

# CS5351

## 2024-2025 Semester B

### *Appendix: Software Testability*

---

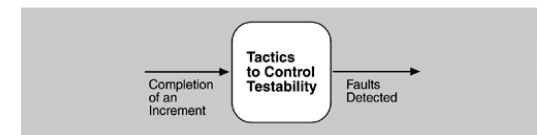
Prof. Zhi-An Huang

Email: `huang.za@cityu-dg.edu.cn`

# Testability

---

- ◆ **Testability** is the ease with which software can be made to demonstrate its faults through testing
- ◆ We want to increase our chance to observe the failure from the next test case
- ◆ It relies on two concepts
  - **Controllability**: control a component's input and preferably the execution internal to the component
  - **Observability**: observe the outputs (and its internal states)

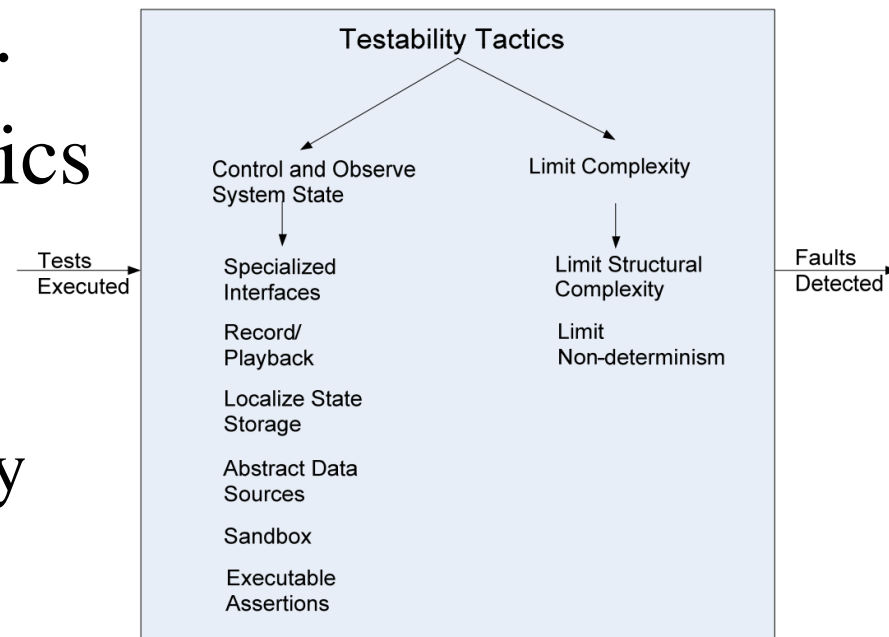


# Testability General Scenario

Source	Unit testers, integration testers, system testers, acceptance testers, end users, either running tests manually or using automated testing tools
Stimulus	A set of tests are executed due to the completion of a coding increment such as a class, layer or service; the completed integration of a subsystem; the complete implementation of the system; or the delivery of the system to the customer.
Environment	Design time, development time, compile time, integration time, deployment time, run time
Artifacts	The component of the system being tested
Response	One or more of the following: execute test suite and capture results; capture activity that resulted in the fault; control and monitor the state of the component
Response Measure	One or more of the following: effort to find a fault or class of faults, effort to achieve a given percentage of state space coverage; probability of fault being revealed by the next test; time to perform tests; effort to detect faults; length of longest dependency chain in test; length of time to prepare test environment; reduction in risk exposure

# Goal of Testability

- ◆ Allow for easier testing when an increment of software development has completed.
- ◆ Two basic categories of tactics
  1. Add controllability and observability to the system
  2. Limit the system complexity



# Category 1: Controllability and Observability

---

- ◆ **Specialized Interfaces:** to control or capture variable values for a component either through a test harness or through normal execution.
- ◆ **Record/Playback:** capturing information crossing an interface and using it as input for further testing.
- ◆ **Abstract Data Sources:** Abstracting the interfaces lets you substitute test data more easily
- ◆ **Sandbox:** isolate the system from the real world to enable experimentation that is unconstrained by the worry about having to undo the consequences of the experiment.
- ◆ **Executable Assertions (or self-testing):** assertions are (usually) hand coded and placed at desired locations to indicate when and where a program is in a faulty state.

# Category 2: Limiting Complexity

---

- ◆ **Limit Structural Complexity:** avoiding or resolving cyclic dependencies between components, isolating and encapsulating dependencies on the external environment, and reducing dependencies between components in general.
- ◆ **Limit Non-determinism:** finding all the sources of non-determinism, such as unconstrained parallelism, and weeding them out as far as possible.

*CS5351 Software Engineering*  
*2024/2025 Semester B*

# Software Maintenance

---

Prof. Zhi-An Huang

Email: `huang.za@cityu-dg.edu.cn`

# Software Maintenance

---

- ◆ The last lifecycle phase of software development lifecycles
- ◆ Account for 40-70% of the budget cost
- ◆ Meaning: product view vs. process view
  - Modification of software product after delivery (IEEE)
  - Activities performed when the software undergoes modifications (ISO)



# *Reasons for Maintenance*

---

- ◆ Correct errors
- ◆ Changes in user requirements
- ◆ Changes in hardware/software environments
- ◆ Improve efficiency
- ◆ Optimize the code
- ◆ Eliminate unwanted effects
- ◆ ...

# Software Maintenance

---

- ◆ Four major types of software maintenance
  - Adaptive: change in the data/processing environment
  - Perfective: new user requirements/better “-lities”
  - Corrective: fixing functional/non-functional bugs
  - Preventive: prevent problems in the future
  
- ◆ We will see other types in the next two slides

*Old days*

} 75%

} 21%

# Exemplified Mapping in the literature

Table I. Approximate correspondence between definitions of types. Correspondence with the Chapin *et al.* definitions exists only when objective evidence confirms the activity. NI—not included; (All)—implicit in all included activities.

Evidence-based (Chapin <i>et al.</i> )		Intention-based definitions			Activity-based definitions	
Cluster	Type	Swanson [8]	IEEE [14,23]	ISO/IEC 14764 [27]	Kitchenham <i>et al.</i> [3]	ESF/EPSON [33]
Support interface	Training	NI	NI	(All)	NI	User support
	Consultive	NI	NI	(All)	NI	User support
	Evaluative*	NI	NI	(All)	(All)	(All)
Documentation	Reformative	Perfective	Perfective	Perfective	(All)	Preventive
	Update*	Perfective	Perfective	Perfective	(All)	Preventive
Software properties	Groomative	Perfective	Perfective	Perfective enhancement	Enhancements	Perfective
	Preventive	Perfective	Perfective, or Preventive	Preventive, or Perfective enhancement	Preventive <sup>‡</sup>	Anticipative or Preventive
	Performance	Perfective	Perfective	Perfective enhancement	Corrections, or Implementation change	Anticipative or Perfective
	Adaptive*	Adaptive	Adaptive	Perfective enhancement	Changed existing requirements <sup>‡</sup>	Anticipative or Adaptive
Business rules	Reductive	Perfective	Perfective	Perfective enhancement	Changed existing requirements <sup>‡</sup>	Evolutionary
	Corrective	Corrective	Corrective	Corrective	Corrective	Corrective
	Enhance*	Perfective	Perfective	Perfective enhancement	New requirements <sup>‡</sup>	Evolutionary

\*Default type within the cluster.

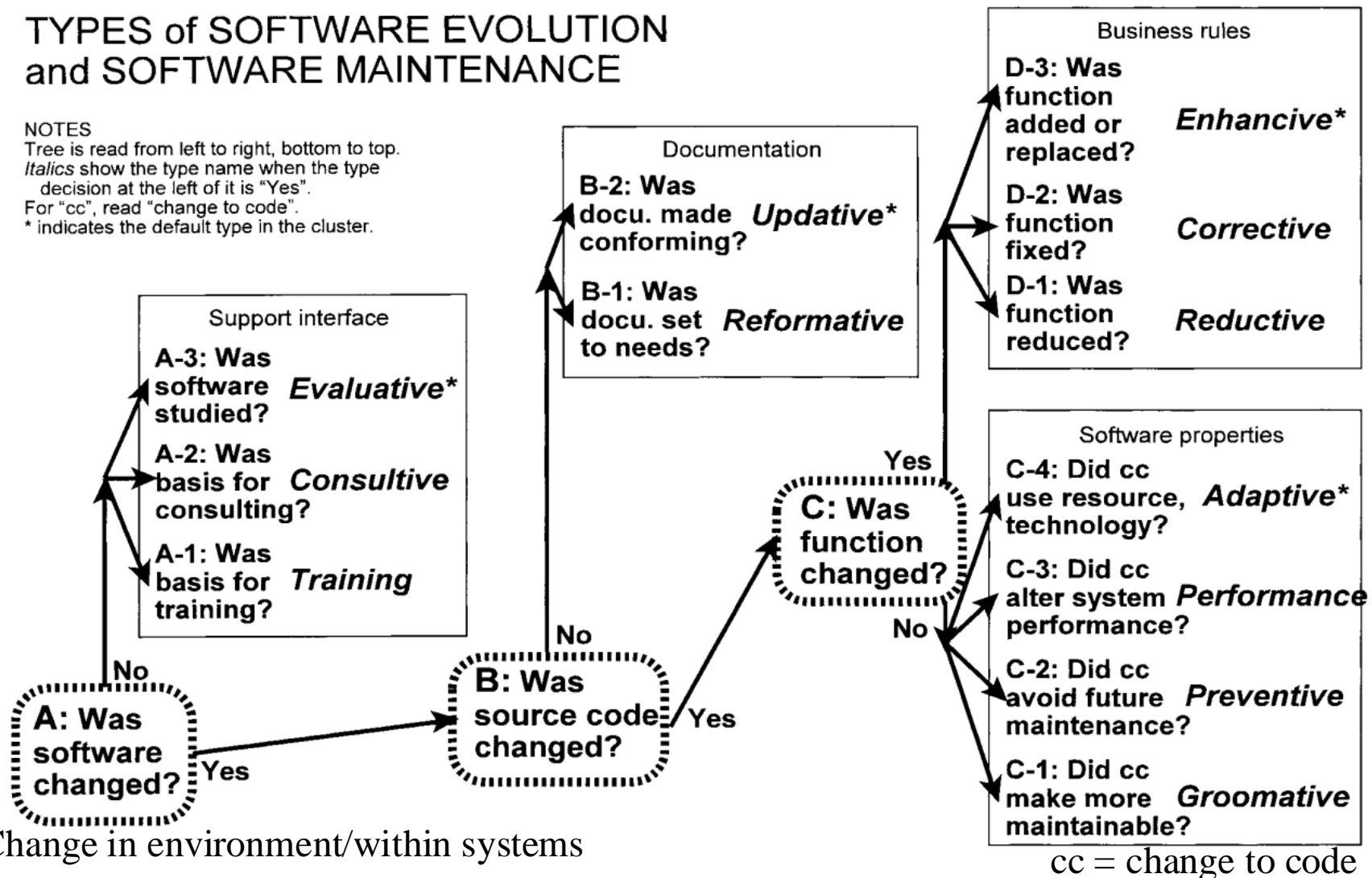
<sup>‡</sup>A subtype of Enhancements.

# Finer classification of maintenance type

## TYPES of SOFTWARE EVOLUTION and SOFTWARE MAINTENANCE

### NOTES

Tree is read from left to right, bottom to top.  
*Italics* show the type name when the type decision at the left of it is "Yes".  
For "cc", read "change to code".  
\* indicates the default type in the cluster.



# Problems in Maintenance

---

- ◆ Code written by others (not us!)
- ◆ We do not fully understand the program before making changes
- ◆ Code is not necessarily well-structured
- ◆ High staff turnover rate
- ◆ We could not access the information in the mind of the coder when we maintain the code
- ◆ Systems are not designed for change

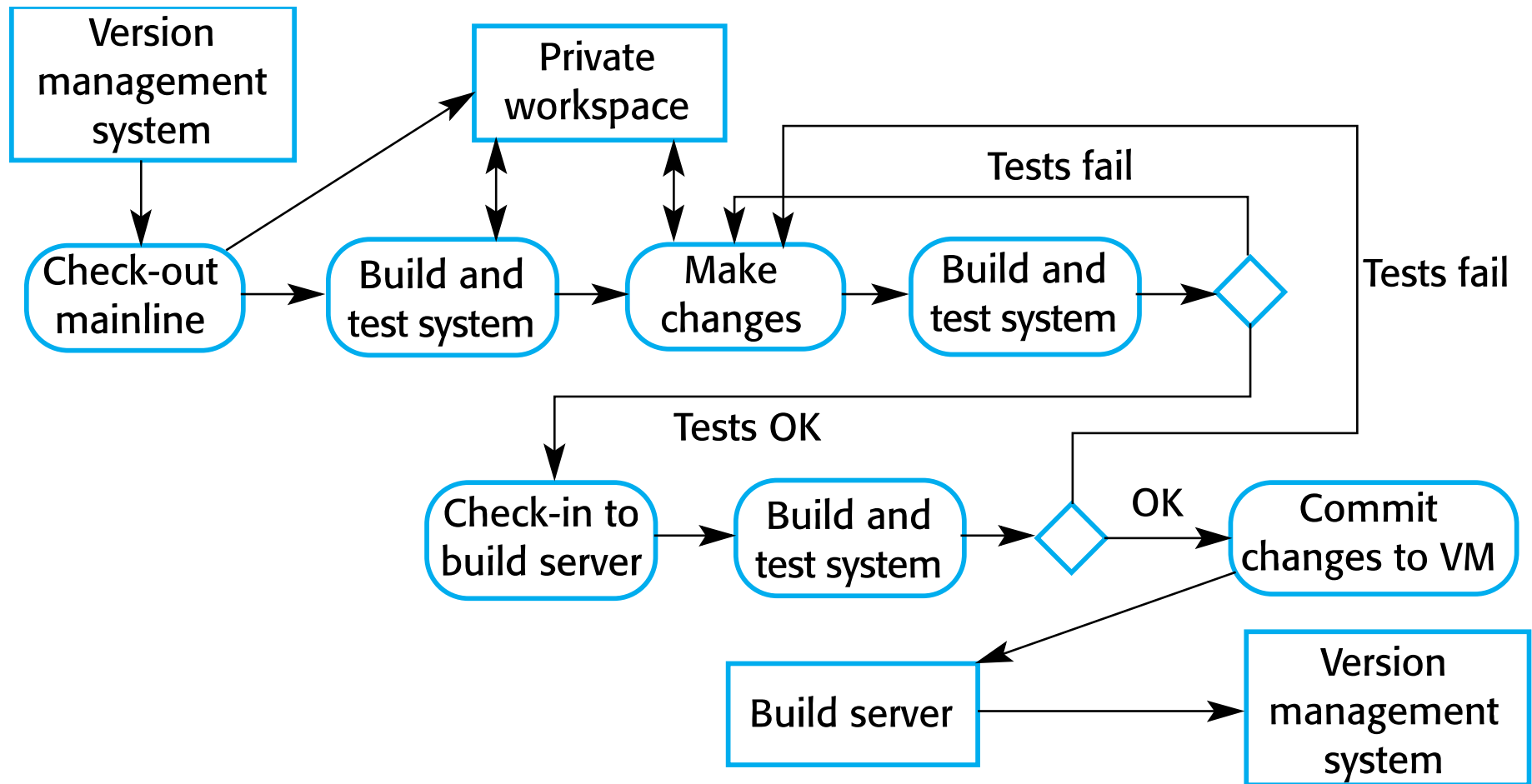
# Procedure of Maintenance

---

1. Determine the maintenance objective
    - Find the type of maintenance
  2. Understand the program to change
  3. Prepare a change proposal
  4. Conduct a change review (like a design review)
  5. Make changes
  6. Validation of changes
- Now, it is often a part of continuous integration

In fact, the procedure of maintenance is absorbed in an Agile process as the process for a typical task

# Continuous integration



# Version management

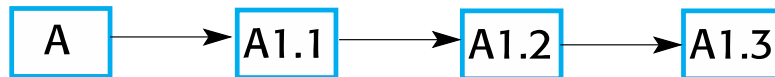
---

- ◆ Version Management is the process of keeping track of different versions of software components or configuration items and the systems in which these components are used.
- ◆ We need to track the evolution of codelines and baselines.
  - Codeline: a sequence of versions of a piece of code
  - Baseline: a definition of a system (e.g., for a release)
    - = a set of versions, each taking from a codeline



# Codelines and baselines

Codeline (A)



Codeline (B)



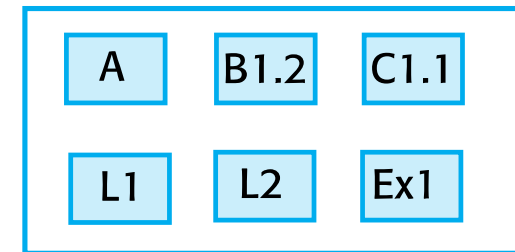
Codeline (C)



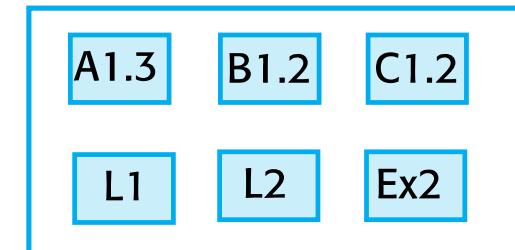
Libraries and external components



Baseline - V1



Baseline - V2

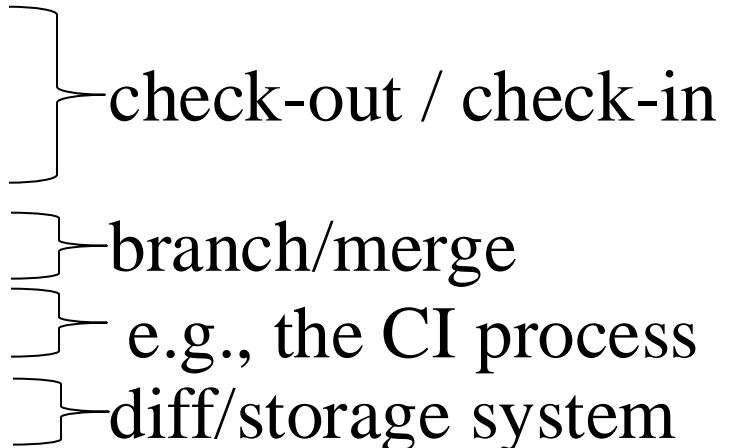


Mainline

# Version management systems

---

- ◆ Version management systems (VMSs) identify, store, and control access to the different versions of components.
  - Centralized vs. Distributed systems vs. cloud-based
- ◆ Key features
  - Version and release identification
  - Change history recording
  - Support for independent development
  - Project support
  - Storage management



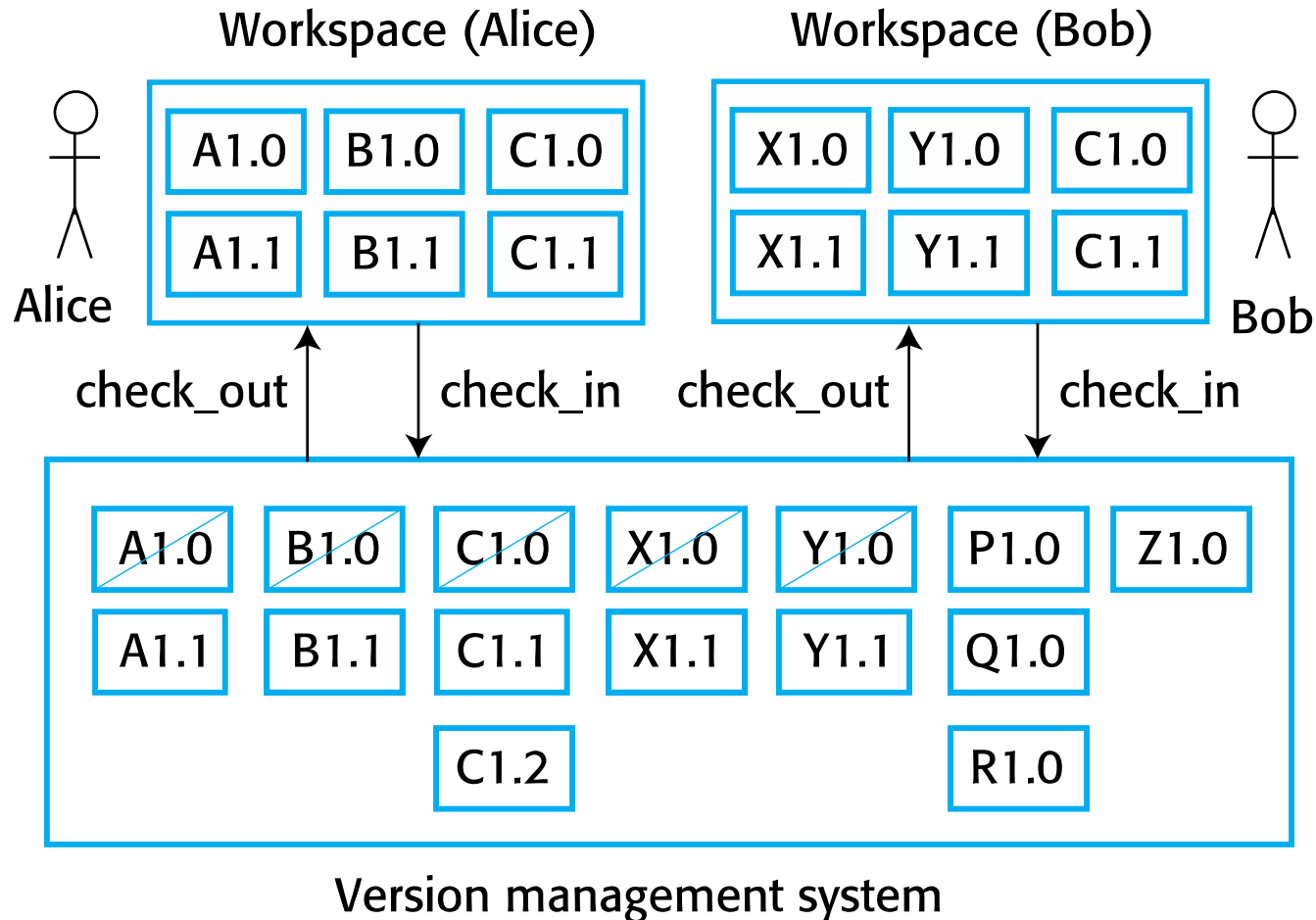
} check-out / check-in

} branch/merge

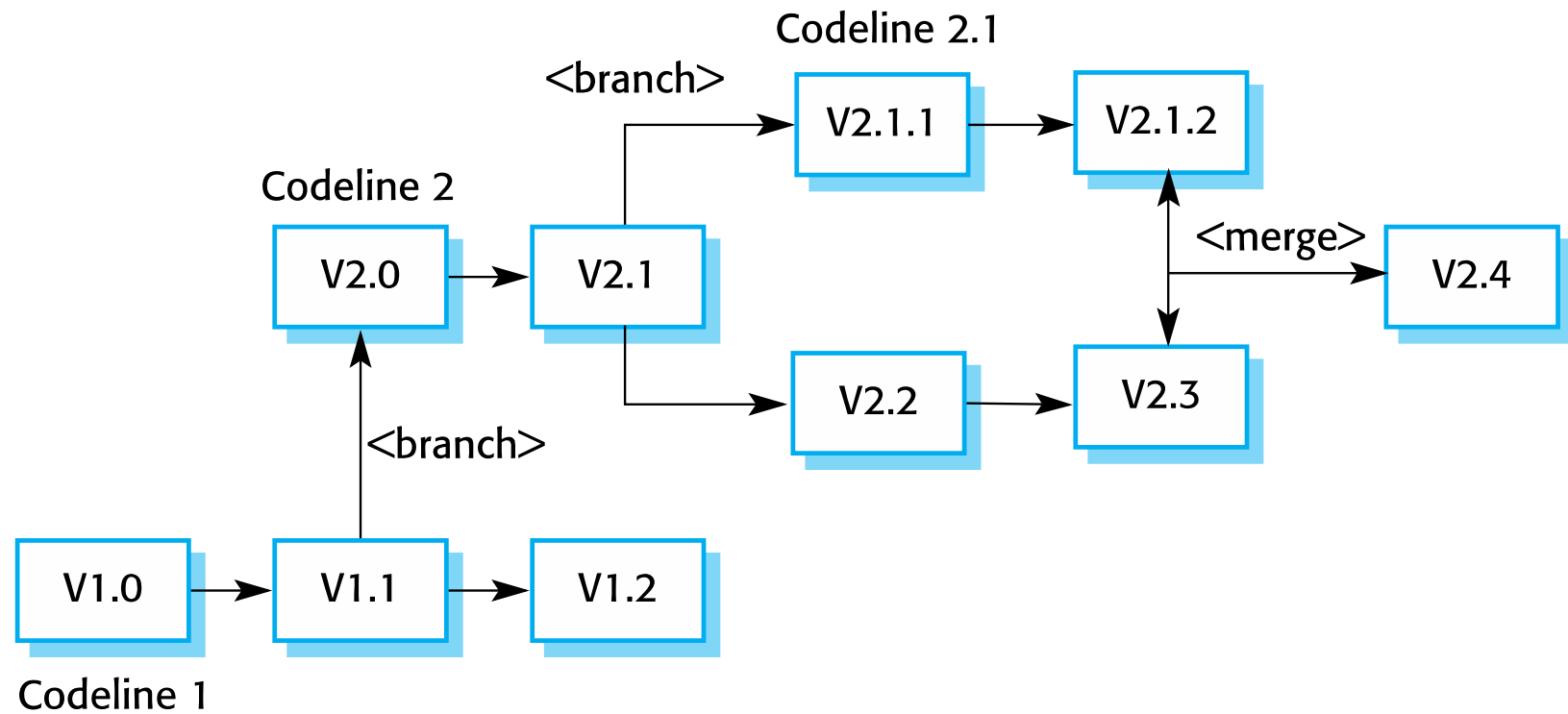
} e.g., the CI process

} diff/storage system

# Check-in/Check-out

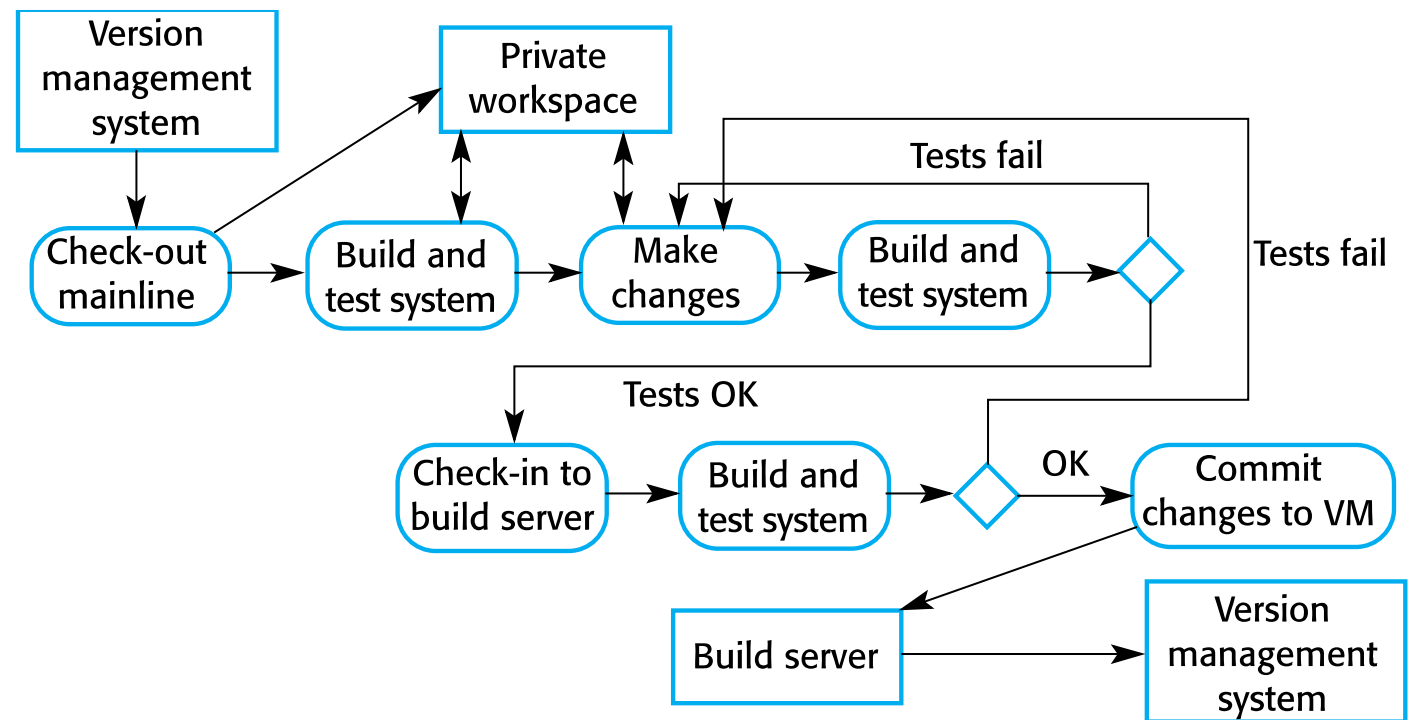


# Branching and merging

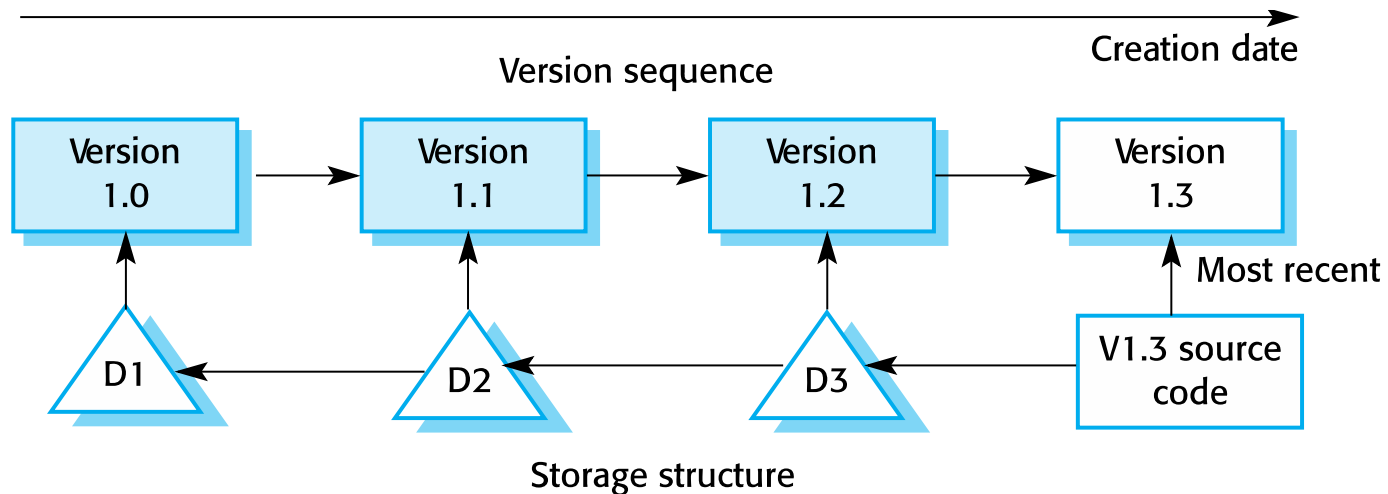


# Project Support

- ◆ E.g., The automated build and test processes in CI require additional baseline retrievals



# Storage management using “Diff”



# Storage management in Git

---

- ◆ Git does not use deltas (“diffs”).
- ◆ Compression and Decompression
  - It runs a standard compression algorithm to compress the stored files and their associated meta-information.
  - Retrieving a file involves decompressing it, with no need to apply a chain of operations.
- ◆ Deduplication
  - It does not store duplicate copies of files.
- ◆ Packing
  - Git also uses the notion of packfiles where several smaller files are combined into an indexed single file. (for faster access)