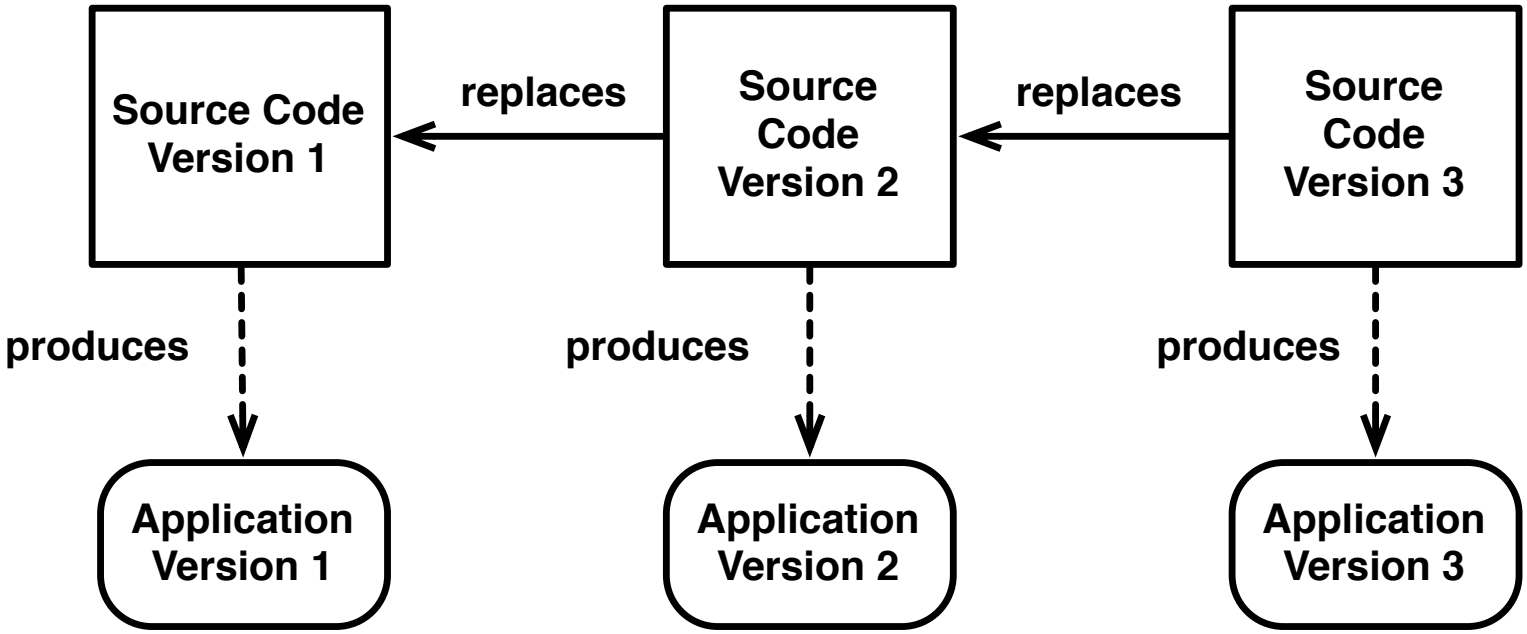
- A **change control board** might be appointed to approve/reject changes
- The cost and risk of changes increase as the project moves to the delivery
- The process ensures a formal record is kept and and a clear procedure is set to evaluate the impact of changes
- Change and change management is embraced by agile methodologies (changes “treated” as requirements)

Software Evolution Models

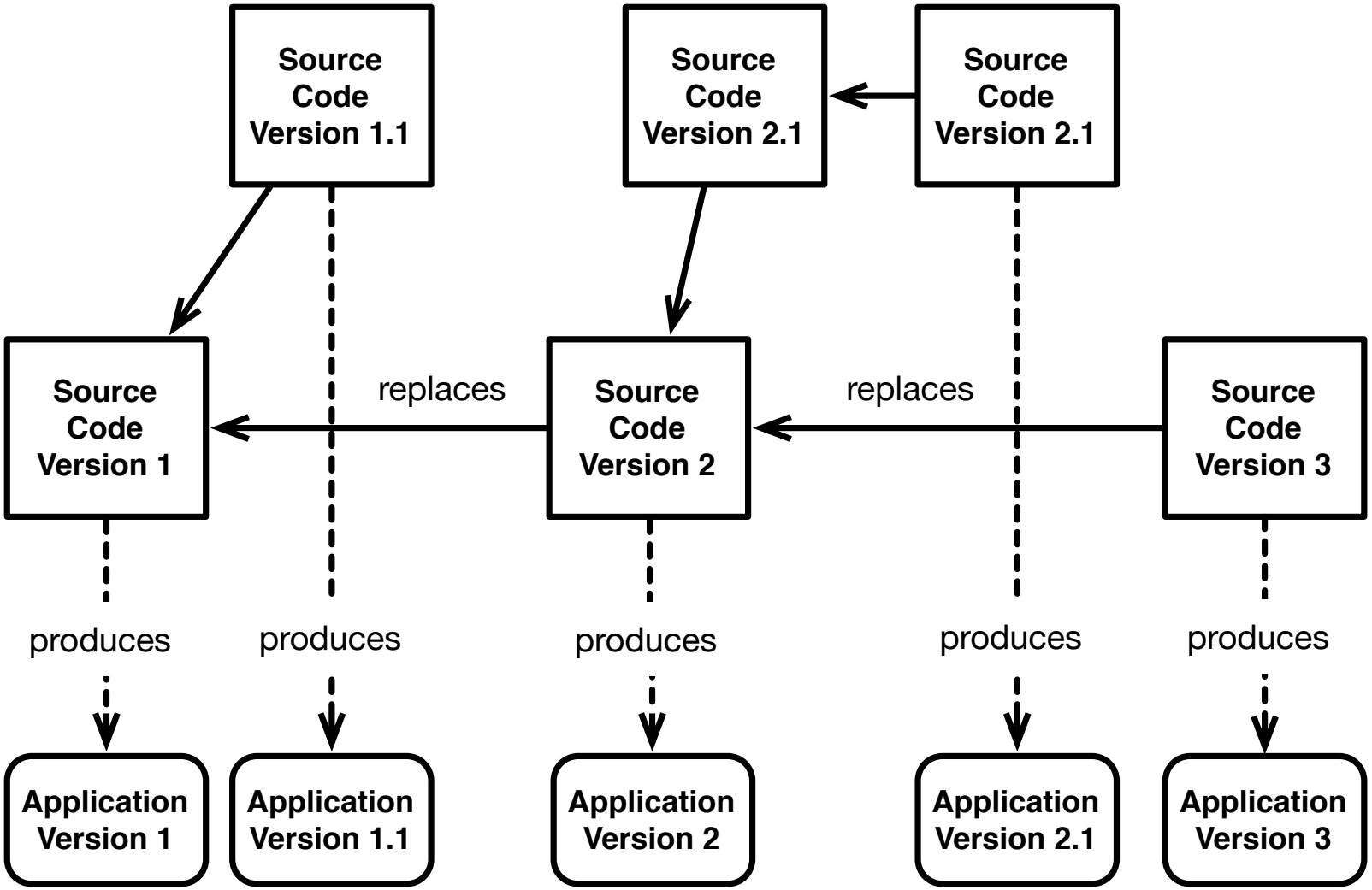
What makes a Software System

- Software systems are made of many different artifacts (sources, libraries, external libraries, documentation, conversion scripts, databases)
- Software systems run in many different configurations (e.g., base/pro, versions 1 and 2, Linux/OSX/Windows)
- Two sources of complexity need to be addressed to develop or maintain a software product:
  - Identification of the artifacts
  - Evolution

Linear Development Model



Branching Development Model



Software Development Models

- **Linear development:**
  - Only one version of an application is running at any given time (Example: one-offs; many web applications are one-offs)
- **Branching development:**
  - Various versions of an application are running at a given time

Configuration Management

Configuration Management

Configuration Management (CM) is a set of activities running in parallel to the development process, whose goal is **establishing** and **maintaining** system's coherency over time

- Part of the project management plan
- Helps define project standards and best practices

Configuration Management Main Goals

- Being able to build a system from a consistent set of components
- Being able to retrieve a software component when needed (consider: storage time, storage means)
- Being able to view the history of changes a system has undergone
- Being able to retrieve a previous version of a system

- Remark: closely related to the change management process

Some Examples

- A bug is reported by a user on a COTS software we have been selling for ten years.
- A client requests an enhancement to a one-off system we sold in 2005.
- We need to reproduce/understand an odd behavior of the control software of a space exploration probe which is now orbiting Jupiter

Steps and Tools: Establish Baseline

- The first step is “establishing what a product is”
- A good CM requires to:
  - Clearly identify the items which constitute a product
  - Identify the relationships among these items
  - Choose an appropriate identification and numbering scheme for versions
  - Take “snapshots”: baseline records

Steps and Tools: Manage Changes

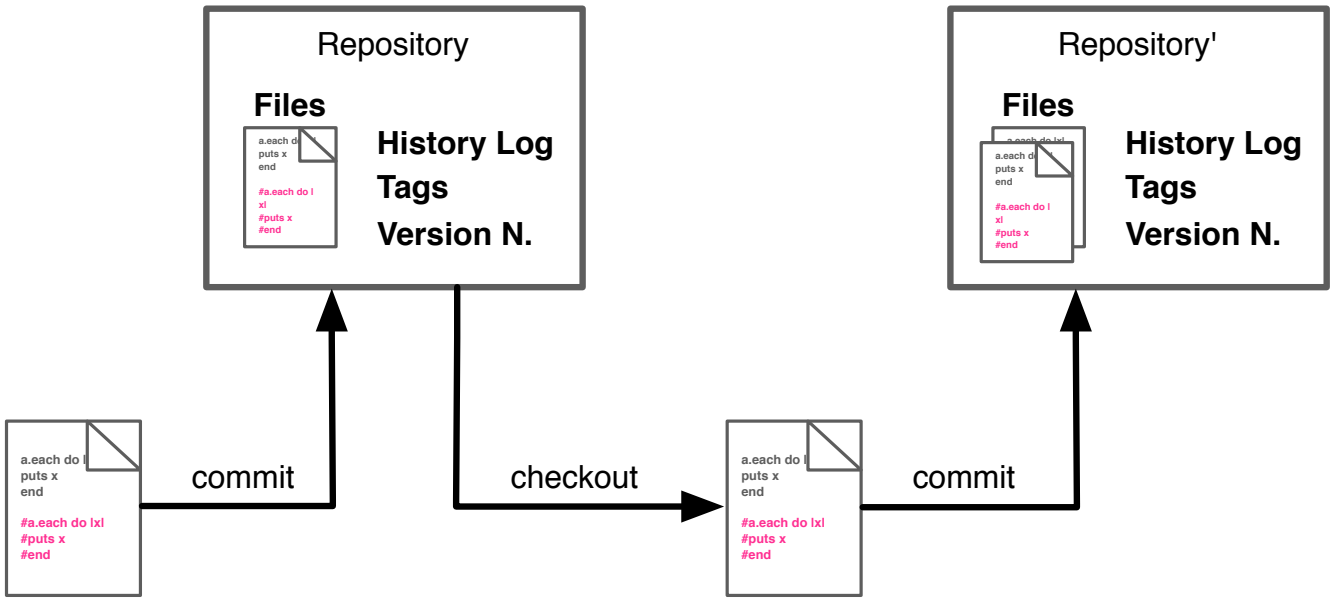
- The second step is “maintaining coherency over time”
- A good CM process requires to:
  - Define the “baseline record” (the starting point)
  - Identify and approve requests for changes (see change control)
  - Formally record changes and history of each item
  - Maintaining old versions
- For family of products there could be different baselines. Changes might need to be applied to one or more baseline (consider a security fix to a browser)

Steps and Tools: Considerations

- For software development, a **version control system** implements various of the functions described above
- Tools are not sufficient: an adequate process has to be in place
- Semantic versioning is an example of numbering schema

Version Control Systems: Main Concepts

- **Working version:** the file (or set of files we are currently editing)
- **Repository:** the storage where all versions of a file (or set of files) are kept together with additional information



Version Control Systems: Main Concepts

- In the simple case (early VCS) each file would have an independent repository
- Coherence is kept by assigning the same tags to all artifacts constituting a baseline
- More recent VCS manage sets of artifacts in an integrated way
- Tagging is used to mark important baseline records
- A VCS typically support parallel access and editing of artifacts