哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

Chapter 2: Process and Tools of Software Construction

2.1 Software Lifecycle and Configuration Management

Ming Liu

February 24, 2019

Outline

- **Software Development Lifecycle (SDLC)**
- **Traditional software process models (waterfall, incremental, V-model, prototyping, spiral)**
- **Agile development and eXtreme Programming (XP)**
- **Collaborative software development**
- **Software Configuration Management (SCM)**
- **Git as a SCM tool**
- **Summary**

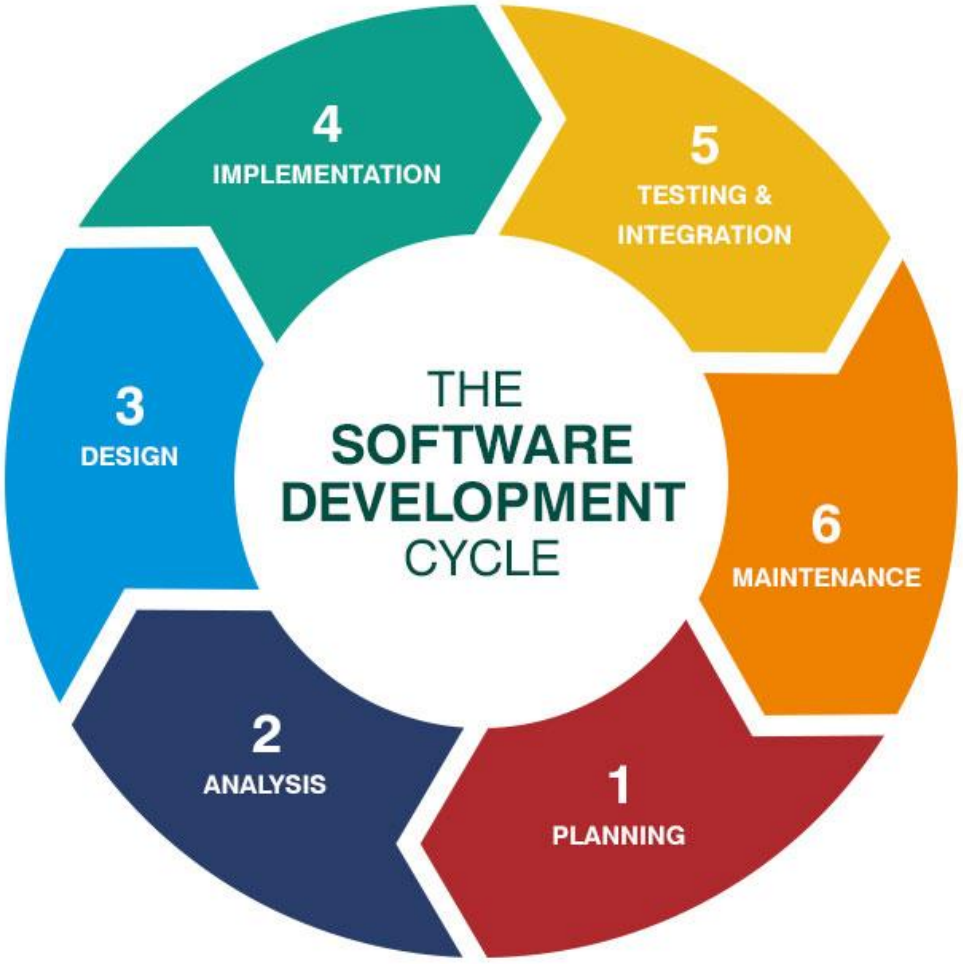
Objectives of this lecture

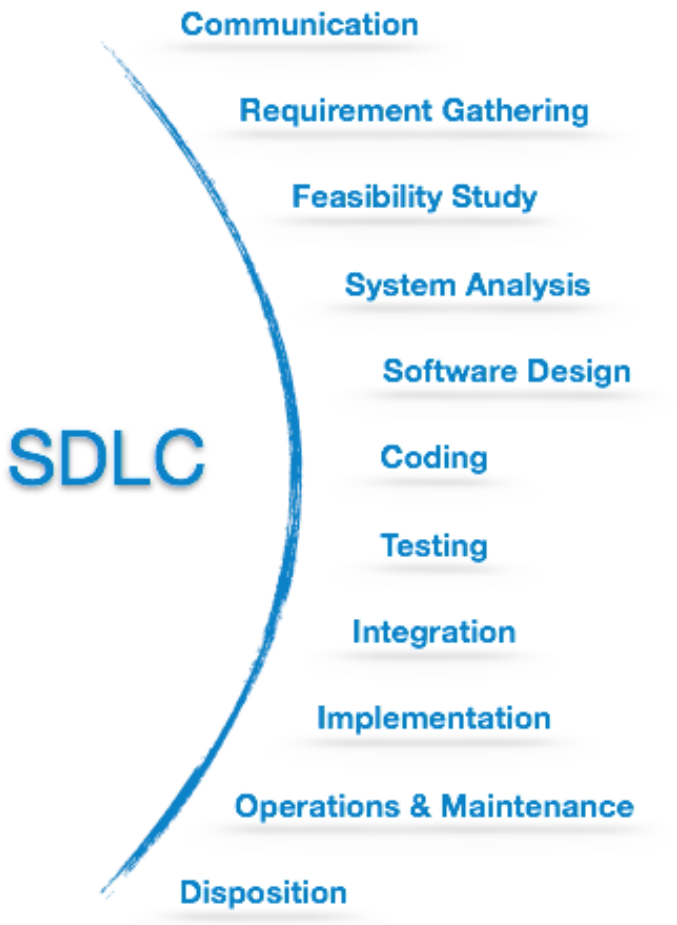
- **To know the general process of software development**
- **To understand the philosophy of traditional software process models including linear and iterative models (waterfall, incremental, prototyping, spiral, and V-model)**
- **To know and make practice of Agile development**
- **To understand Software Configuration Management (SCM)**
- **To learn how to use Git for daily SCM tasks (basic commands for personal dev., advanced commands for collaborative dev.)**

1 Software Development Lifecycle (SDLC)

Lifecycle of a software

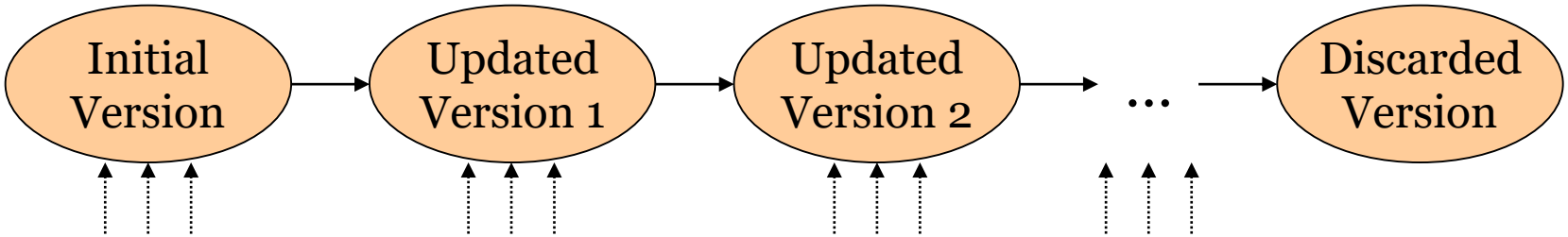
- **Software Development Life Cycle (SDLC): From Zero to One**

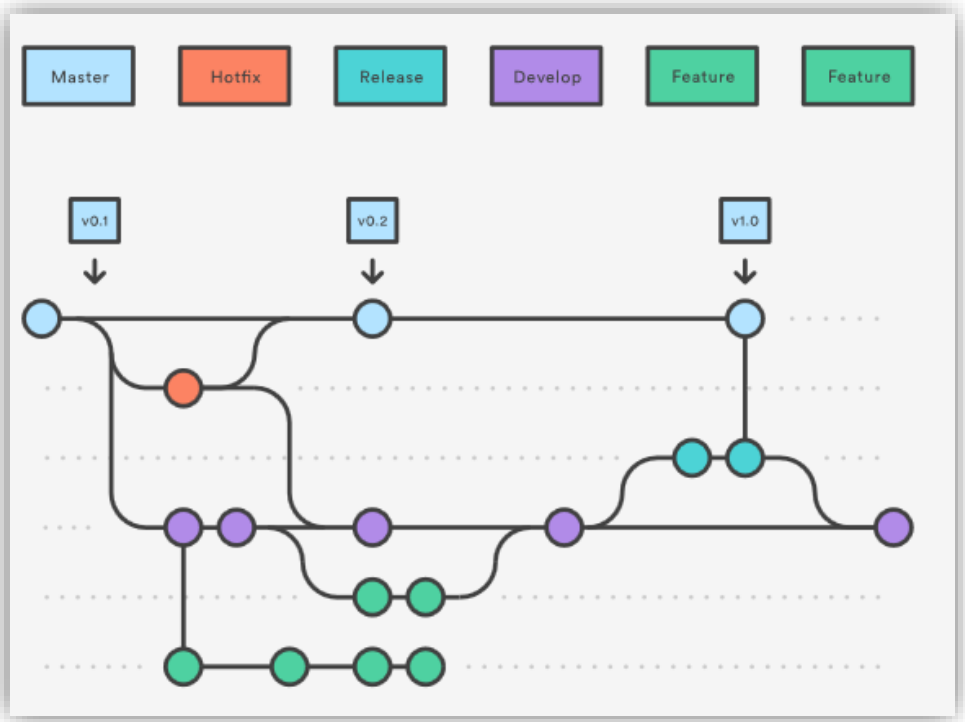


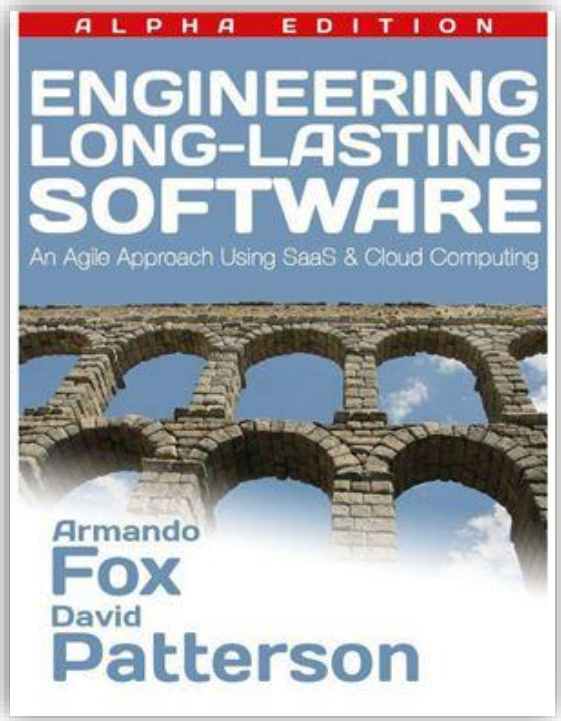


Lifecycle of a software

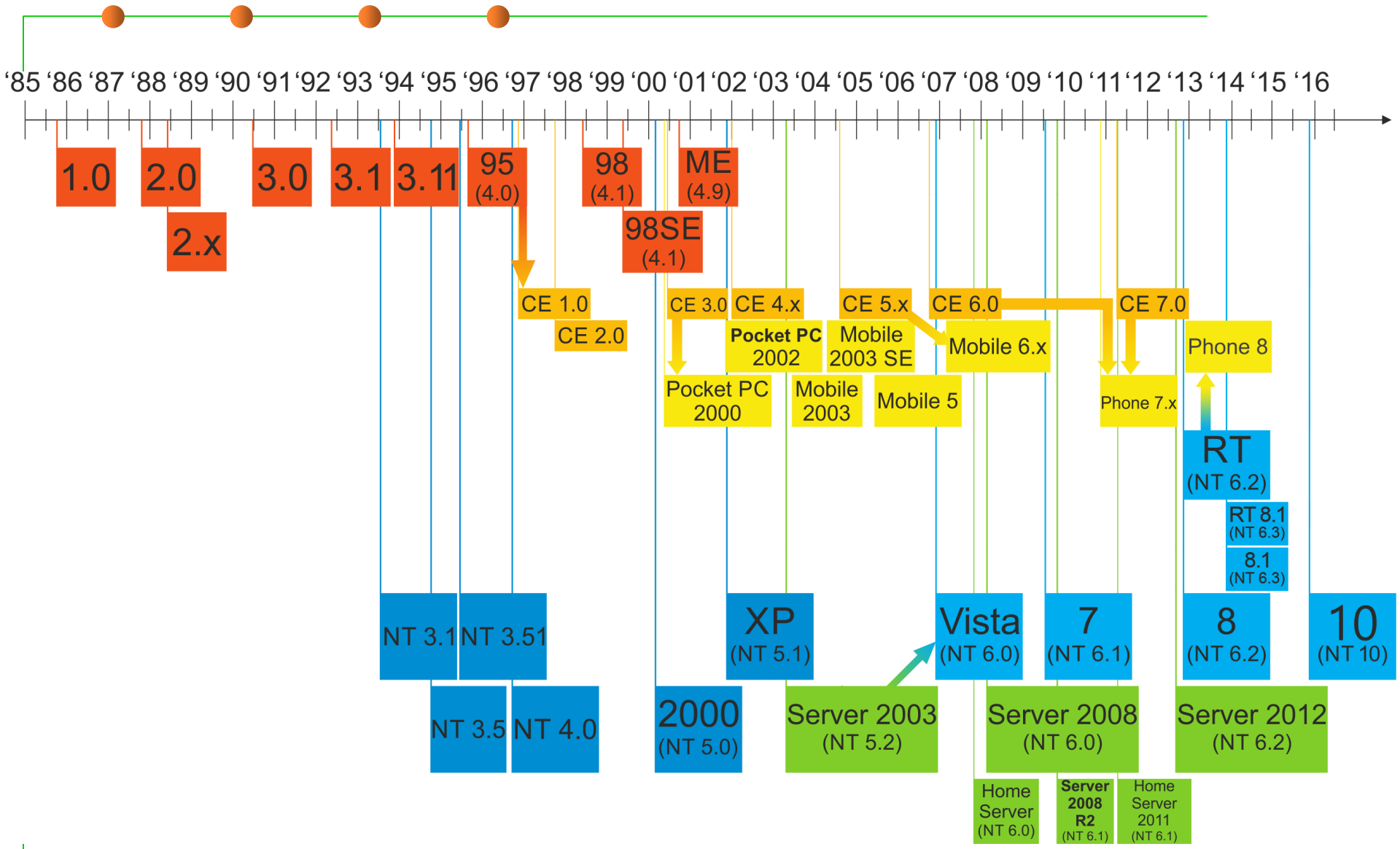
- **Multiple versions in the life of a software: From 1 to *n***







Example 1: Microsoft Windows (1985-2016)



"Is Software Alive?"

- **Sure! Any software has its own life.**
  - "Age" of a software: how long has it been produced and used?
  - "Vitality" of a software: at a particular time, to what degree is it welcome by the market and users?
- **Expectation:**

Long-lasting and full of vitality at any time

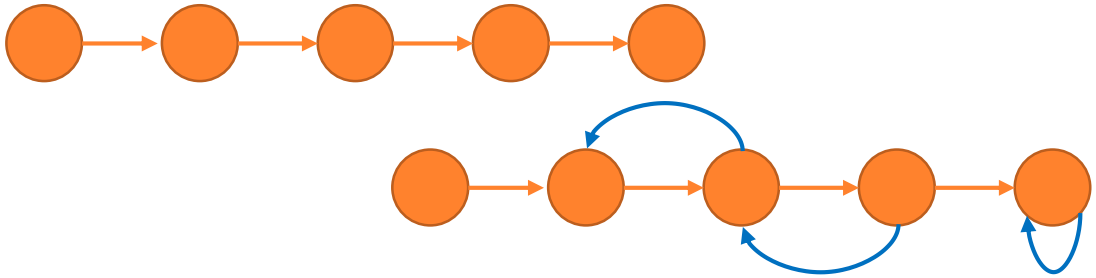
활력

- **However,**
  - Failed software development (from zero never to 1)
  - Full of faults/errors during execution
  - Software aging / decay (less validity)
  - The DEATH of a software

2 Traditional Software process models

Traditional software process models

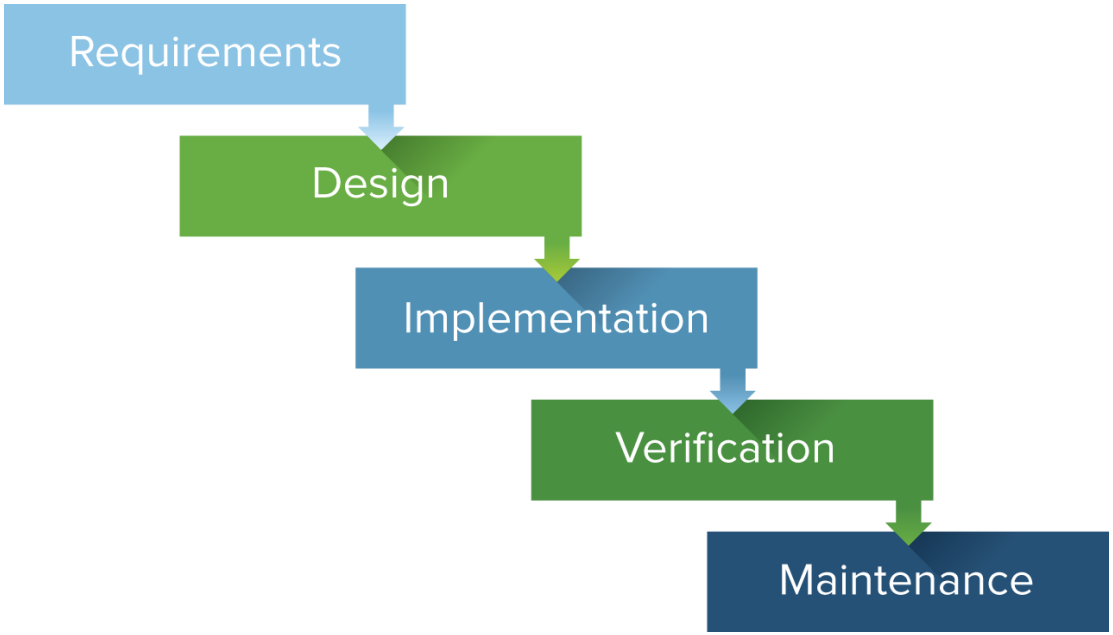
- **Two basic types:**
  - Linear
  - Iterative



- **Existing models:**
  - Waterfall (Linear, non-iterative)
  - Incremental (non-iterative)
  - V-Model (for verification and validation)
  - Prototyping (iterative)
  - Spiral (iterative)
- **Key quality considerations:**
  - User involvement (adapt to changes)
  - Development efficiency, project management complexity
  - Quality of software

Waterfall (sequential, non-iterative)

- **Progress is seen as flowing steadily downwards (like a waterfall) through the phases of conception, initiation, analysis, design, construction, testing, implementation and maintenance.**
- **Easy to use, but after-the-fact changes are prohibitively costly.**
- **Defined by Winston W. Royce in 1970.**

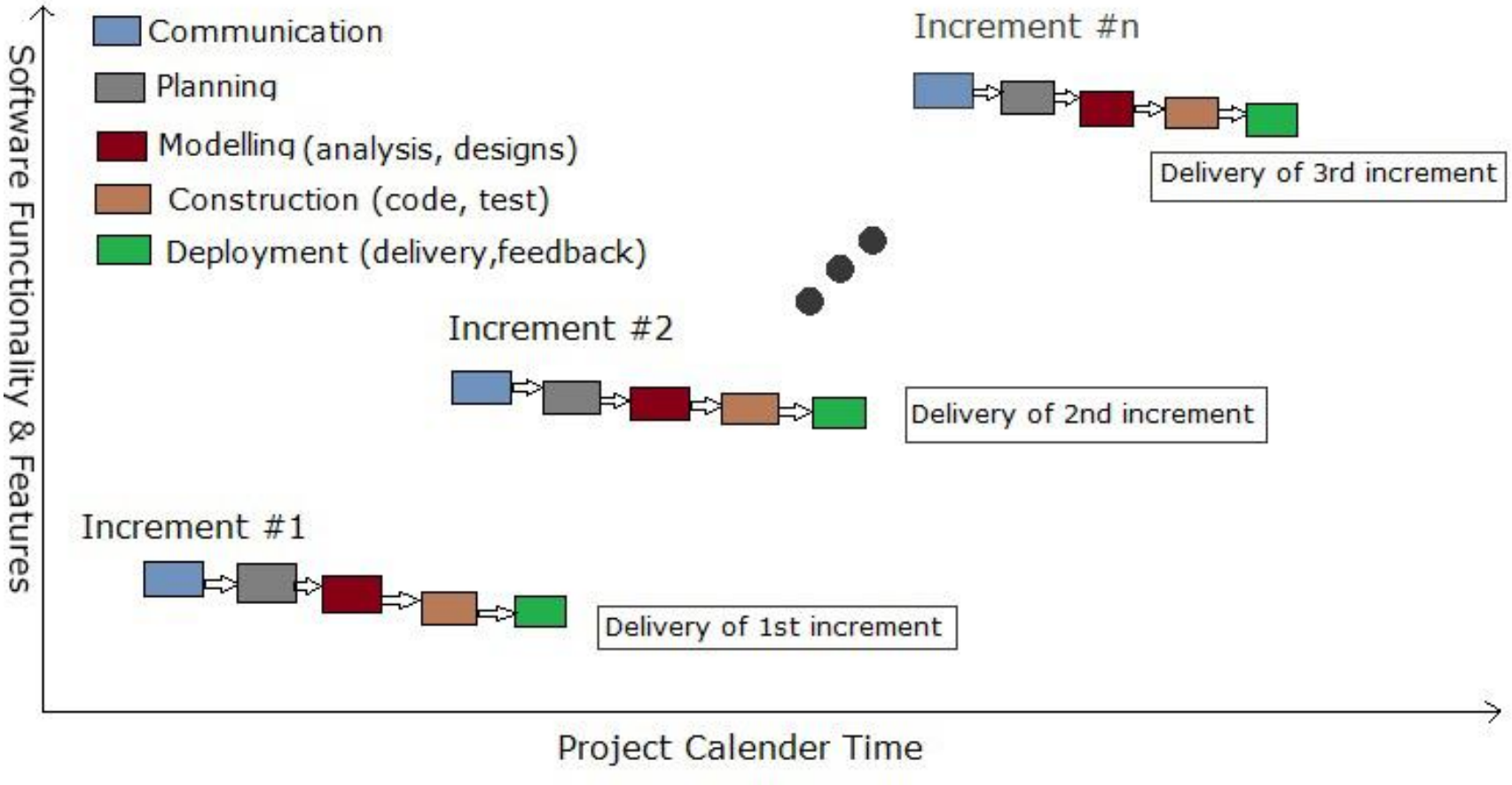


Incremental (non-iterative)

반복하는

- **The product is designed, implemented and tested incrementally (a little more is added each time) until the product is finished.**
- **It applies the waterfall model incrementally.**
  - System is broken down into many mini development projects.
  - Partial systems are built to produce the final system.
  - First tackled highest priority requirements.
  - The requirement of a portion is frozen once the incremented portion is developed.

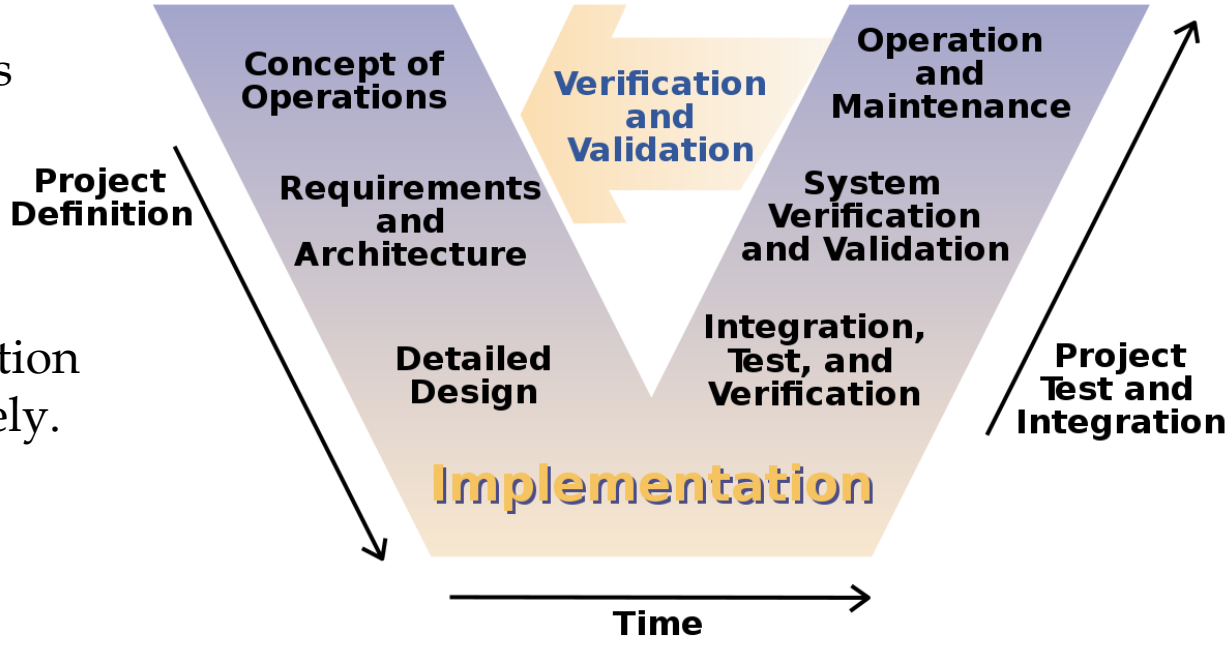
Incremental (non-iterative)



V-Model (for verification and validation)

- **V-model represents a development process that may be considered an extension of the waterfall model.**
  - Instead of moving down in a linear way, the process steps are bent upwards after the coding phase, to form the typical V shape.
  - Demonstrates the relationships between each phase of the development life cycle and its associated phase of testing.
  - The horizontal and vertical axes represents time or project completeness (left-to-right) and level of abstraction (coarsest-grain abstraction uppermost), respectively.

x+y=z

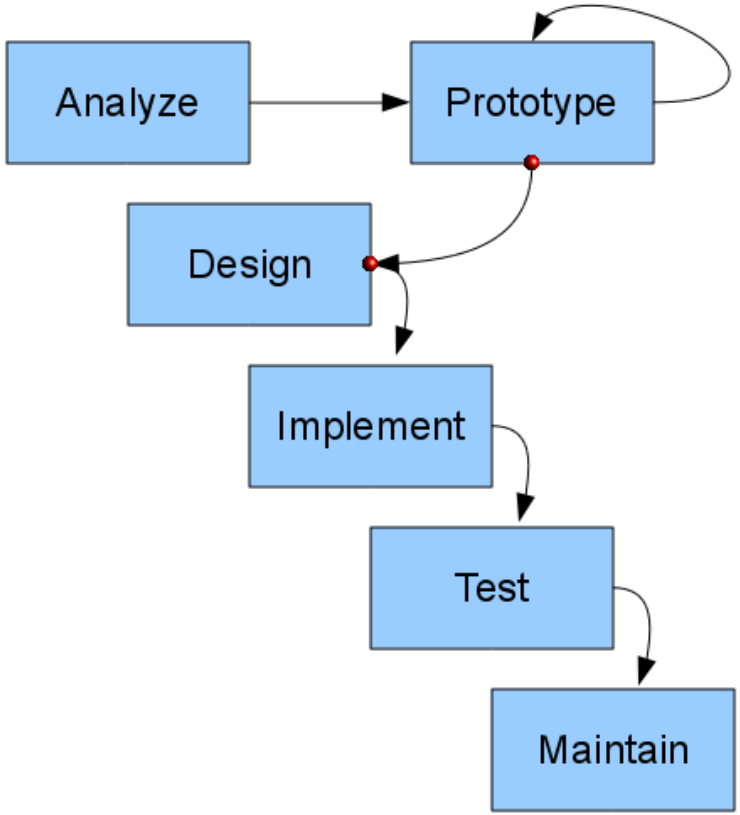


Prototyping (iterative)

- **Software prototyping is the activity of creating prototypes of software applications, i.e., incomplete versions of the software program being developed.**
  - A prototype typically simulates a few aspects of, and may be completely different from, the final product.
- **Process:**
  - **Identify basic requirements**: determine basic requirements including the input and output information desired. Details can typically be ignored.
  - **Develop initial prototype**: The initial prototype is developed that includes only user interfaces.
  - **Review**: The customers, including end-users, examine the prototype and provide feedback on additions or changes.
  - **Revise and enhance the prototype**: Using the feedback both the specifications and the prototype can be improved. If changes are introduced then a repeat of steps #3 and #4 may be needed.

Prototyping (iterative)

- **Benefits:**
  - The software designer and implementer can get valuable feedback from the users early in the project.
  - The client can compare if the software made matches the software specification, according to which the software program is built.
  - It also allows the software engineer some insight into the accuracy of initial project estimates and whether the deadlines and milestones proposed can be successfully met.

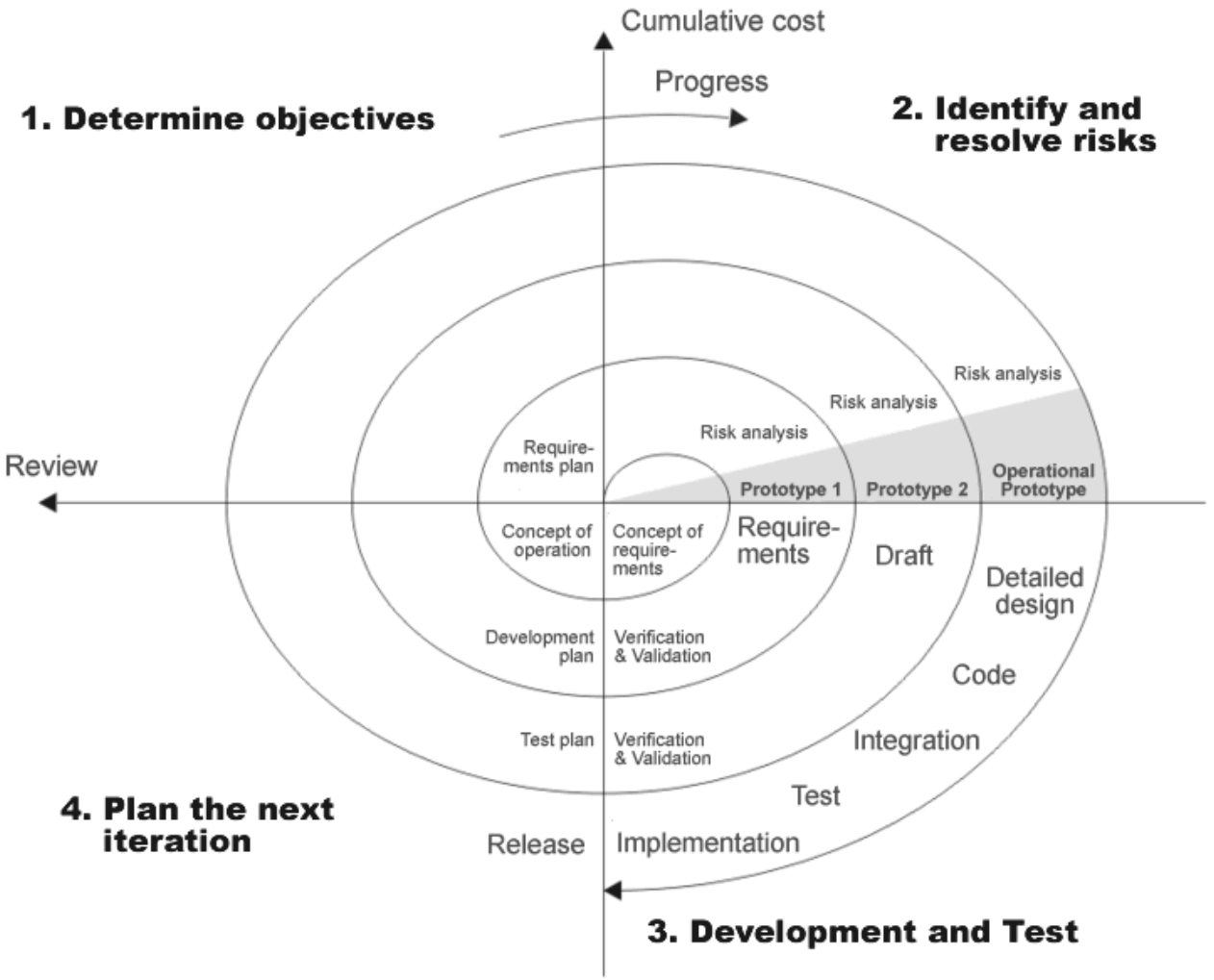


나선형

Spiral (iterative)

- **The spiral model is a risk-driven process model generator for software projects.**
  - Based on the unique risk patterns of a given project, the spiral model guides a team to adopt elements of one or more process models, such as incremental, waterfall, or evolutionary prototyping.
- **Firstly described by Barry Boehm in 1986.**

Spiral (iterative)



3 Agile development

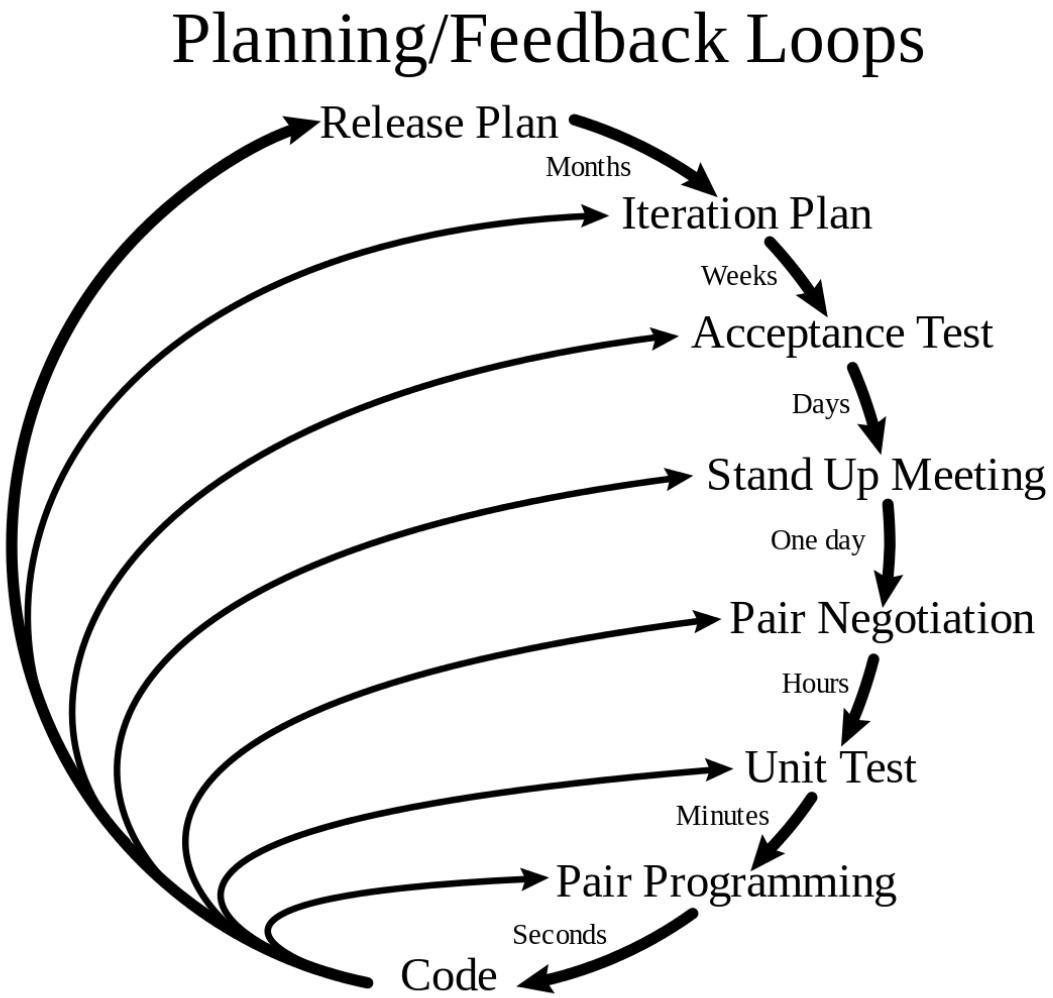
날렵한

Agile development

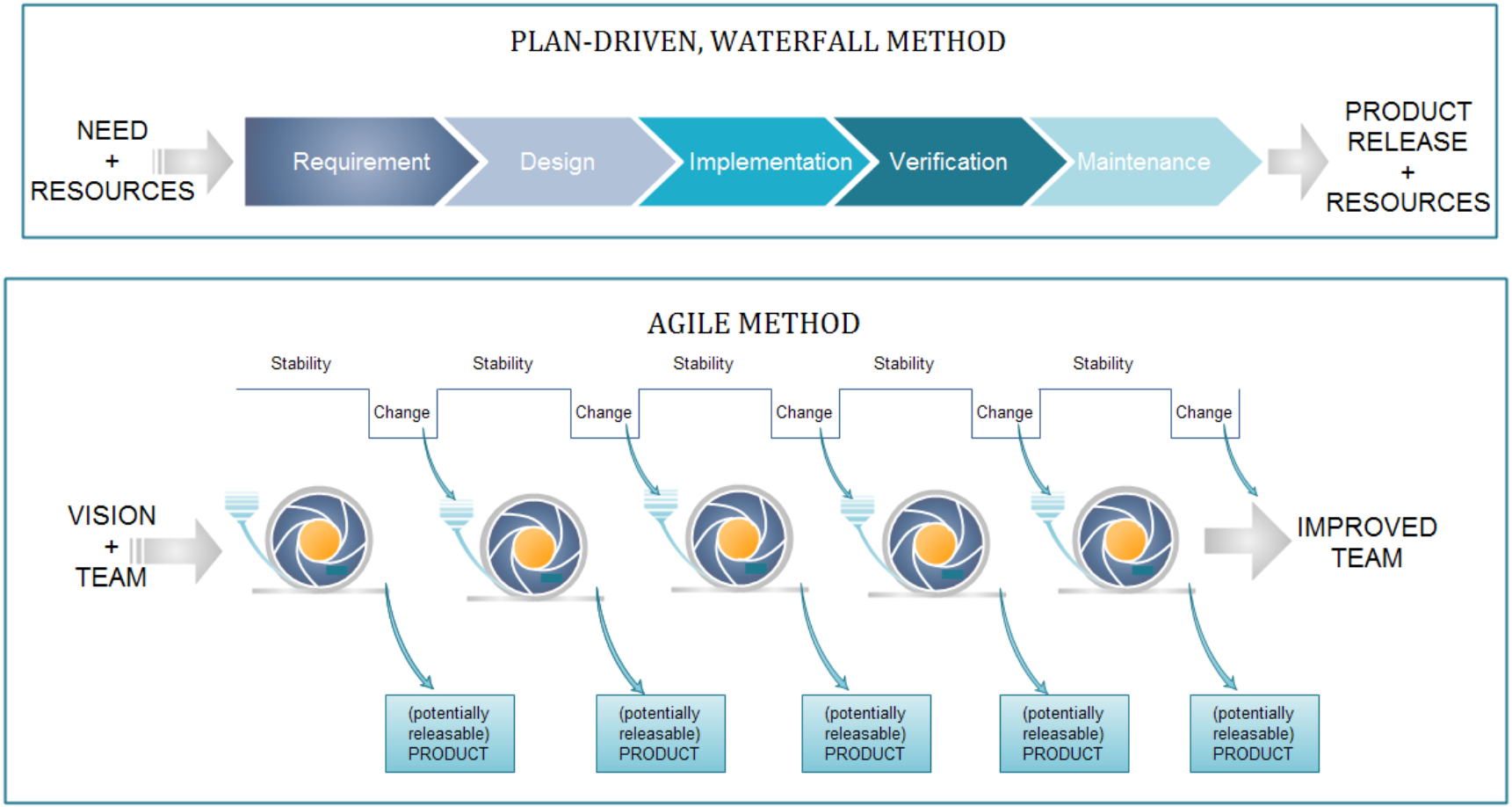
- **It advocates adaptive planning, evolutionary development, early delivery, and continuous improvement, and it encourages rapid and flexible response to change. Agile Manifesto was firstly coined in 2001 by 17 famous "programmers".**
  - Individuals and Interactions over processes and tools
    - Self-organization and motivation are important, as are interactions like co-location and pair programming.
  - Working Software over comprehensive documentation
    - Working software is more useful and welcome than just presenting documents to clients in meetings.
  - Customer Collaboration over contract negotiation
    - Requirements cannot be fully collected at the beginning of the software development cycle, therefore continuous customer or stakeholder involvement is very important.
  - Responding to Change over following a plan
    - Agile methods are focused on quick responses to change and continuous development.

Agile development

- **Extreme user involvement**
- **Extreme small iteration**
- **Extreme V&V**



Waterfall vs. Agile

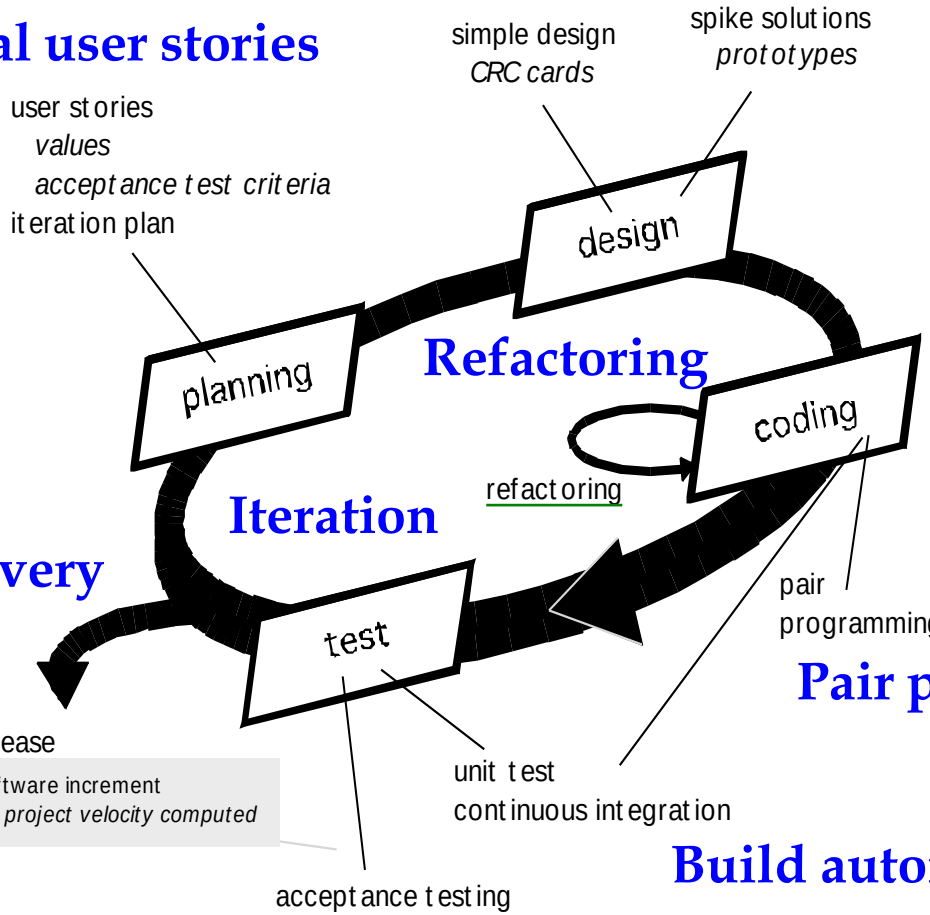


eXtreme Programming (XP)

refactoring : 소프트웨어를 보다 쉽게 이해할수 있고 적은비용으로
수정할수 있도록 겉으로 보이는 동작의 변화없이 내부 구조를 변경하는것

Prototyping

Incremental user stories



development

Test-Driven Dev. (TDD)

Continuous delivery

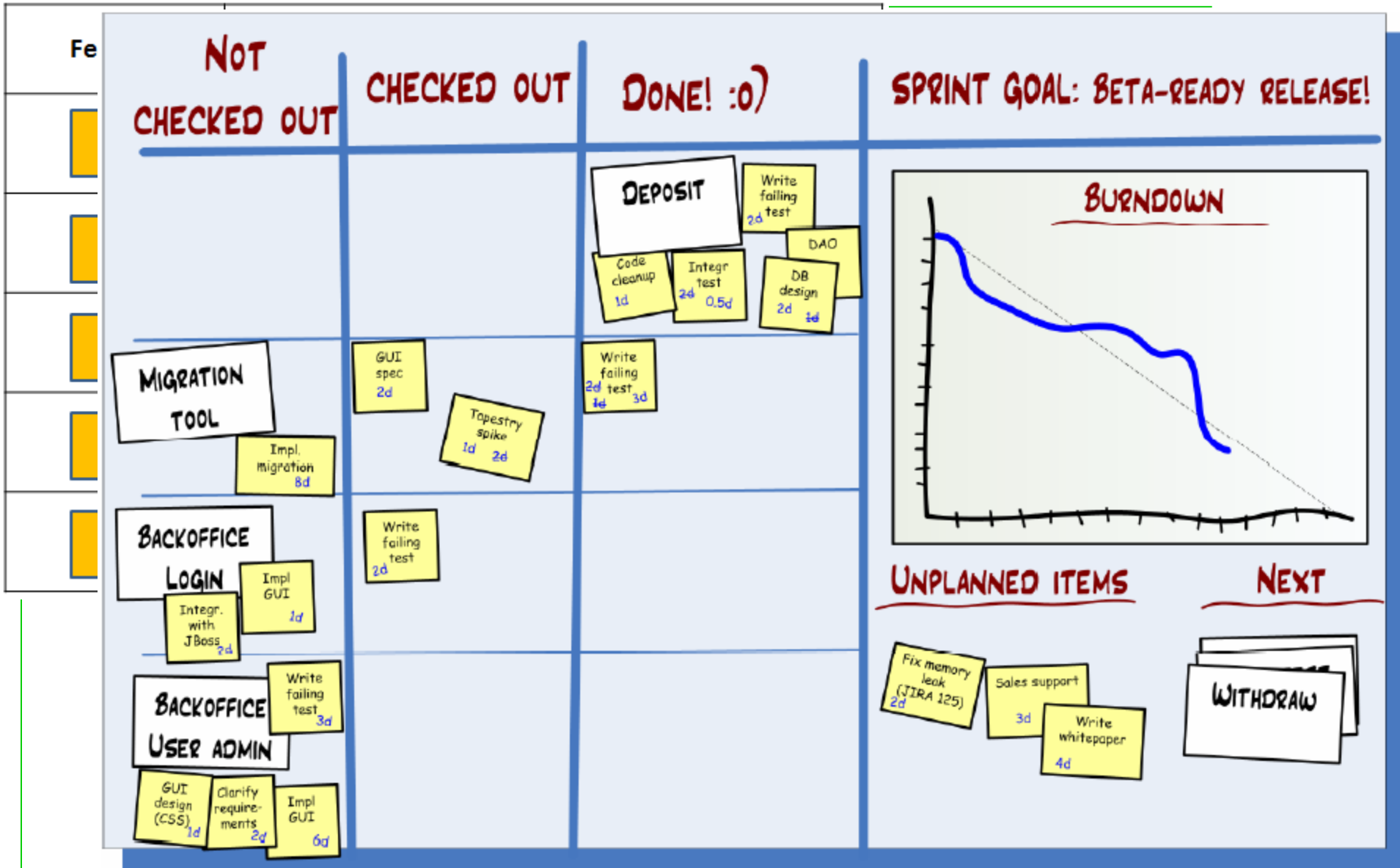
Pair programming

Build automation

Continuous integration

Pair Programming

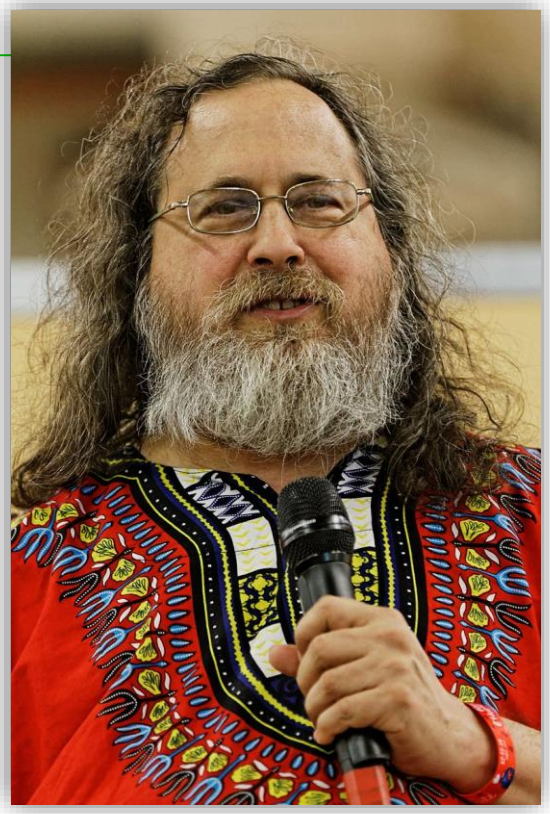
Task board and progress monitoring



4 Collaborative software development

Open Source

open source initiative



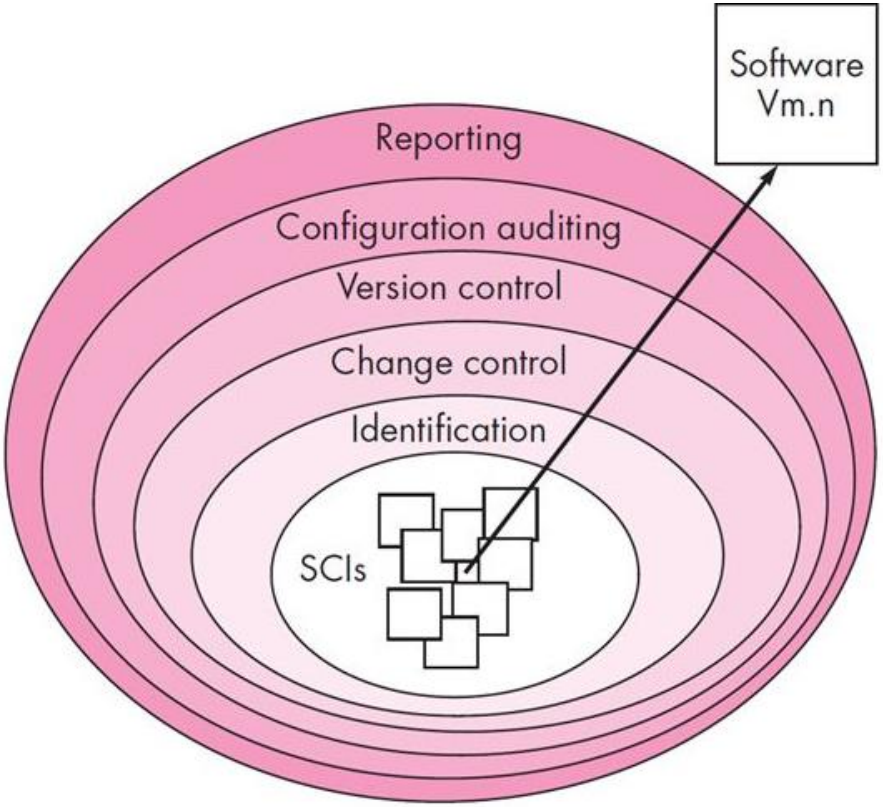
Richard Matthew Stallman (1953-)
Known for Free software movement, GNU, Emacs, GCC, GPL, FSF

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

5 Software Configuration Management (SCM) and Version Control System (VCS)

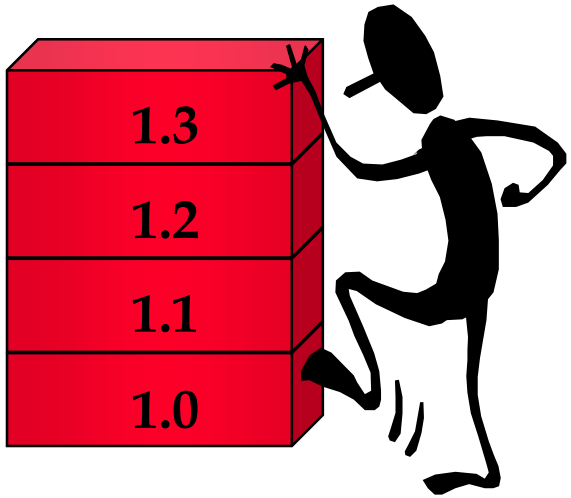
Software Configuration Mgmt. (SCM)

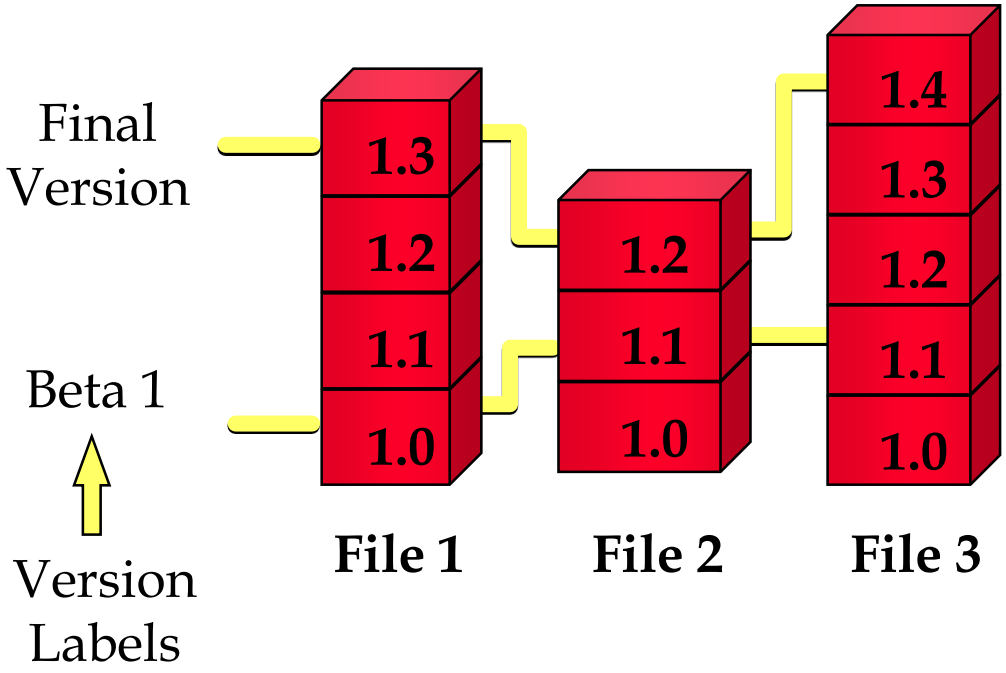
- **SCM is the task of tracking and controlling changes in the software.**
- **SCM practices include revision control and the establishment of baselines.**



Life Cycle of a Configuration Item (CI)

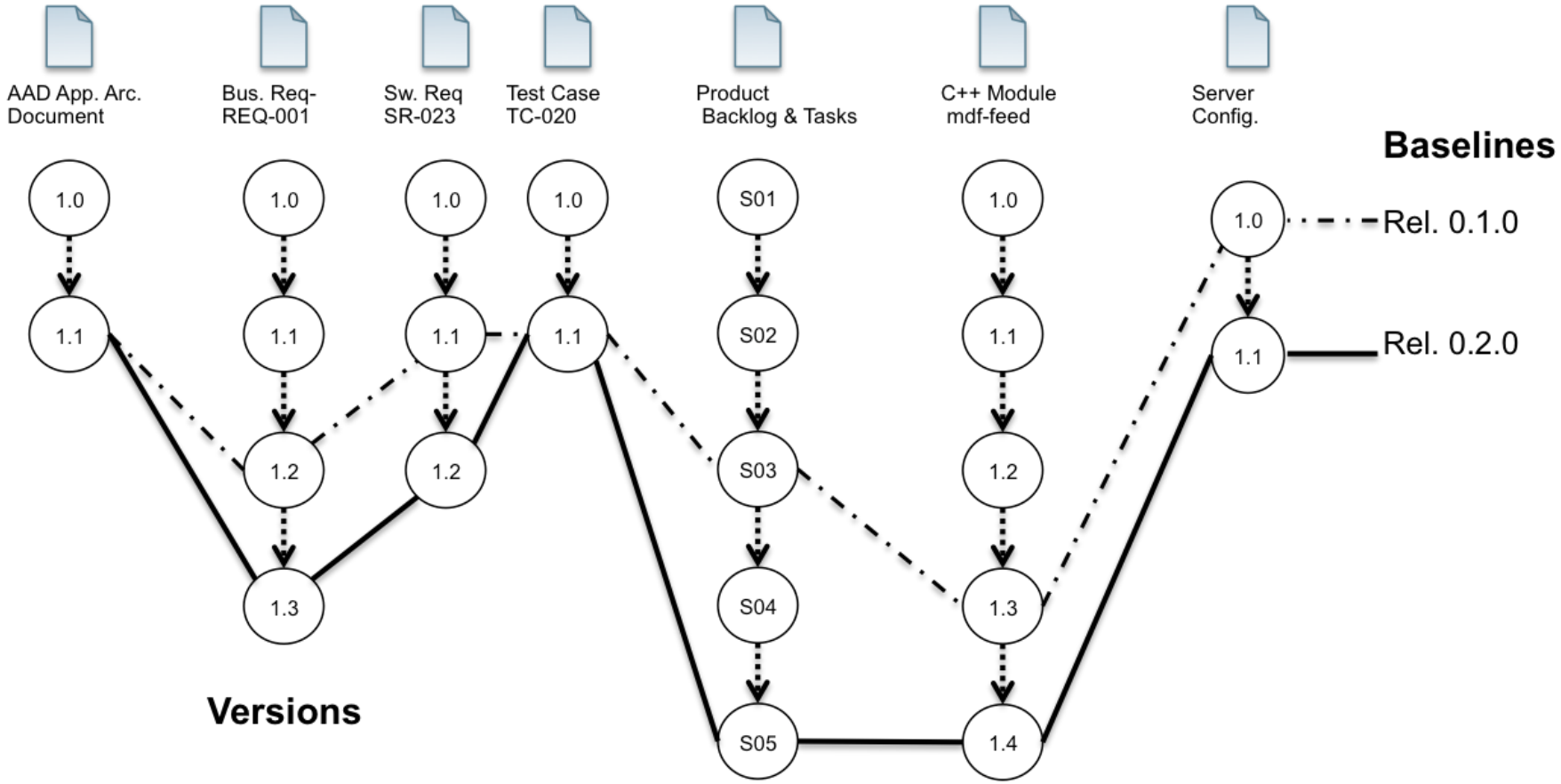
- **Any constituents of a software (source code, data, documents, hardware, various environments) may be updated along with the time in the life cycle of the software.**
- **Software Configuration Item (SCI): the fundamental structural unit of SCM.**





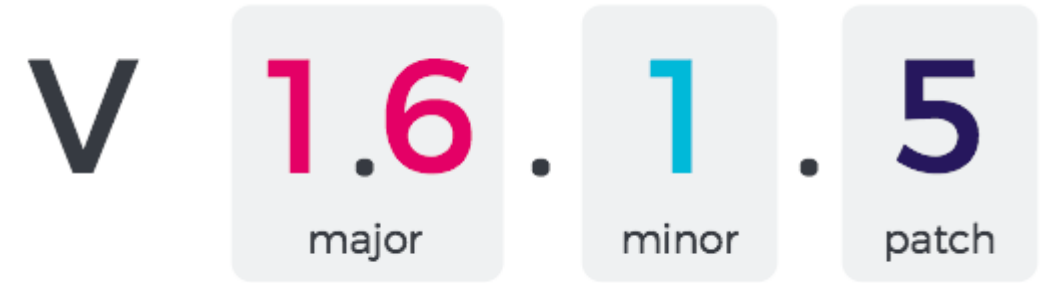
Configuration Items (SCI) and Baselines

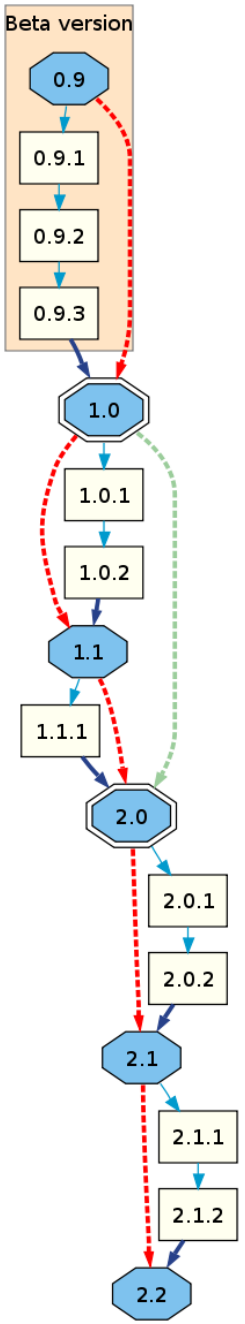
- **A baseline is an agreed description of the attributes of a product, at a point in time, which serves as a basis for defining change.**



Versioning

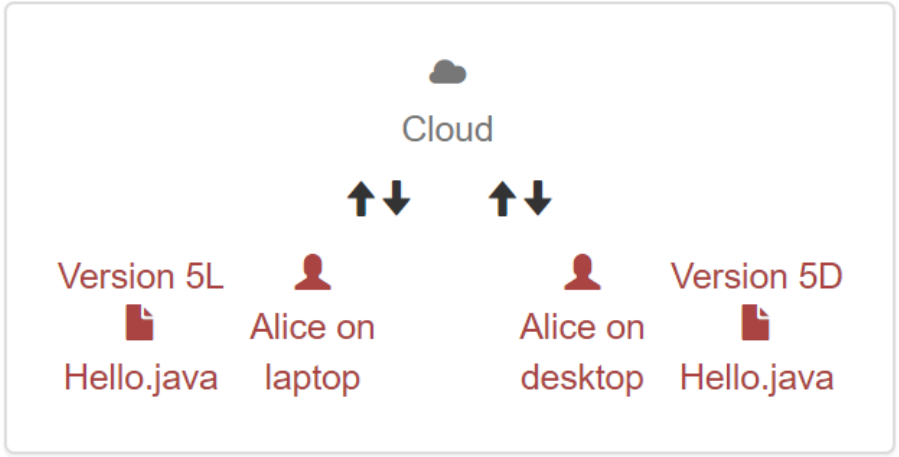
- **Software versioning is the process of assigning either unique version names or unique version numbers to unique states of computer software.**
  - Within a given version number category (major, minor), these numbers are generally assigned in increasing order and correspond to new developments in the software.
  - At a fine-grained level, revision control is often used for keeping track of incrementally different versions of electronic information, whether or not this information is computer software.





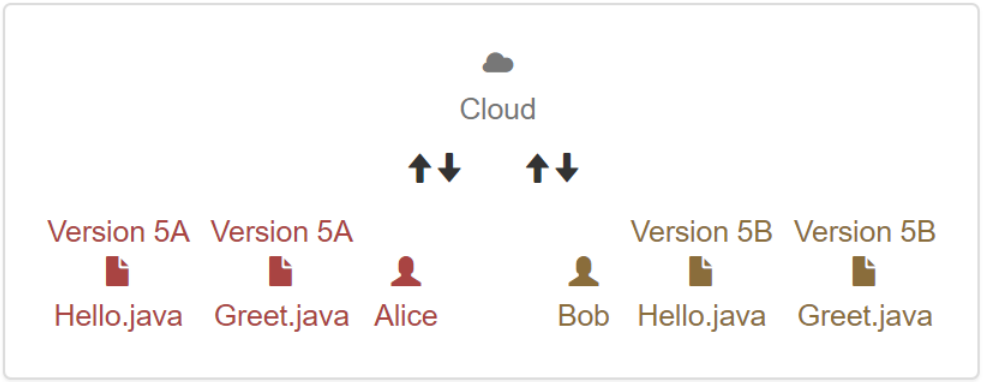
Why version control is required – for individuals

- *Reverting* to a past version
- *Comparing* two different versions
- *Pushing* full version history to another location
- *Pulling* history back from that location
- *Merging* versions that are offshoots of the same earlier version

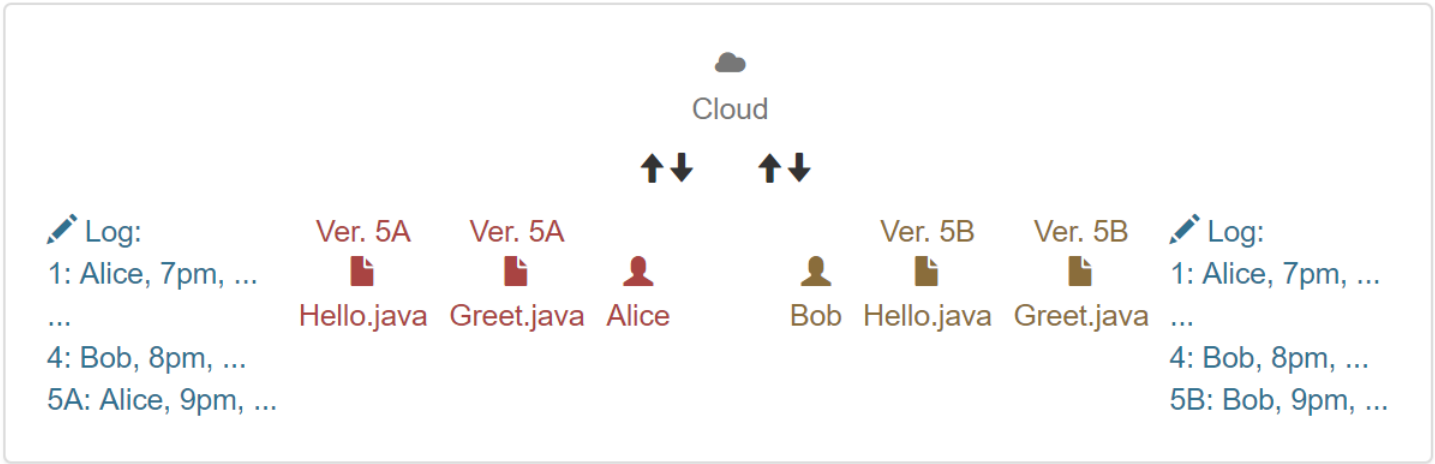


Why version control is required – for teamwork

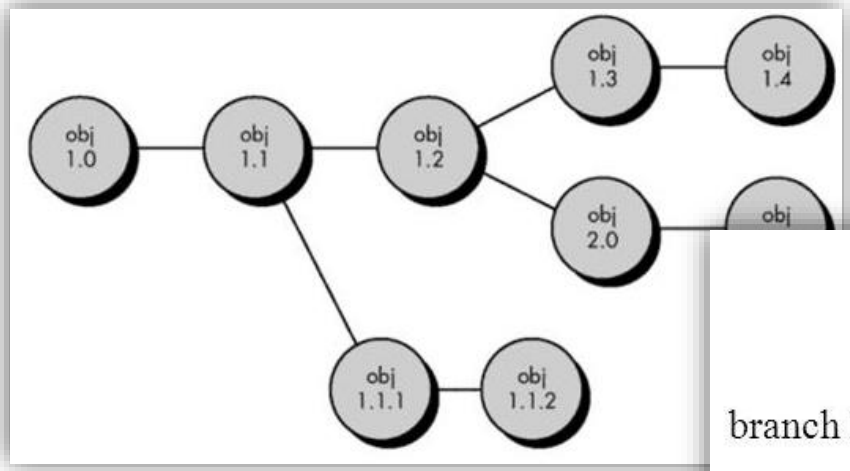
- **Communications and share/merge works among multiple developers**

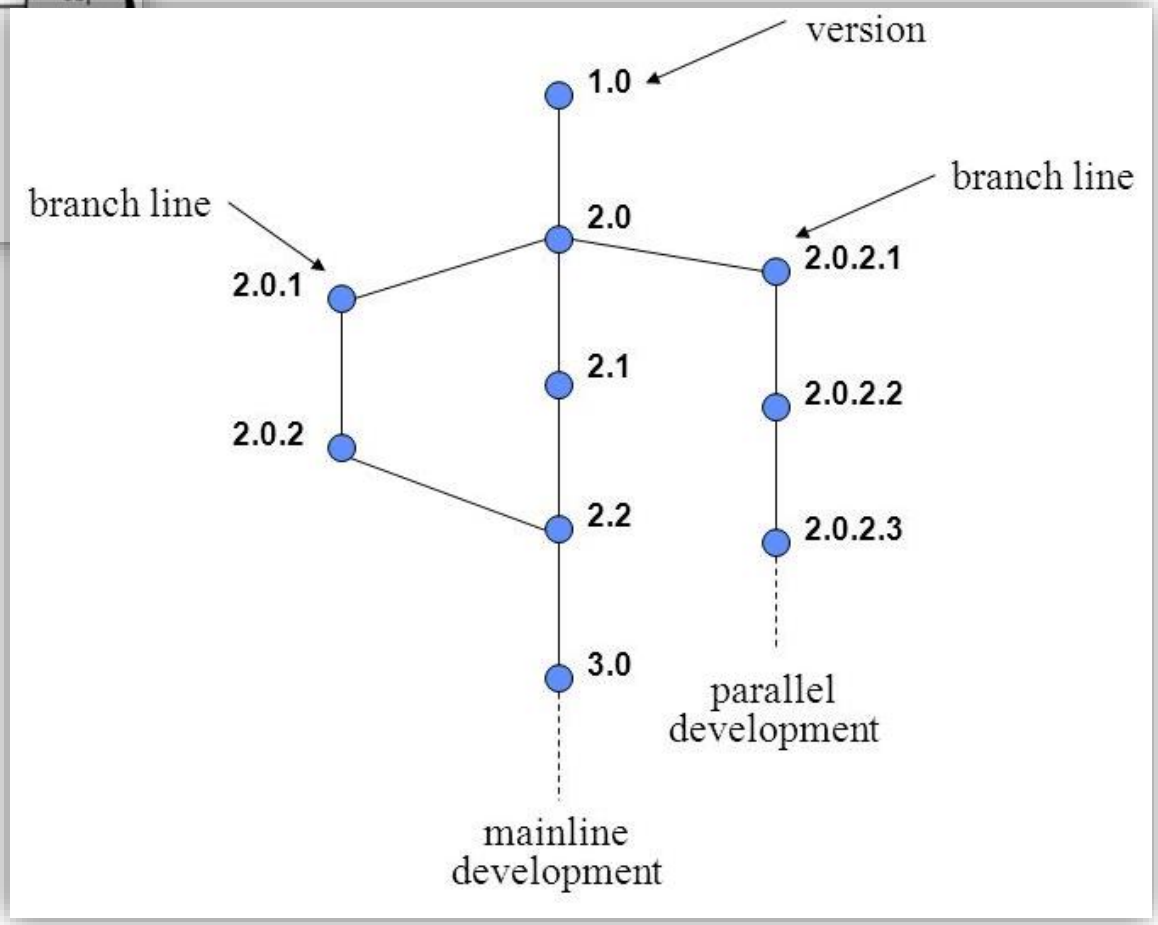


- **Logging individualized works of different developers for auditing**



Branches and Evolution Graph (of a SCI or a system)



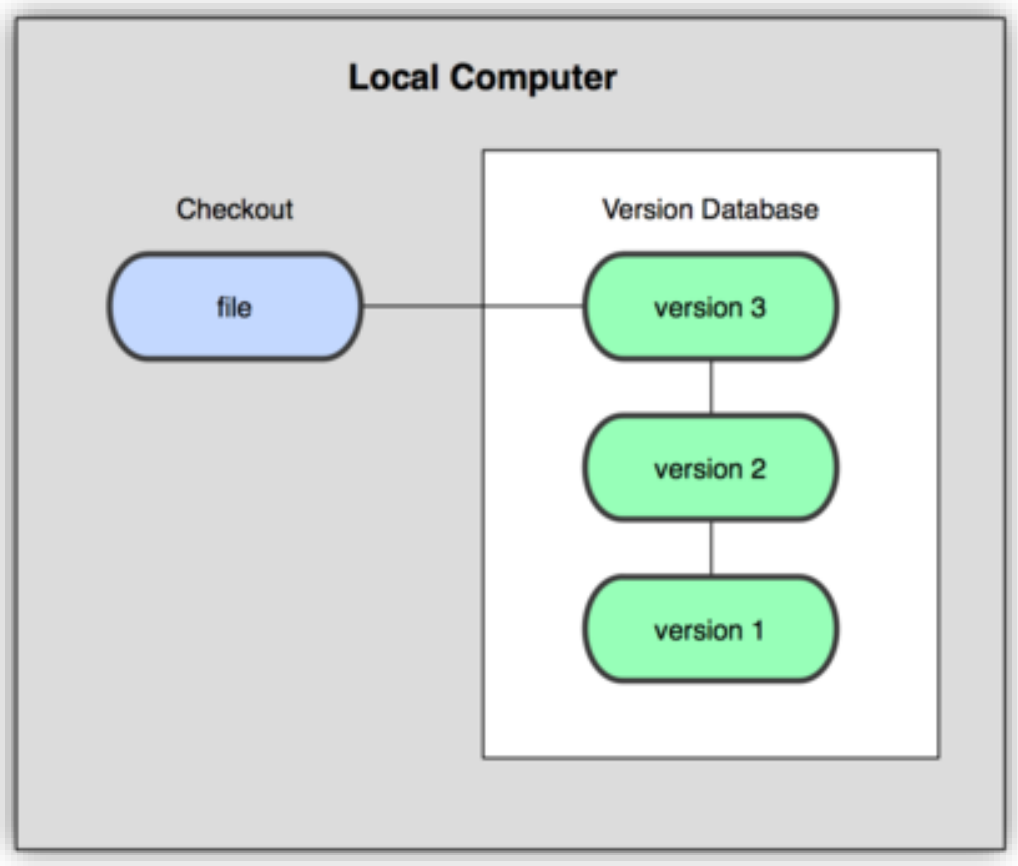


Branches

- It sometimes makes sense for a subset of the developers to go off and work on a *branch*, a parallel code universe for, say, experimenting with a new feature.
- The other developers don't want to pull in the new feature until it is done, even if several coordinated versions are created in the meantime.
- Even a single developer can find it useful to create a branch, for the same reasons that Alice was originally using the cloud server despite working alone.
- In general, it will be useful to have many shared places for exchanging project state. There may be multiple branch locations at once, each shared by several programmers. With the right set-up, any programmer can pull from or push to any location, creating serious flexibility in cooperation patterns.

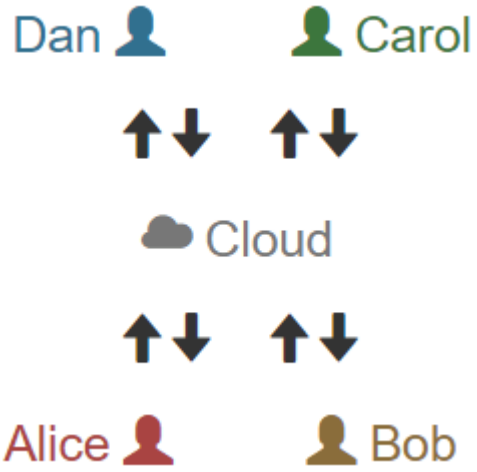
Version Control System (VCS)

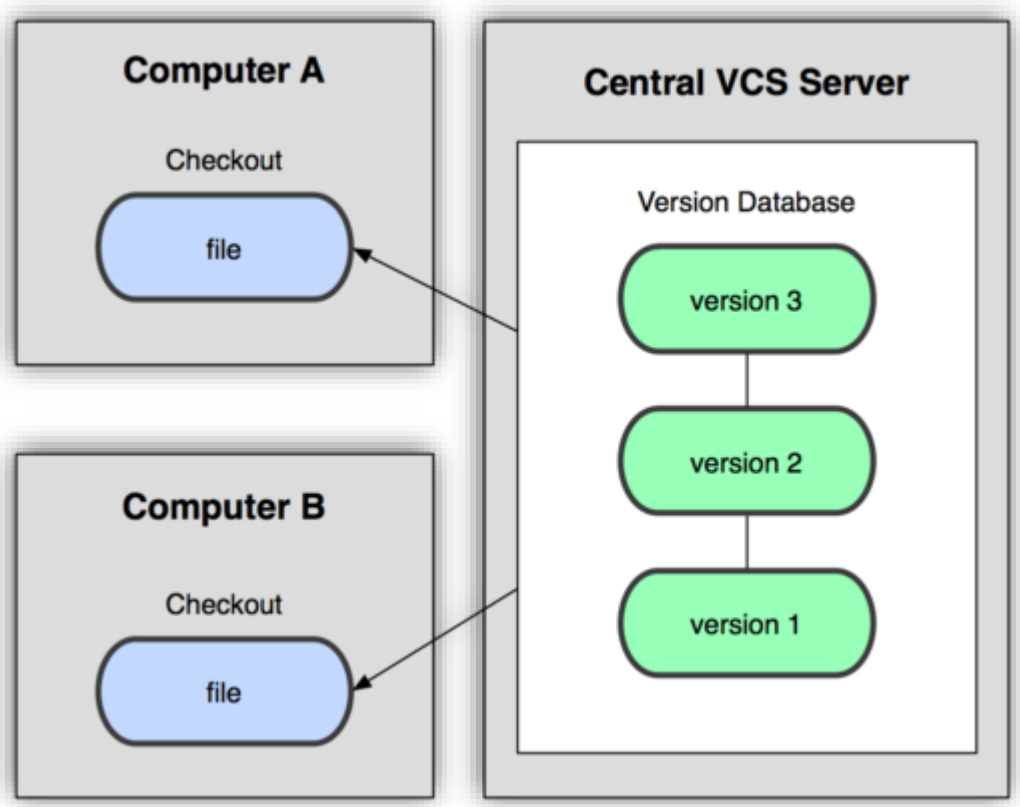
- **Local VCS**
- **Centralized VCS**
- **Distributed VCS**



Version Control System (VCS)

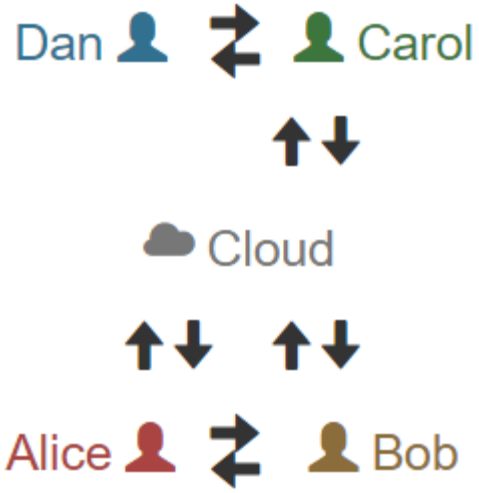
- **Local VCS**
- **Centralized VCS**
- **Distributed VCS**

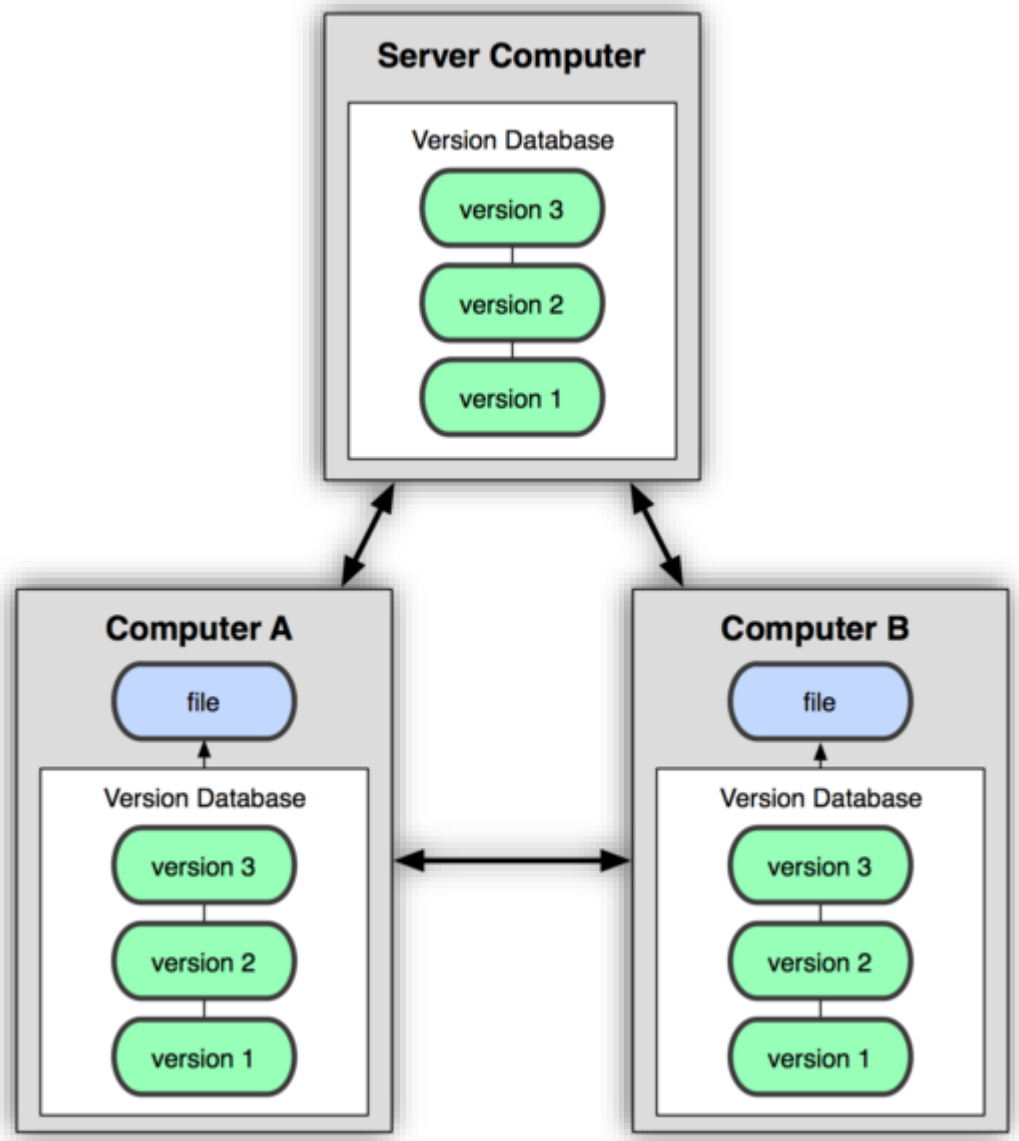




Version Control System (VCS)

- **Local VCS**
- **Centralized VCS**
- **Distributed VCS**





Version control terminology

- **Repository**: a local or remote store of the versions in our project
- **Working copy**: a local, editable copy of our project that we can work on
- **File**: a single file in our project
- **Version** or **revision**: a record of the contents of our project at a point in time
- **Change** or **diff**: the difference between two versions
- **Head**: the current version

Features of a version control system

- It should **allow multiple people to work together**:
  - **Merge**: combine versions that diverged from a common previous version
  - **Track responsibility**: who made that change, who touched that line of code?
  - **Work in parallel**: allow one programmer to work on their own for a while (without giving up version control)
  - **Work-in-progress**: allow multiple programmers to share unfinished work (without disrupting others, without giving up version control)

5 Git as an example of SCM tool

What is Git?

Linux

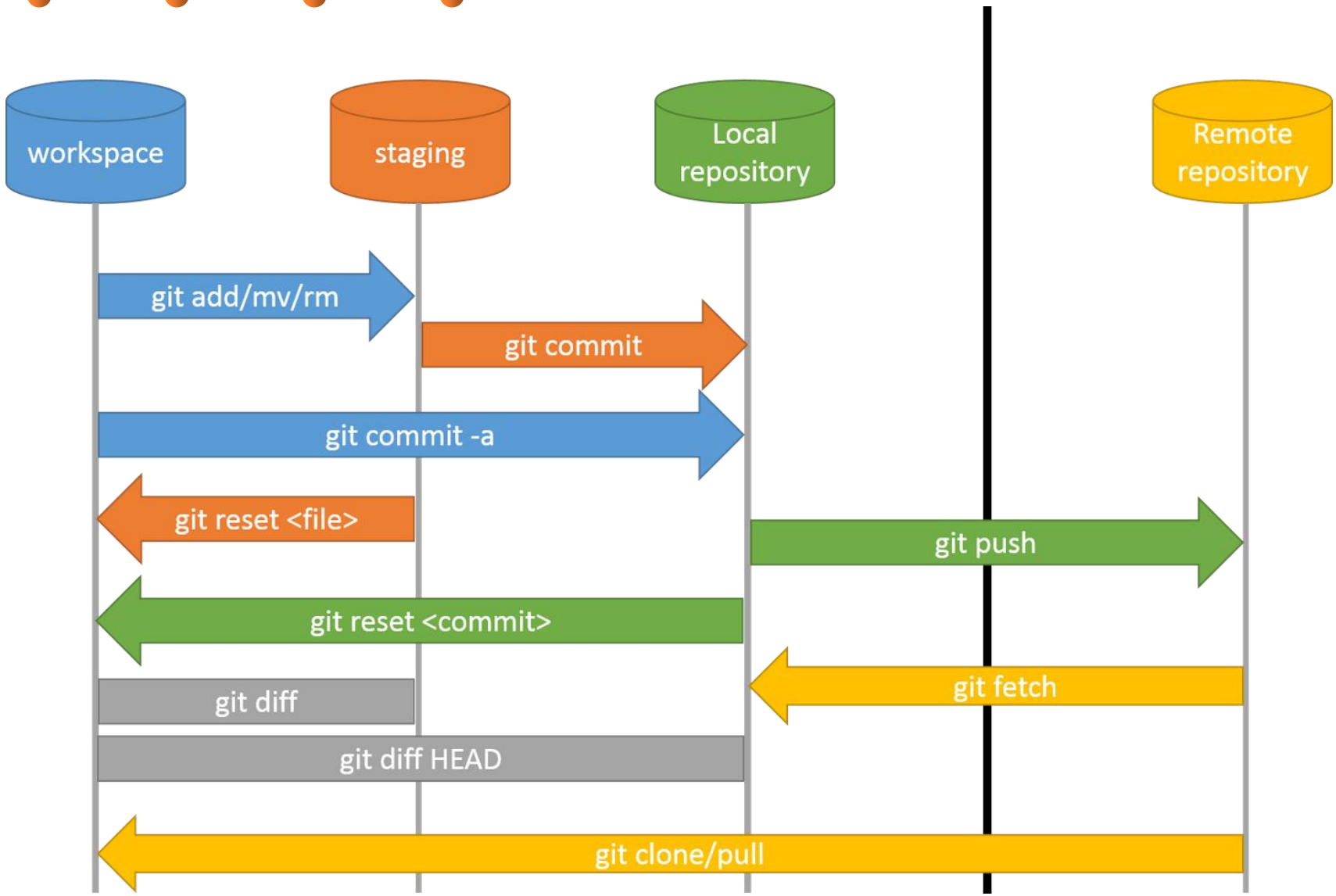
- **Initial release: 2005**
- **Initial Author: Linus Torvalds**
- **For development of the Linux kernel.**

Linus Torvalds (1969-)

Version	Original release date	Latest version	Release date
0.99	2005-07-11	0.99.9n	2005-12-15
1.0	2005-12-21	1.0.13	2006-01-27
1.1	2006-01-08	1.1.6	2006-01-30
1.2	2006-02-12	1.2.6	2006-04-08
1.3	2006-04-18	1.3.3	2006-05-16
1.4	2006-06-10	1.4.4.5	2008-07-16
1.5	2007-02-14	1.5.6.6	2008-12-17
1.6	2008-08-17	1.6.6.3	2010-12-15
1.7	2010-02-13	1.7.12.4	2012-10-17
1.8	2012-10-21	1.8.5.6	2014-12-17
1.9	2014-02-14	1.9.5	2014-12-17
2.0	2014-05-28	2.0.5	2014-12-17
2.1	2014-08-16	2.1.4	2014-12-17
2.2	2014-11-26	2.2.3	2015-09-04
2.3	2015-02-05	2.3.10	2015-09-29
2.4	2015-04-30	2.4.11	2016-03-17
2.5	2015-07-27	2.5.5	2016-03-17
2.6	2015-09-28	2.6.6	2016-03-17
2.7	2015-10-04	2.7.4	2016-03-17
2.8	2016-03-28	2.8.4	2016-06-06
2.9	2016-06-13	2.9.3	2016-08-12
2.10	2016-09-02	2.10.2	2016-10-28
2.11	2016-11-29	2.11.0	2016-11-29

Legend: Old version | Older version, still supported | **Latest version** | Latest preview version

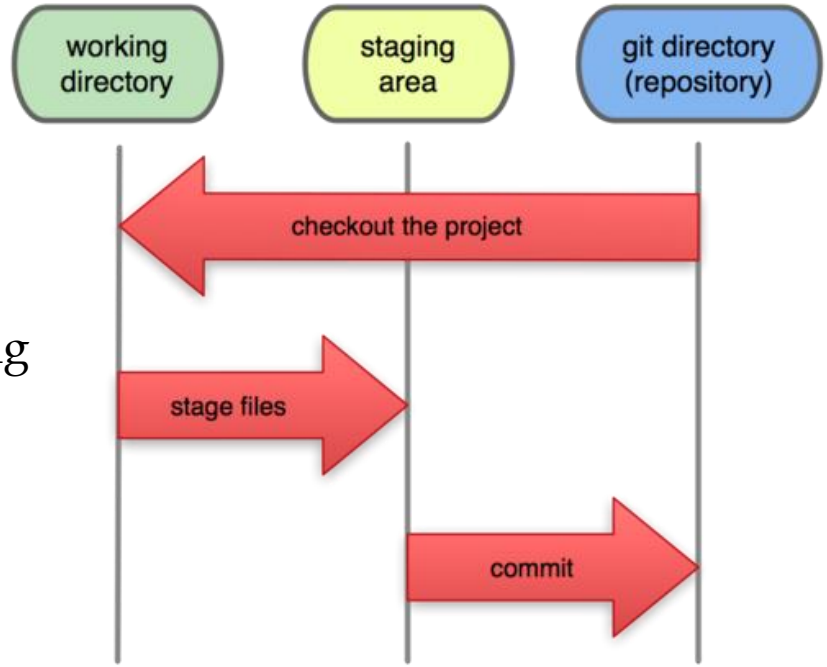
Managing changes in software evolution process



Git repository

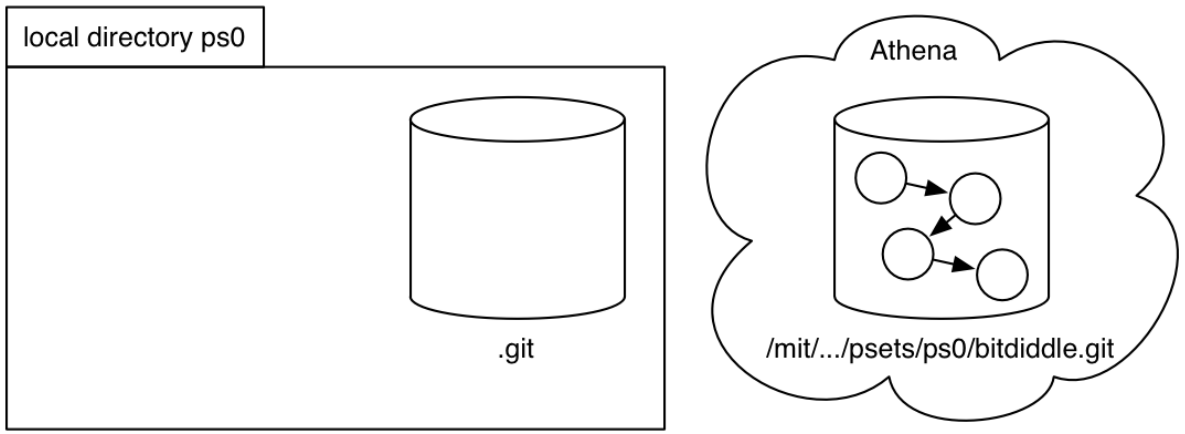
저장소

- **A Git repository has three parts:**
  - .git directory (a repository storing all version control data)
  - Working directory (local file system)
  - Staging area (in memory)
- **Each file belongs to one of the following three states:**
  - Modified (the file in working directory is different from the one in git repository, but is not in staging area)
  - Staged (the file is modified and has been added into the staging area)
  - Committed (the file keeps same in working directory and git directory)



Object graph in Git

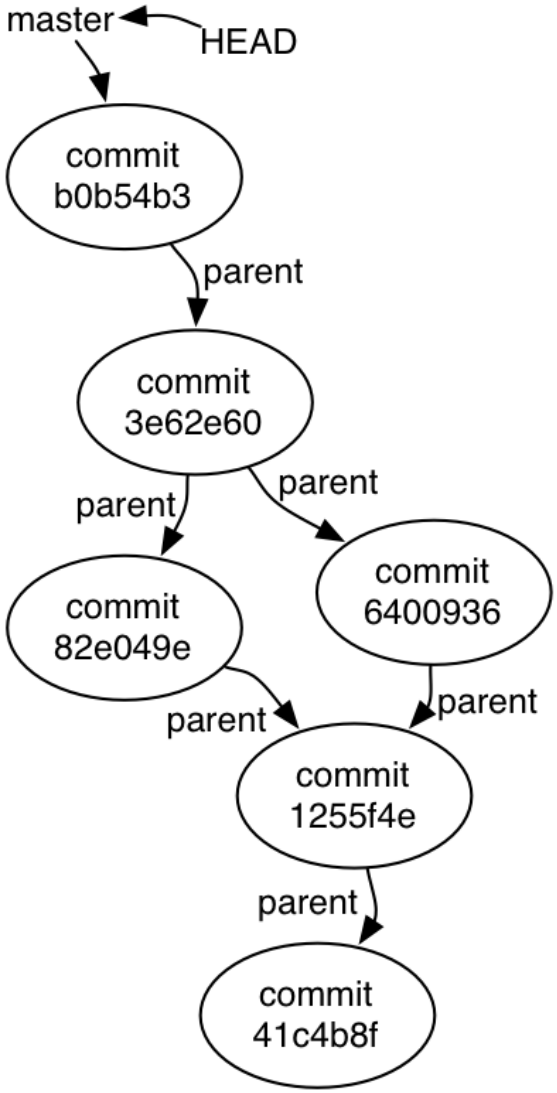
- **All of the operations we do with Git — clone, add, commit, push, log, merge, … — are operations on a graph data structure that stores all of the versions of files in the project, and all the log entries describing those changes.**
- **The Git object graph is stored in the .git directory of the repository.**
- **Copying a git project from another machine/server means copying the whole object graph.**
  - git clone



What an Object Graph looks like?

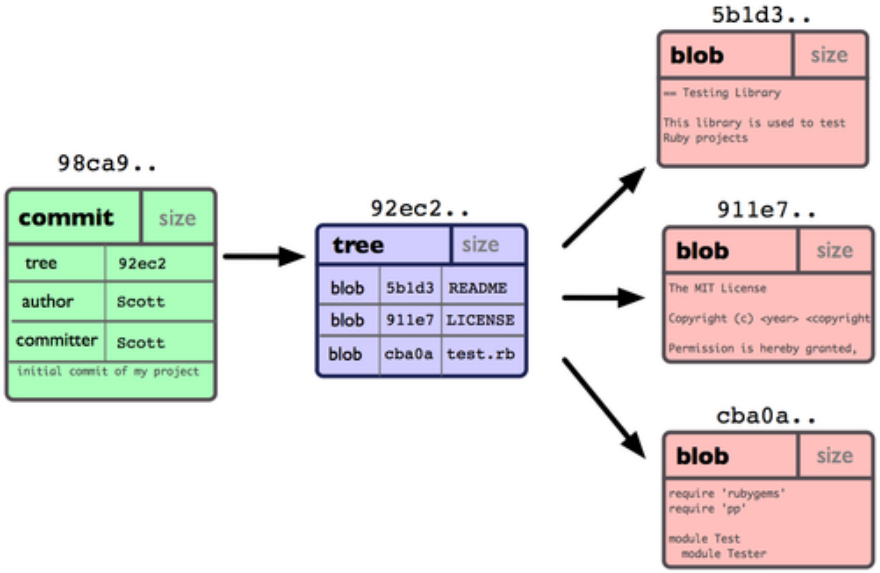
- **Object graph, being the history of a Git project, is a directed acyclic graph (DAG).**

```
* b0b54b3 (HEAD, origin/master, origin/HEAD, master) Greeting in Java
*   3e62e60 Merge
|\
| * 6400936 Greeting in Scheme
* | 82e049e Greeting in Ruby
|/
* 1255f4e Change the greeting
* 41c4b8f Initial commit
```



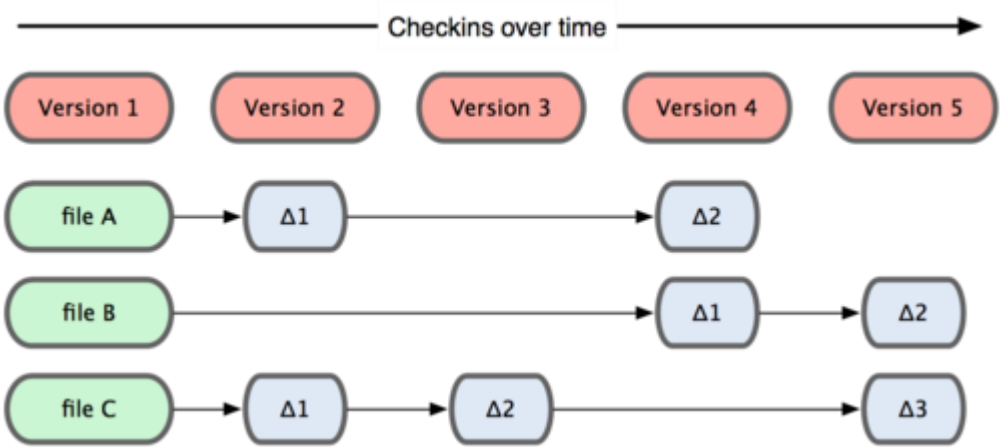
Commits: nodes in Object Graph

- Each commit is a snapshot of our entire project, which Git represents with a **tree** node. For a project of any reasonable size, most of the files *won't* change in any given revision. Storing redundant copies of the files would be wasteful, so Git doesn't do that.
- Instead, the Git object graph stores each version of an individual file *once*, and allows multiple commits to *share* that one copy.
- Each commit also has log data — who, when, short log message, etc.

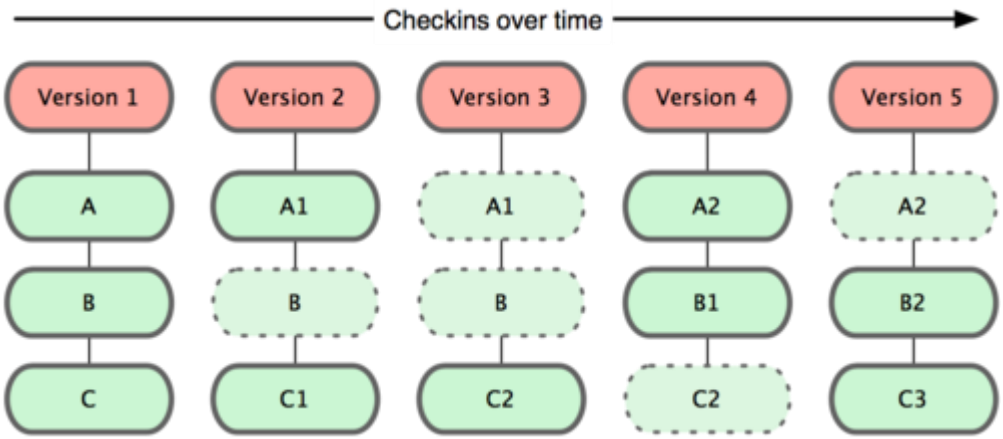


Managing changes in Git

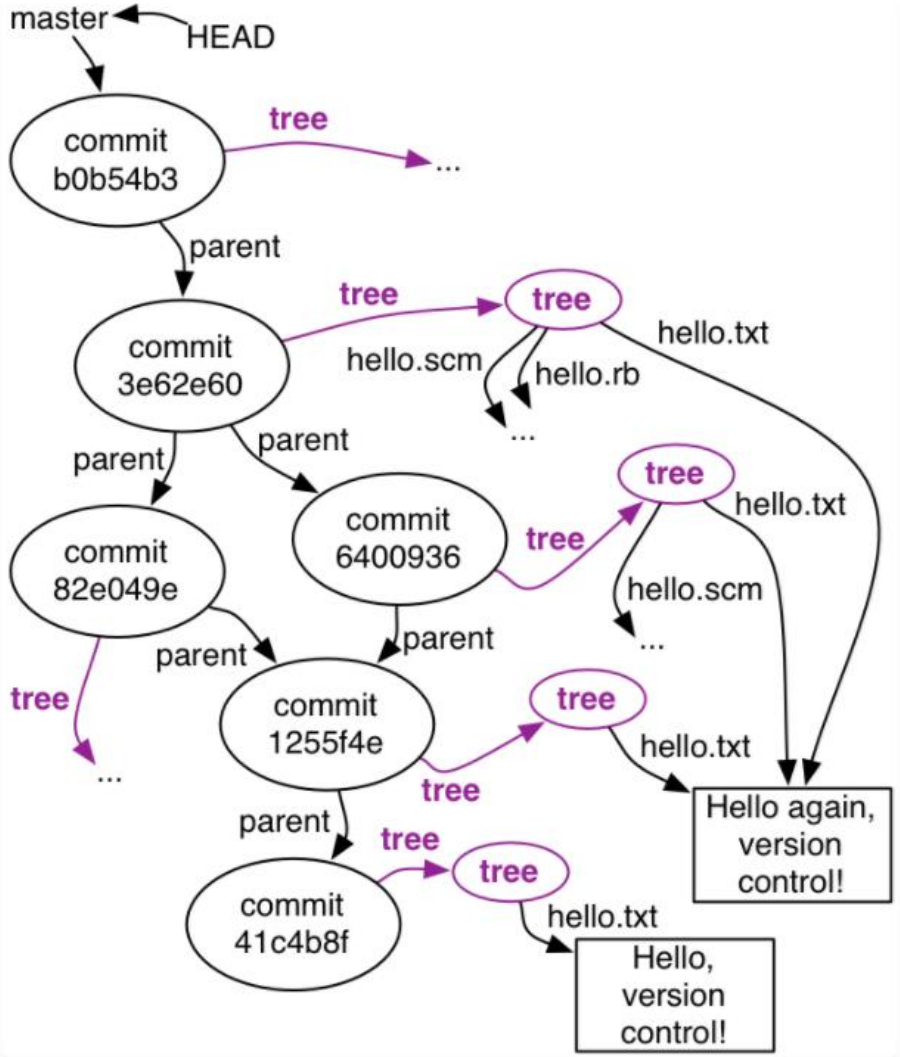
- **Traditional VCS:**



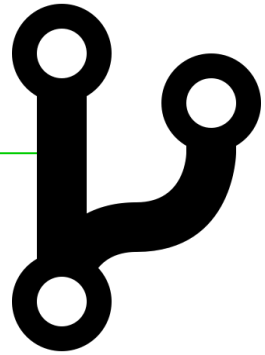
- **In Git:**



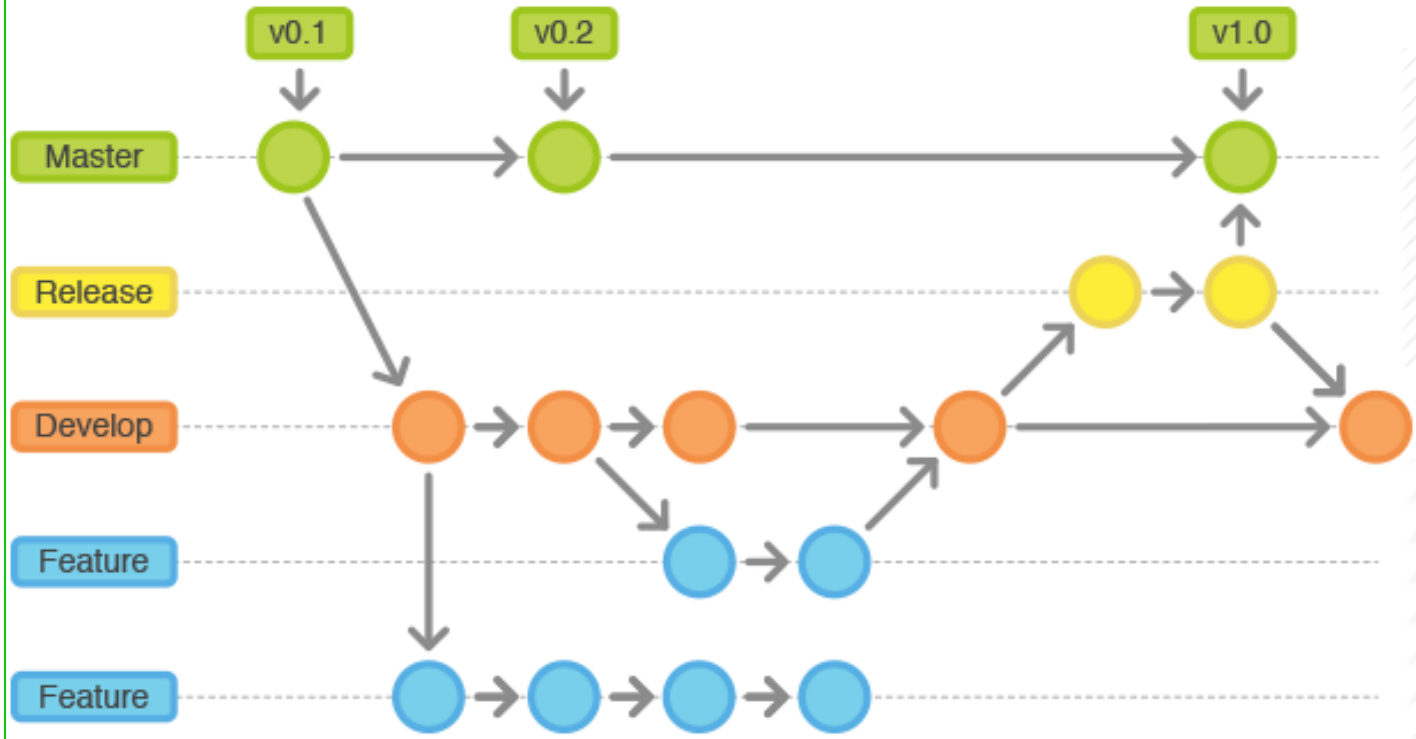
Commits: nodes in Object Graph

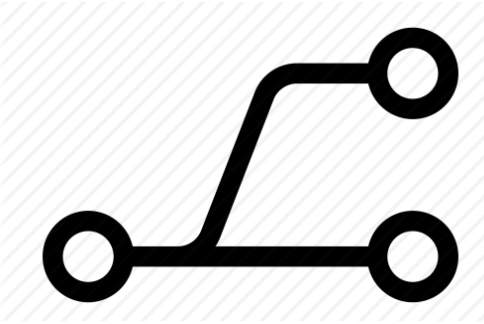


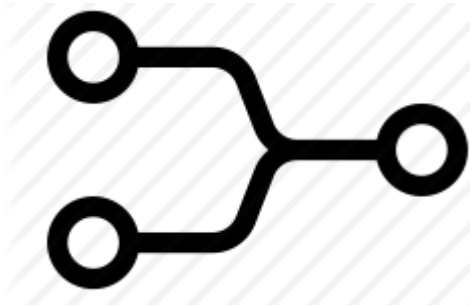
Git supports Branch and Merge



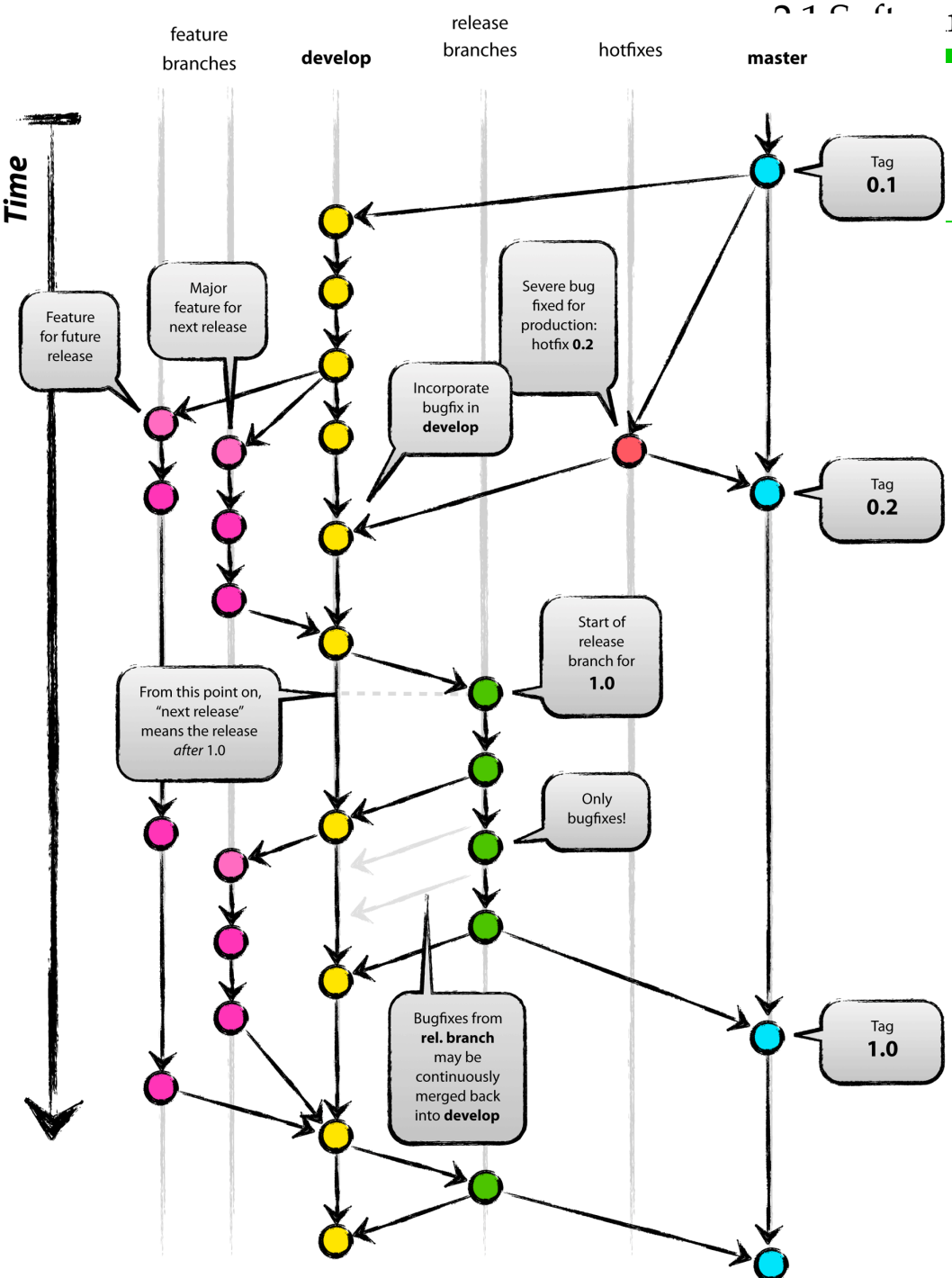
- **A branch is the duplication of an object under revision control so that modifications can happen in parallel along both branches.**
- **Merging two branches together.**





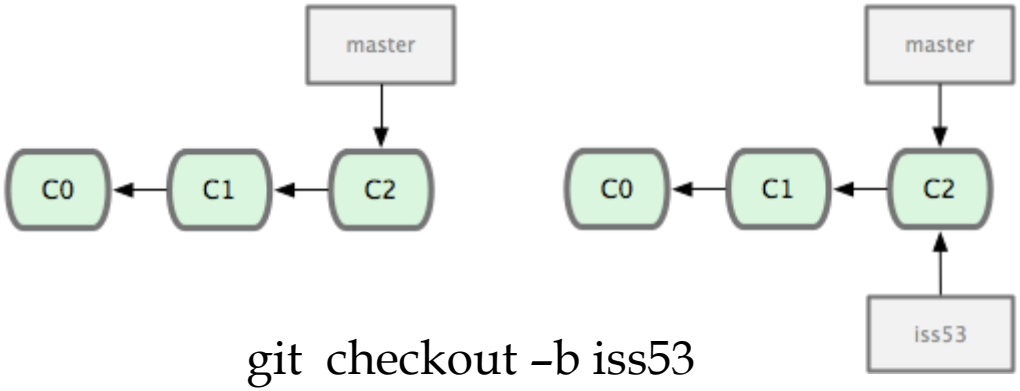


Branching

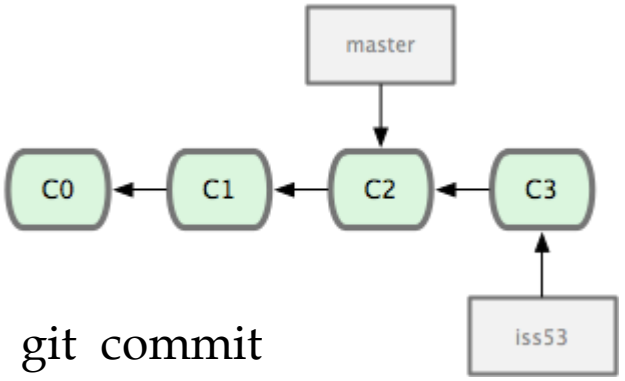

feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Major feature for next release
Feature for future release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after 1.0
Only bugfixes!
Bugfixes from rel. branch may be continuously merged back into develop
Tag 1.0

저장

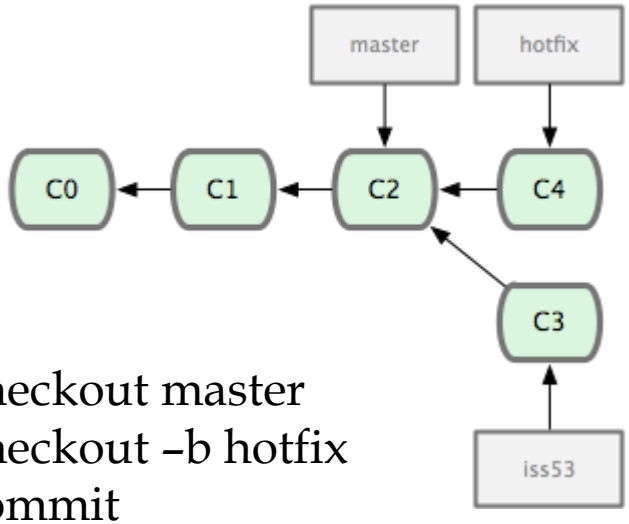
Creating and merging branches in Git



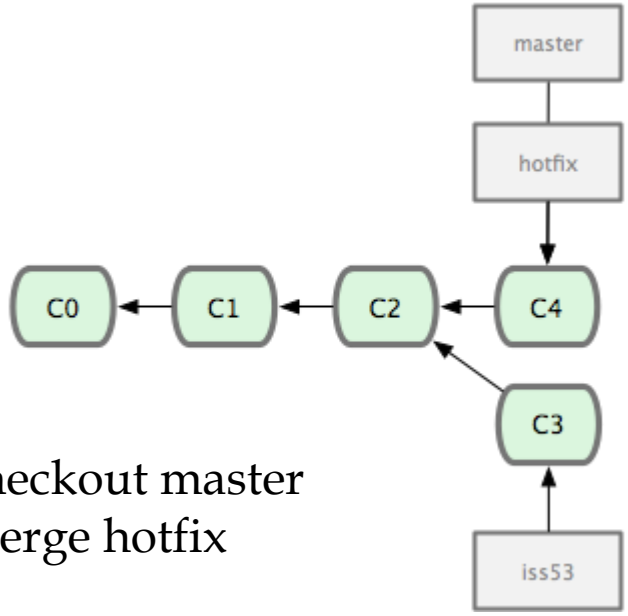
git checkout –b iss53



git commit

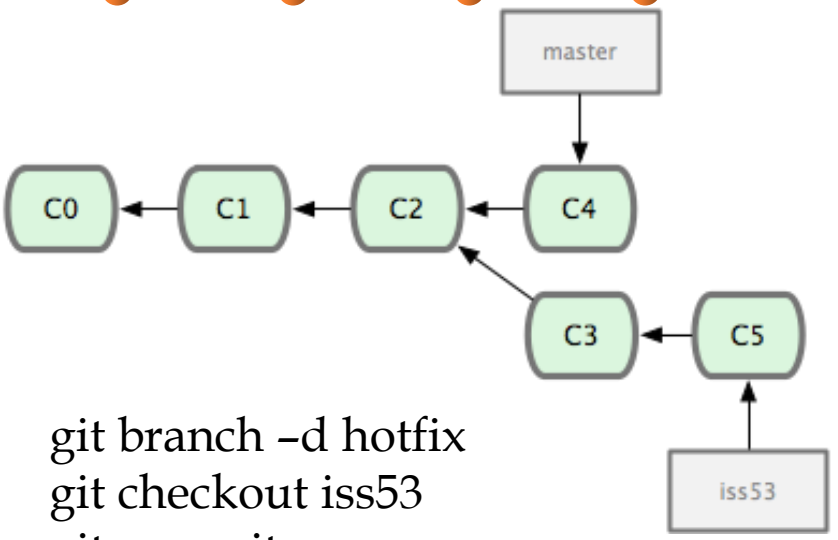


git checkout master
git checkout –b hotfix
git commit

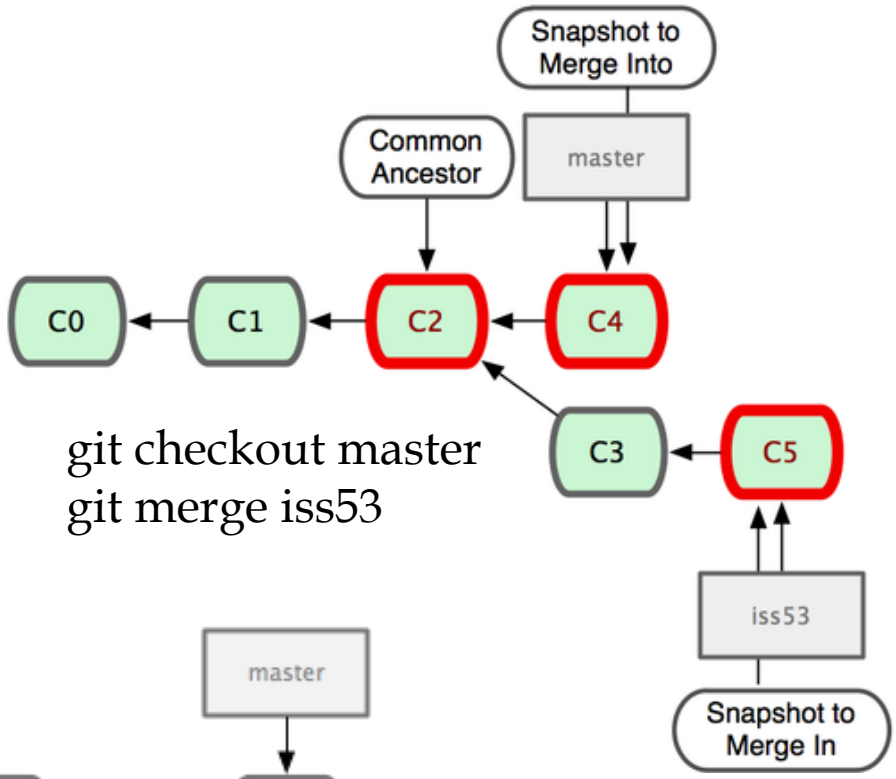


git checkout master
git merge hotfix

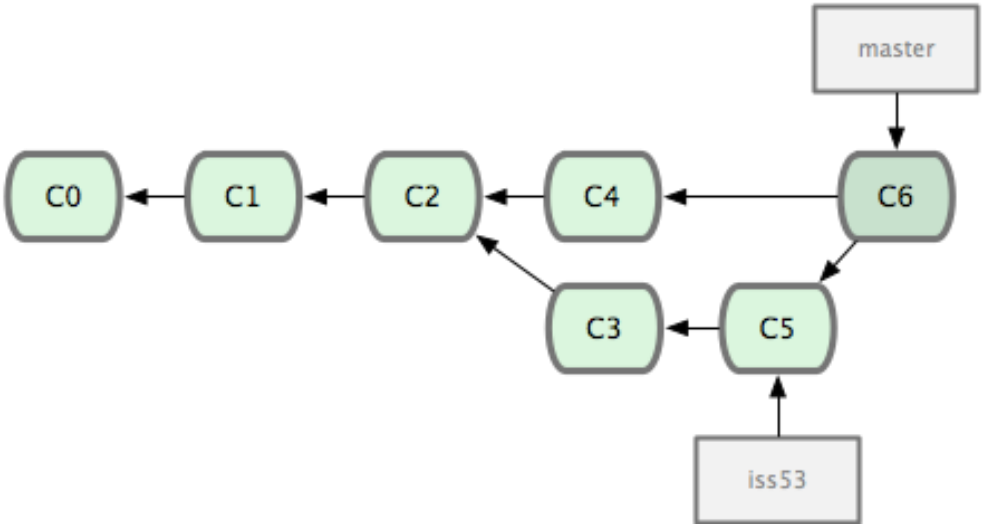
Creating and merging branches in Git



git branch –d hotfix
git checkout iss53
git commit

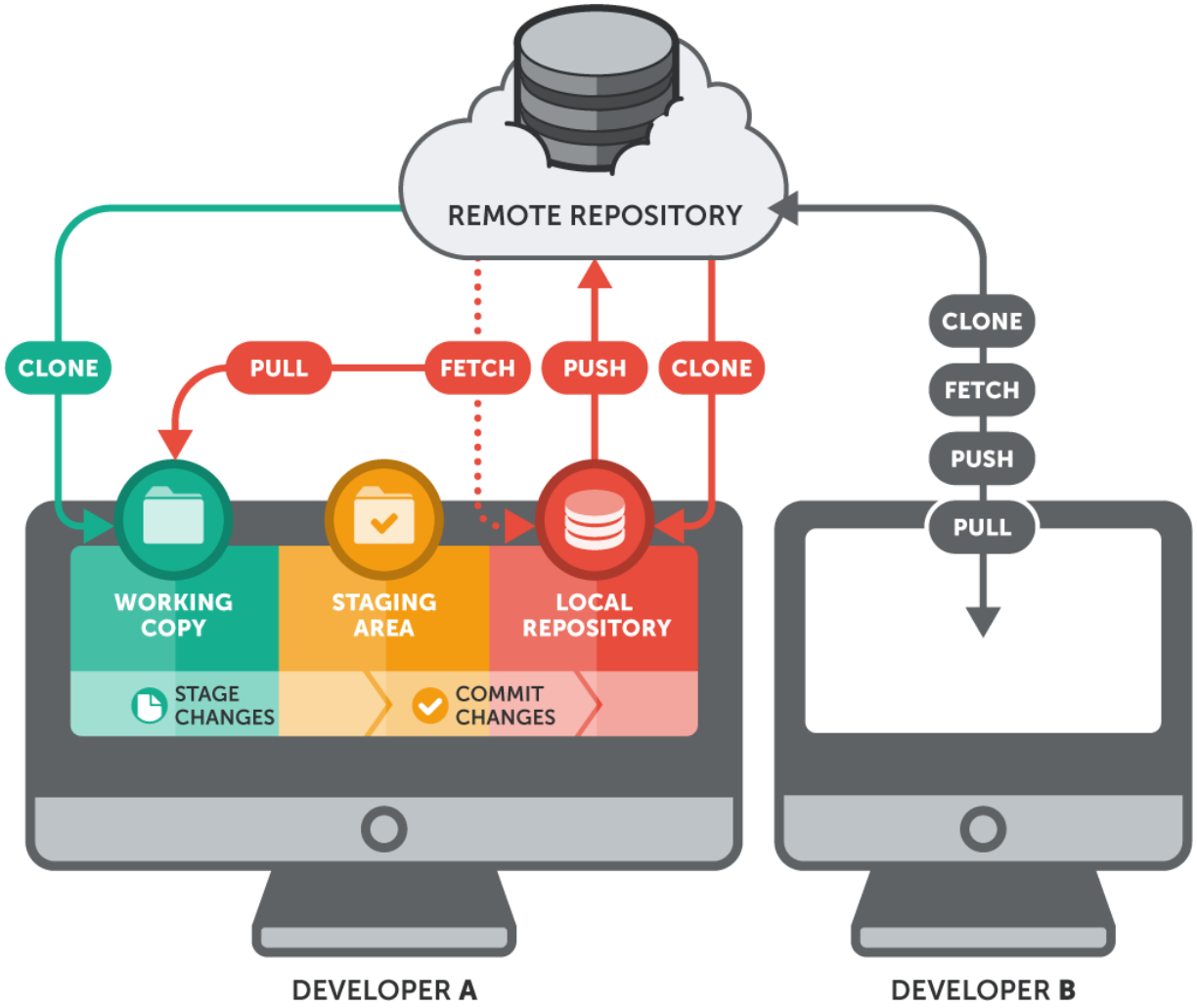


git checkout master
git merge iss53

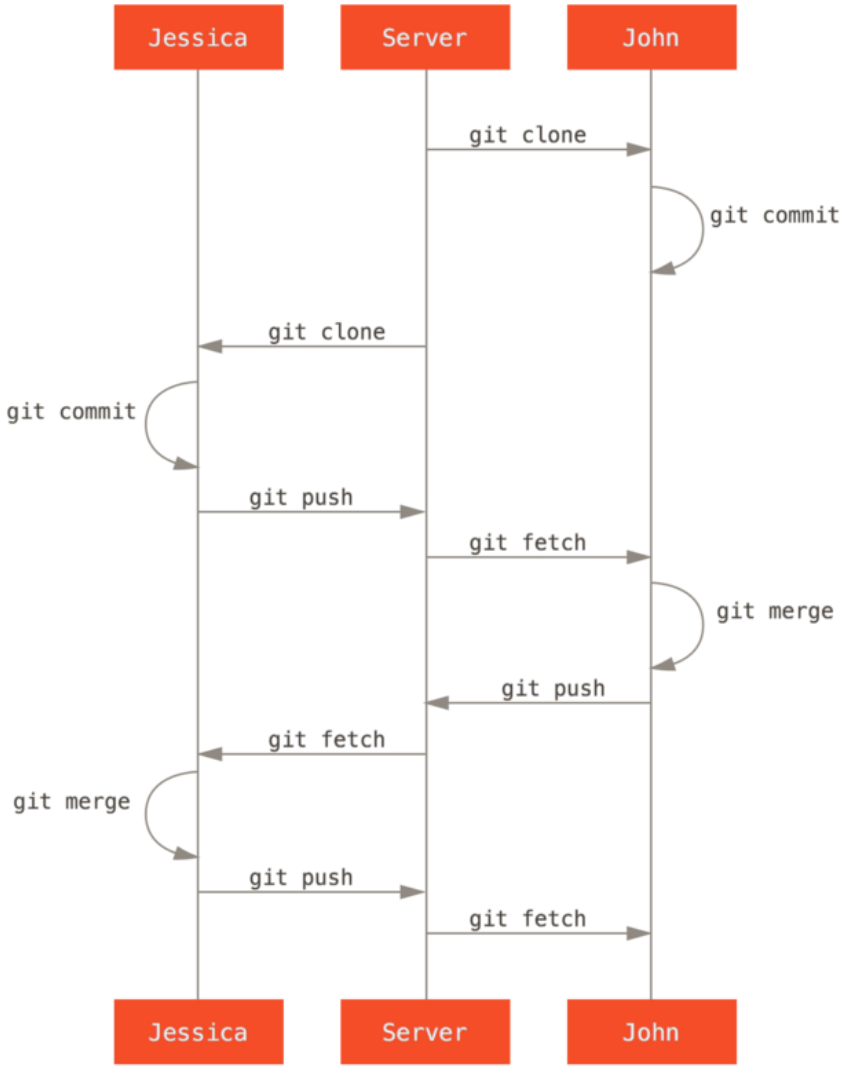


Git supports collaboration

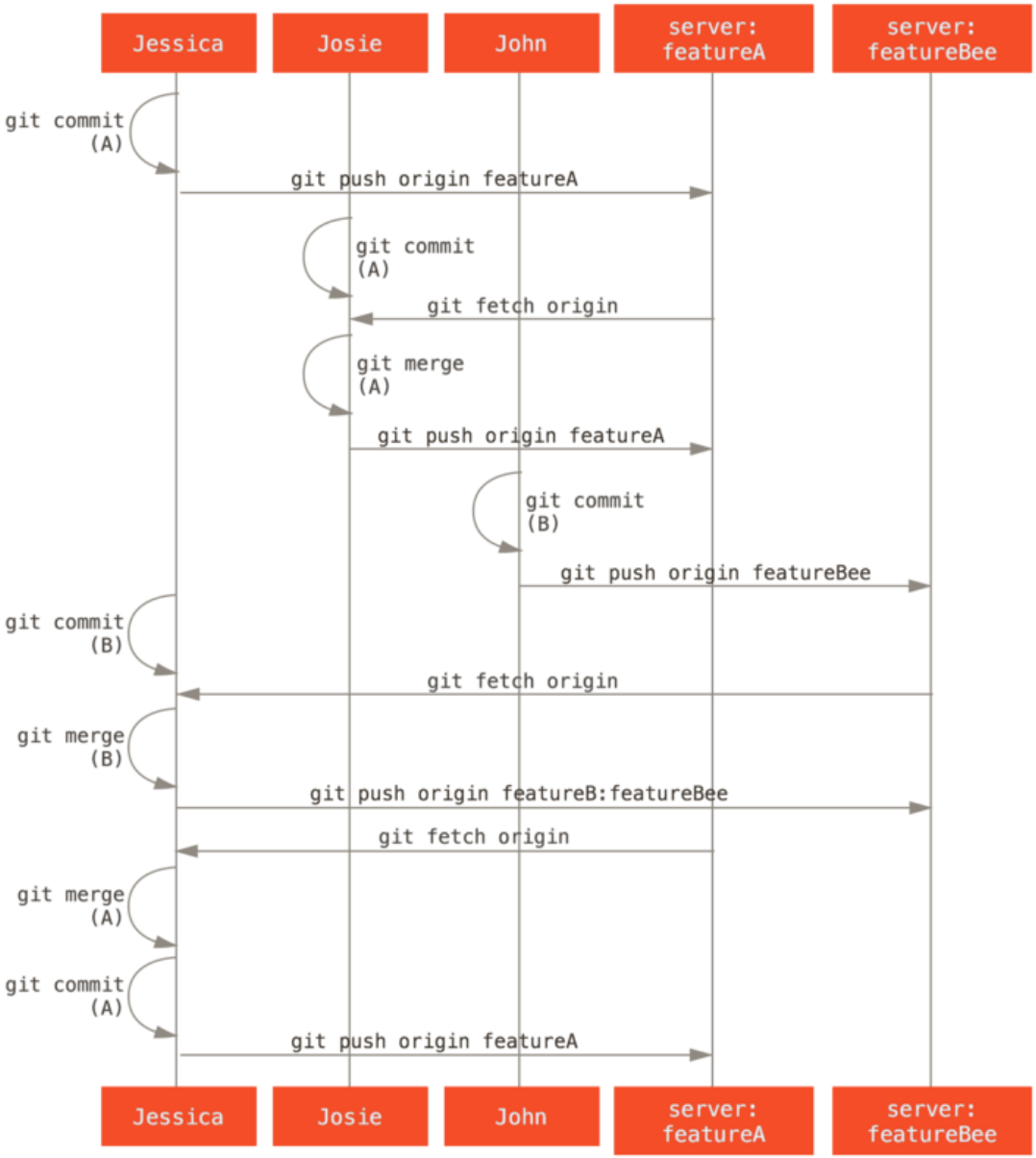
- **Local repository and Remote Repository**



Git supports collaboration

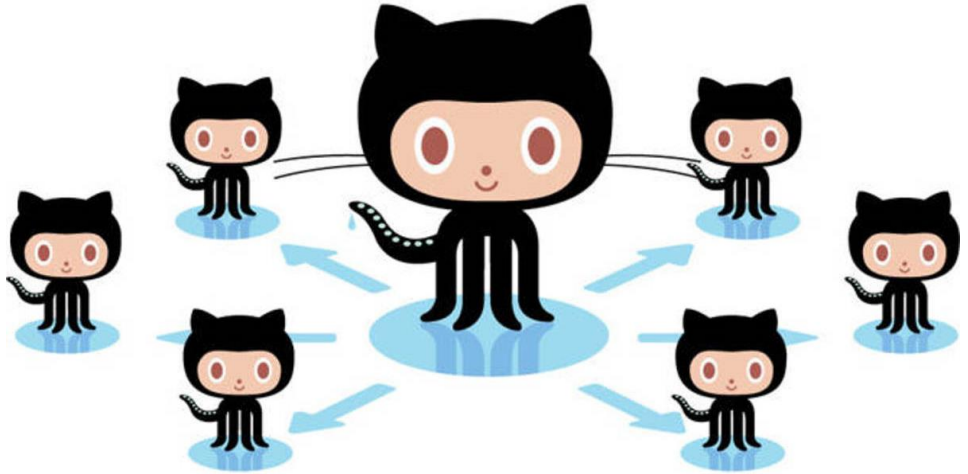


Git supports collaboration



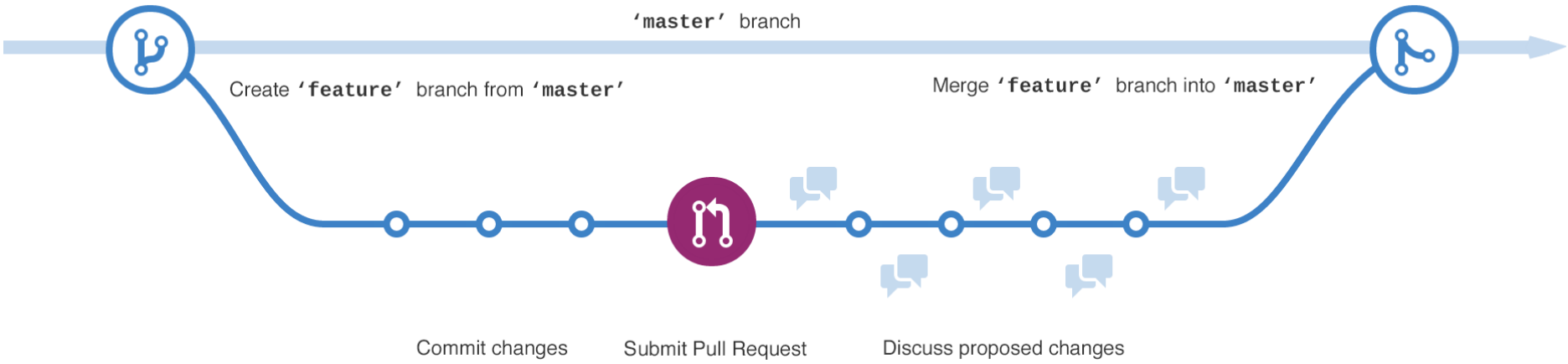
GitHub

- **GitHub: a web-based Git server and Internet hosting service.**
  - It offers all of the distributed version control and SCM functionality of Git as well as adding its own features.
  - It provides access control and several collaboration features such as bug tracking, feature requests, task management, and wikis for every project.
  - Private and free repositories (for open-source projects)
- **In 2016, it has more than 14 million users above 35 million repositories.**

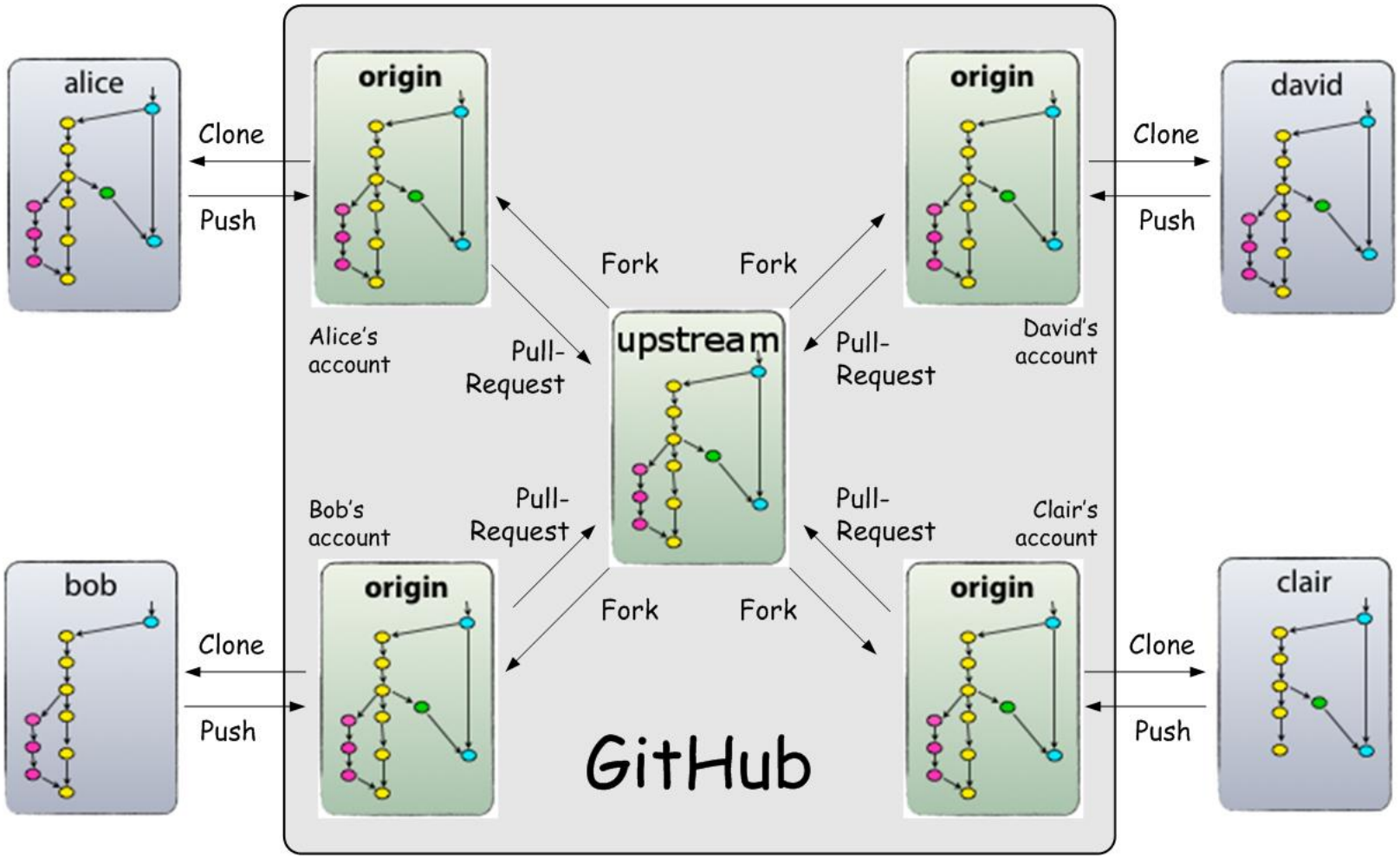


GitHub working process

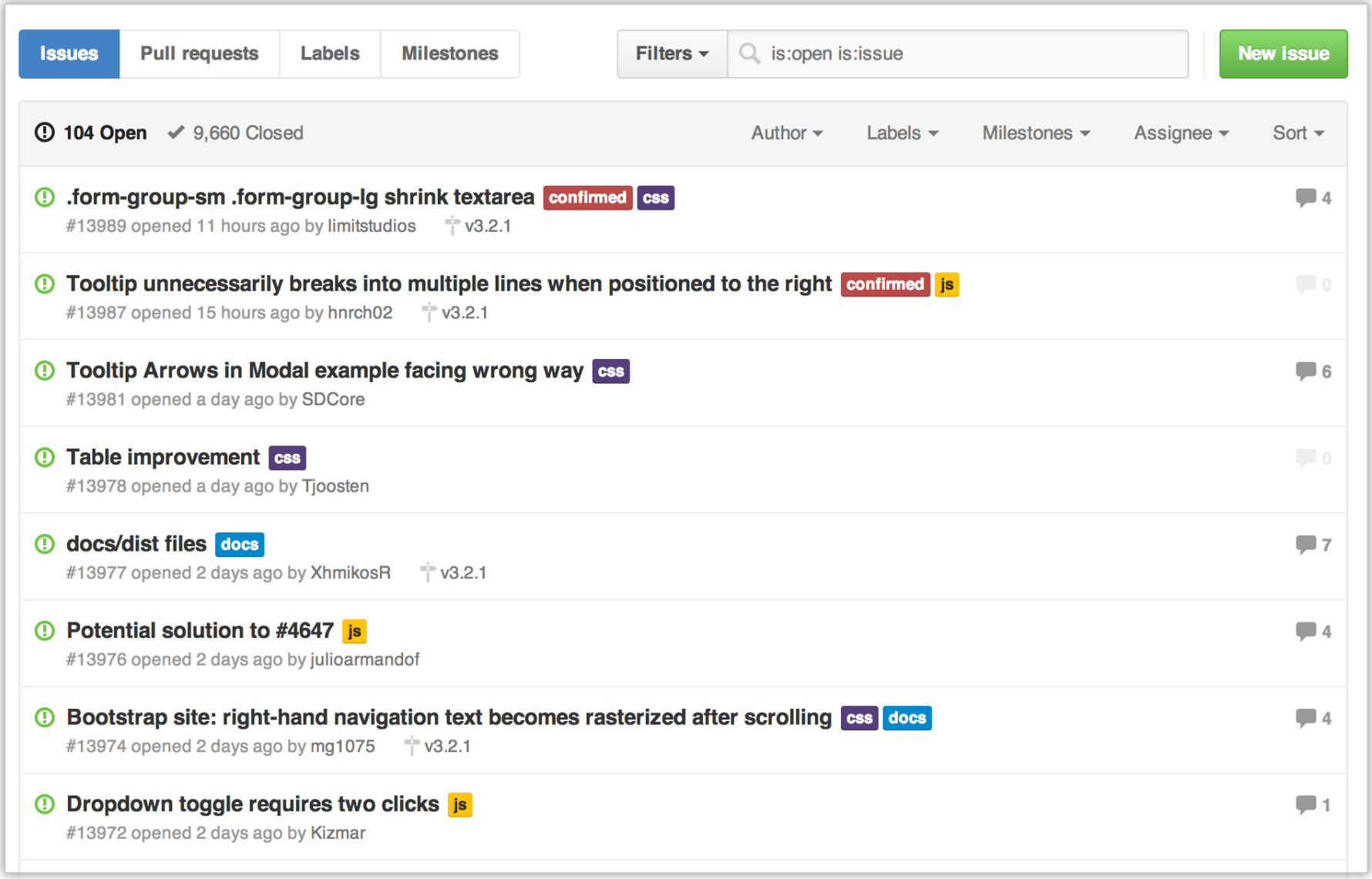
- **Basic process: commit, branch and merge**
- **Collaboration process: fork and pull request**



GitHub working process



Issue Tracking and Pull Request



6 Summary

Summary

- **General Software Development Lifecycle (SDLC)**
- **Traditional software process models**
  - Waterfall, incremental, prototype, iterative
- **Agile development**
- **Collaborative software development**
- **Software Configuration Management (SCM)**
- **Git as a SCM tool**

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

The end

February 24, 2019