哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

3 Software Construction Process and Configuration Management

软件构造过程与配置管理

刘铭

2021/5/4

Objectives of this lecture

- To know the general process of software development 软件开发的基本过程
- To understand the philosophy of traditional software process models including linear and iterative models (waterfall, incremental, prototyping, spiral, and V-model) 传统的软件开发过程模型
- To know and make practice of Agile development 敏捷开发
- To understand Software Configuration Management (SCM) 软件配置管理
- To learn how to use Git for daily SCM tasks (basic commands for personal dev., advanced commands for collaborative dev) Git作为配置管理工具

Objective of this lecture

- **Get to know the general process of software construction (Design ⇒ Programming ⇒ Debug ⇒ Testing ⇒ Build ⇒ Release) 广义的软件构造过程**
- **Use Eclipse IDE as Java construction environment and tools Eclipse作为Java构造工具**
- **Get to know typical tools for review and static analysis, debug (dumping, logging) and testing, and dynamic analysis / profiling 软件构造各阶段的常见工具**
- **Learn the narrow-sense process of software construction (Build: Validate ⇒ Compile ⇒ Link ⇒ Test ⇒ Package ⇒ Install ⇒ Deploy) 狭义的软件构造：build**
- **Use one of build tools (make, Ant, Maven, Gradle, Eclipse IDE) to build your own Java projects 常见的build工具**

Outline

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

第1次课：软件构造的结果形态、如何是“好”
本次课关心：软件开发(0→1→…→n)遵循什么过程

Outline

以及：每个阶段内部的“子过程”

- **General process of software construction:** Design ⇒ Programming /refactoring ⇒ Debugging ⇒ Testing ⇒ Build ⇒ Release
  - Programming / refactoring
  - Review and static code analysis
  - Debugging (dumping and logging) and Testing
  - Dynamic code analysis /profiling
- **Narrow-sense process of software construction (Build):** Validate ⇒ Compile ⇒ Link ⇒ Test ⇒ Package ⇒ Install ⇒ Deploy
  - Build system: components and process
  - Build variants and build language
  - Build tools: Make, Ant, Maven, Gradle, Eclipse/IDEA/NetBeans
- **Summary**

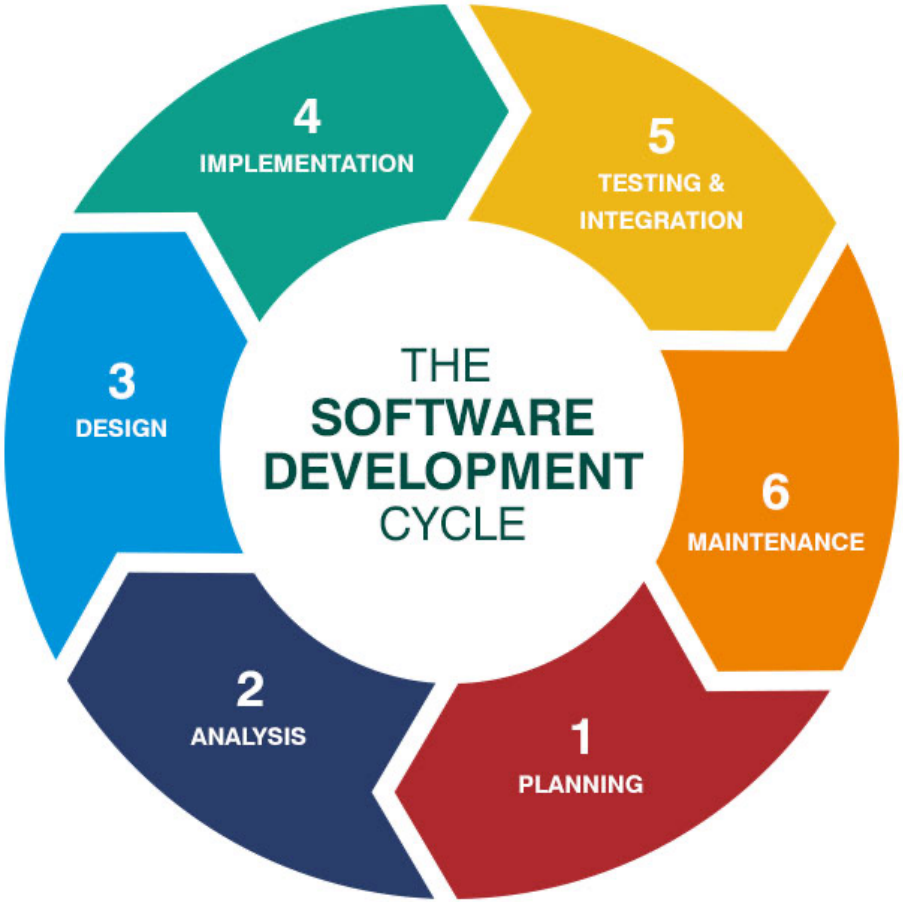
Reading

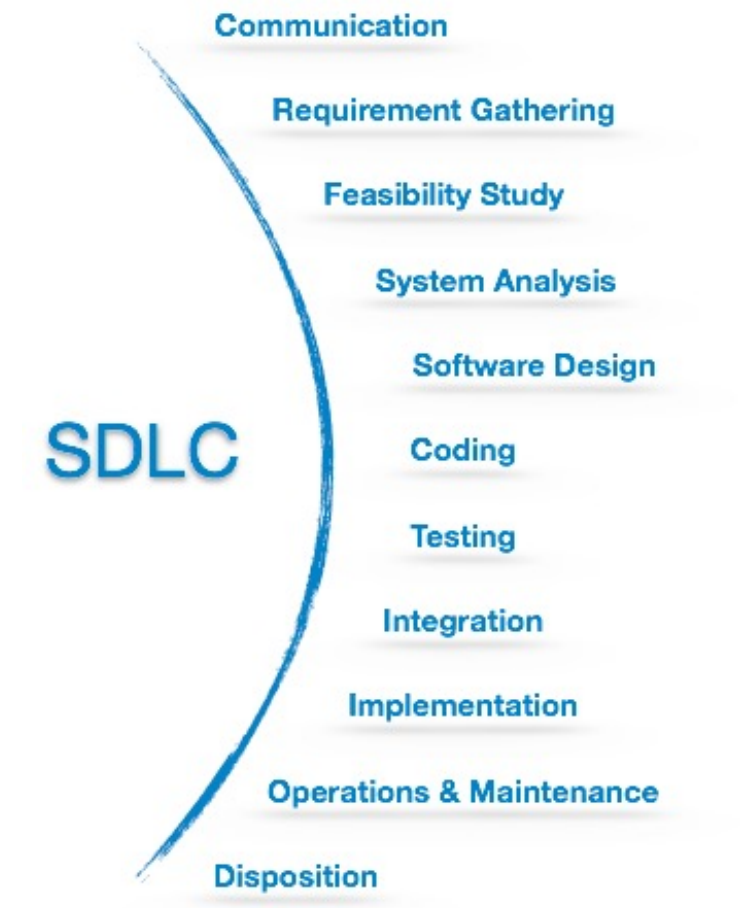
- **MIT 6.031：05、28**
- **CMU 17-214：Nov 19、Nov 21**
- **软件工程--实践者的研究方法：第2-3、22章**
- **代码大全：第21、27-30章**
- **深入理解软件构造系统：第1-5章、第7章、第10章**
- **持续集成实践：第1-5章**

1 Software Development Lifecycle (SDLC)

Lifecycle of a software

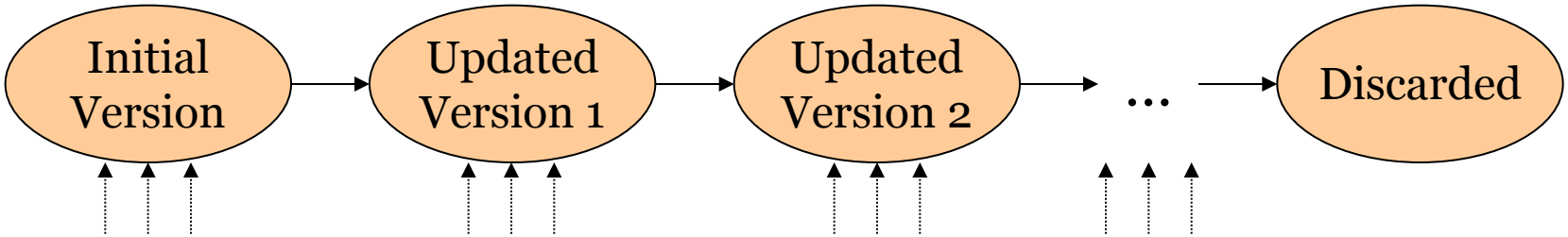
- **Software Development Life Cycle (SDLC): From 0 to 1 从无到有**

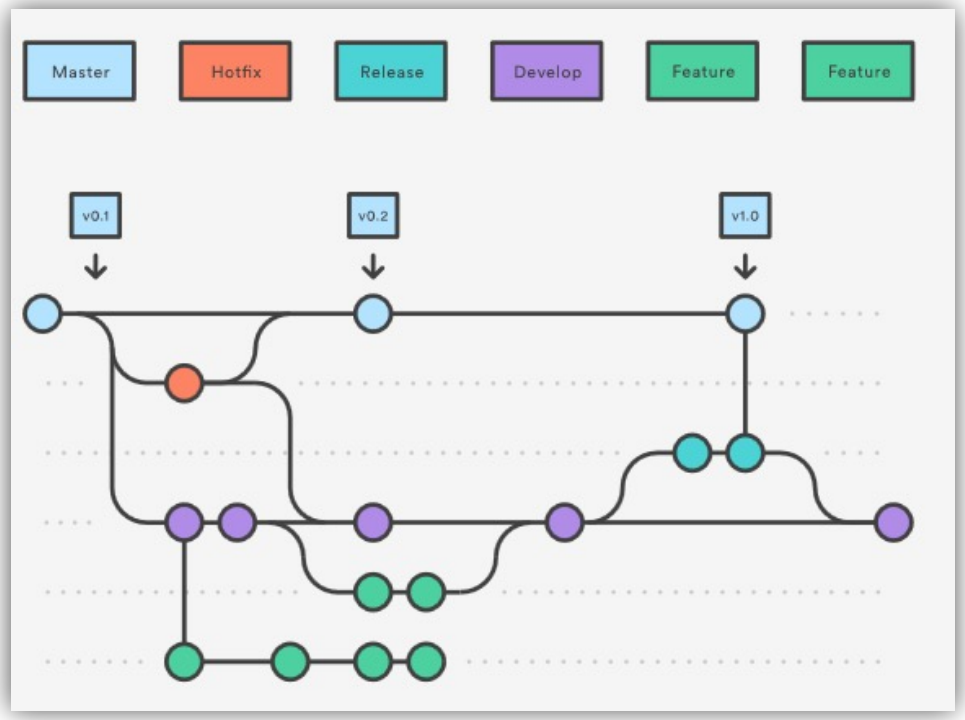


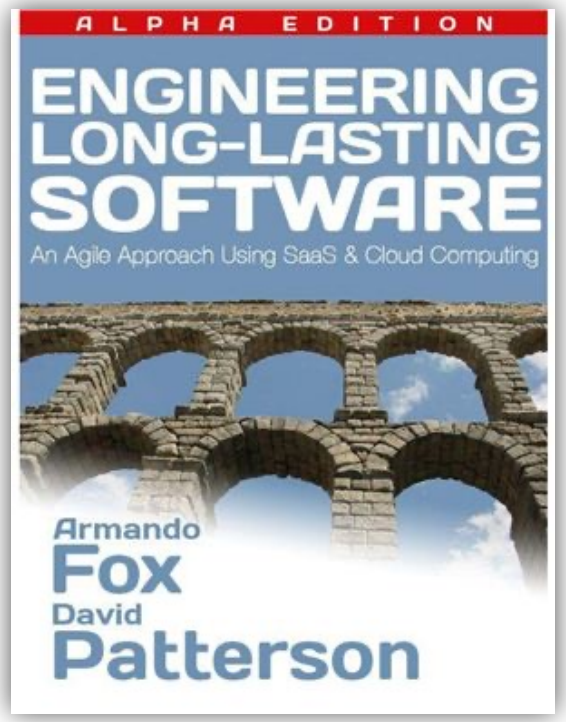


Lifecycle of a software

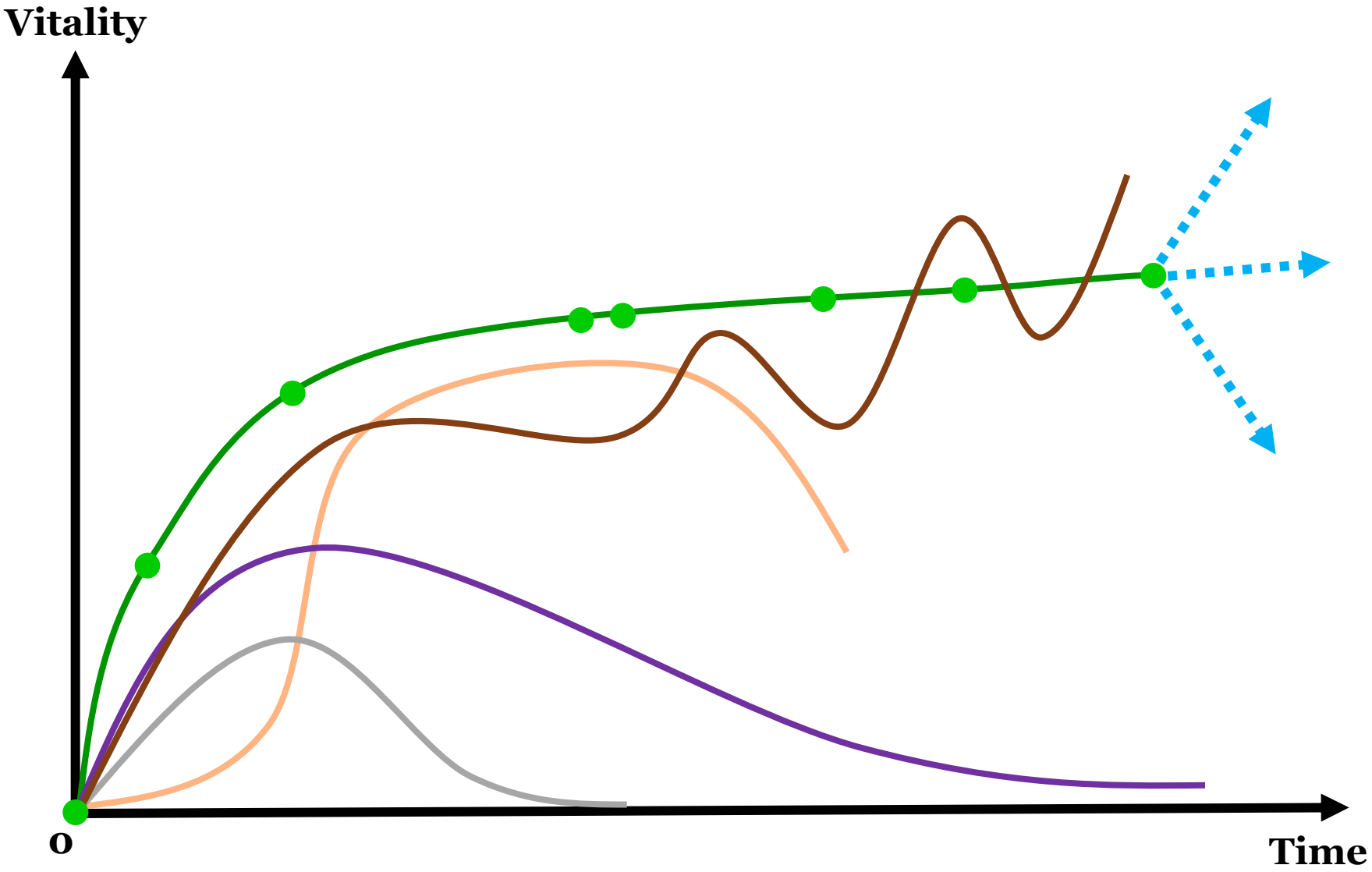
- **Multiple versions in the life of a software: From 1 to n 从有到好**







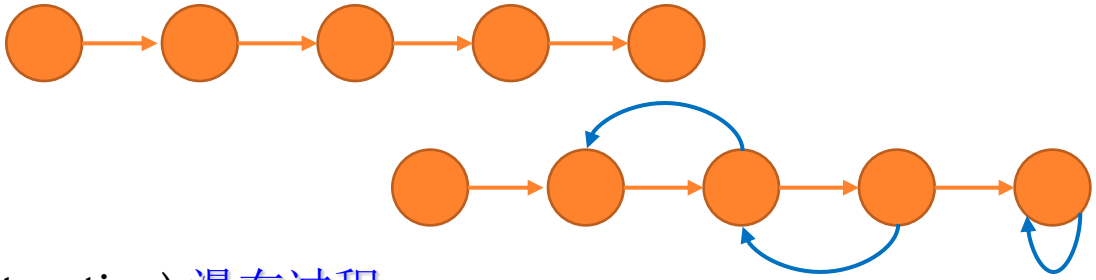
Life patterns of software



2 Traditional Software Process Models

Traditional software process models

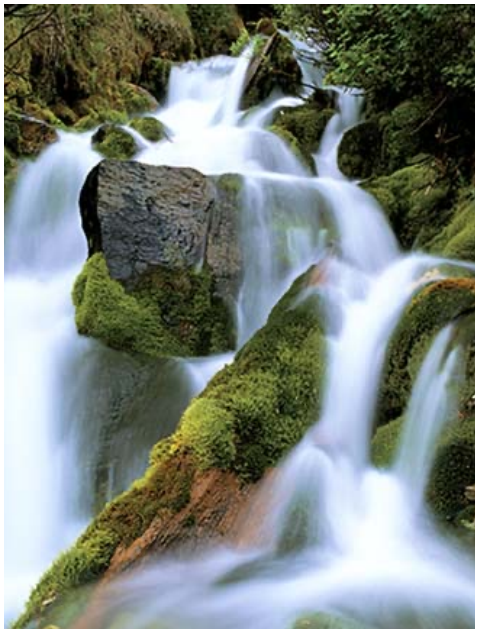
- **Two basic types:**
  - Linear 线性过程
  - Iterative 迭代过程

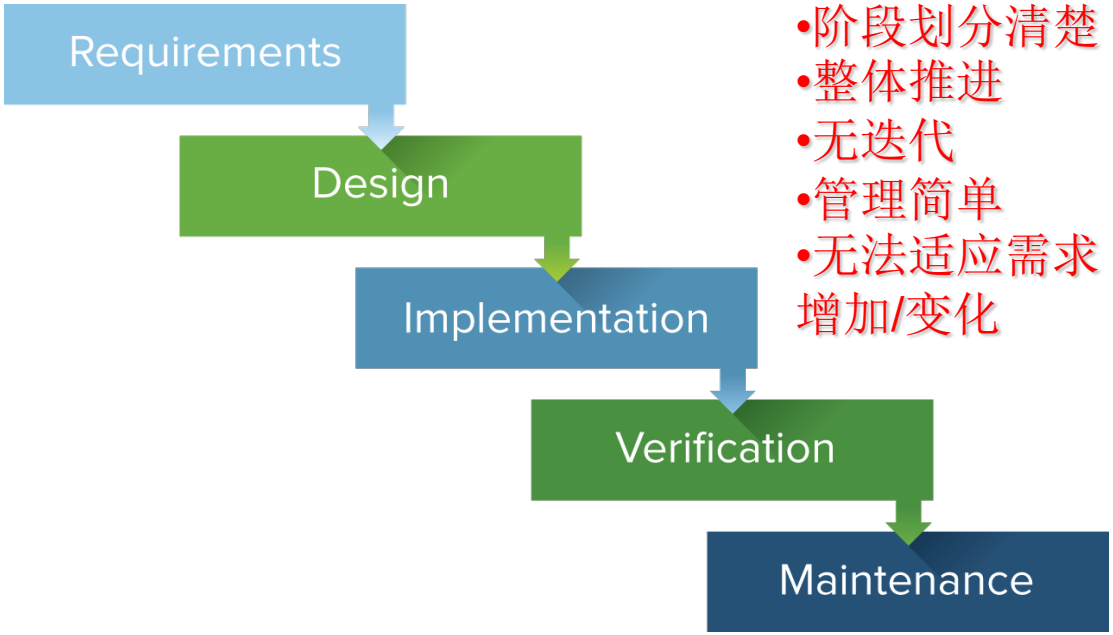


- **Existing models:**
  - Waterfall (Linear, non-iterative) 瀑布过程
  - Incremental (non-iterative) 增量过程
  - V-Model (for verification and validation) V字模型
  - Prototyping (iterative) 原型过程
  - Spiral (iterative) 螺旋模型
- **Key quality considerations 选择合适的过程模型的依据:**
  - User involvement (adapt to changes) 用户参与程度有多大？--适应变化的能力
  - Development efficiency, management complexity 开发效率/管理复杂度
  - Quality of software 开发出的软件的质量

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.**

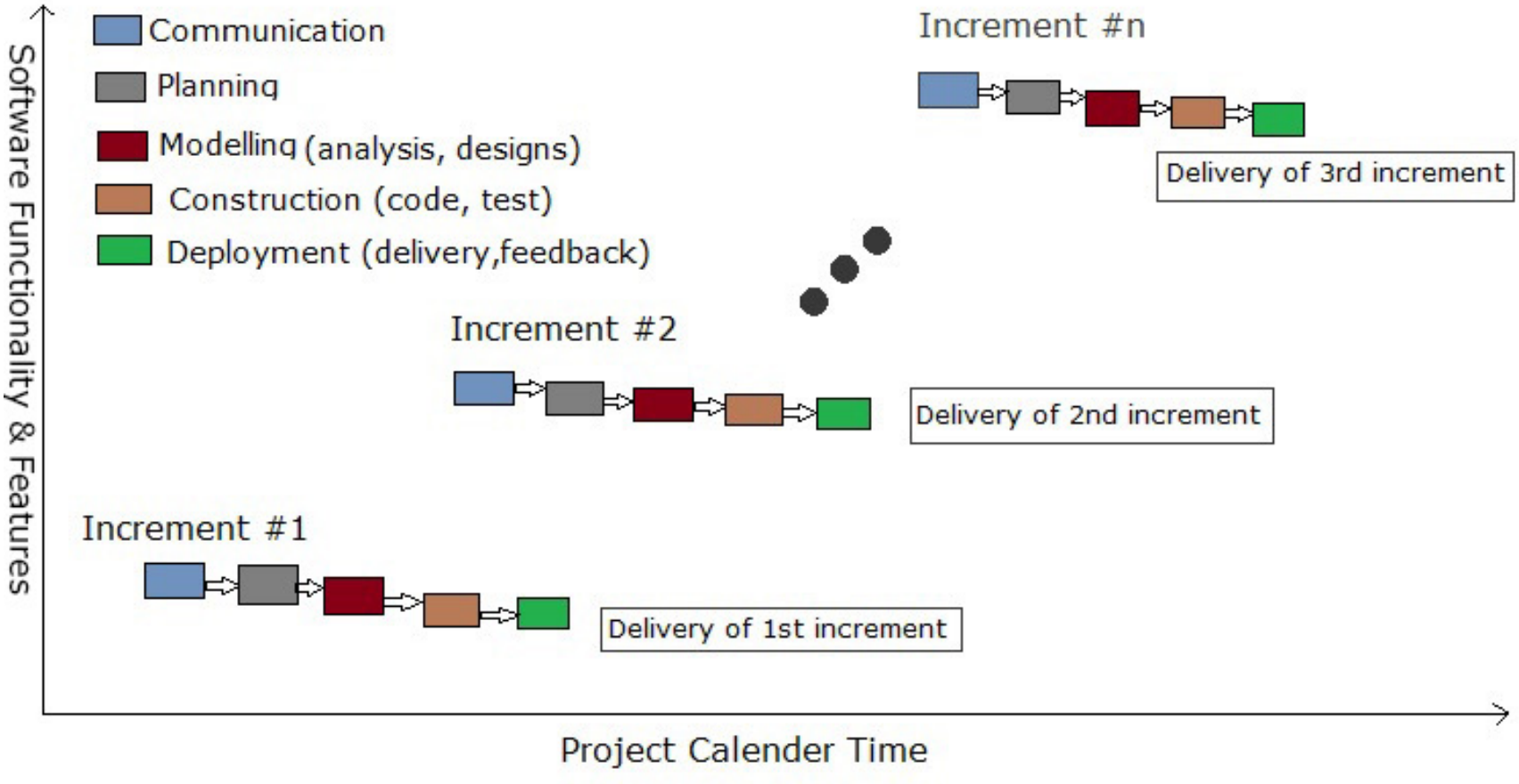




瀑布过程：
•线性推进
•阶段划分清楚
•整体推进
•无迭代
•管理简单
•无法适应需求增加/变化

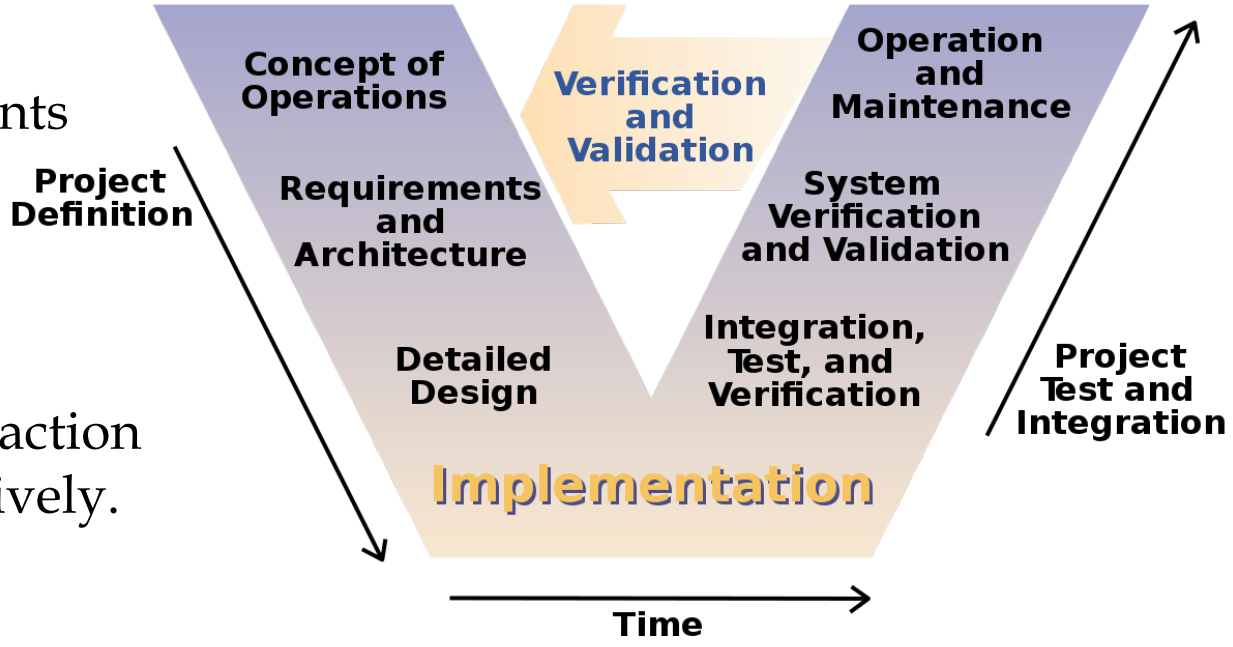
Incremental (non-iterative)

增量过程：
- 线性推进
- 增量式（多个瀑布的串行）
- 无迭代
- 比较容易适应需求的增加



V-Model (for verification and validation)

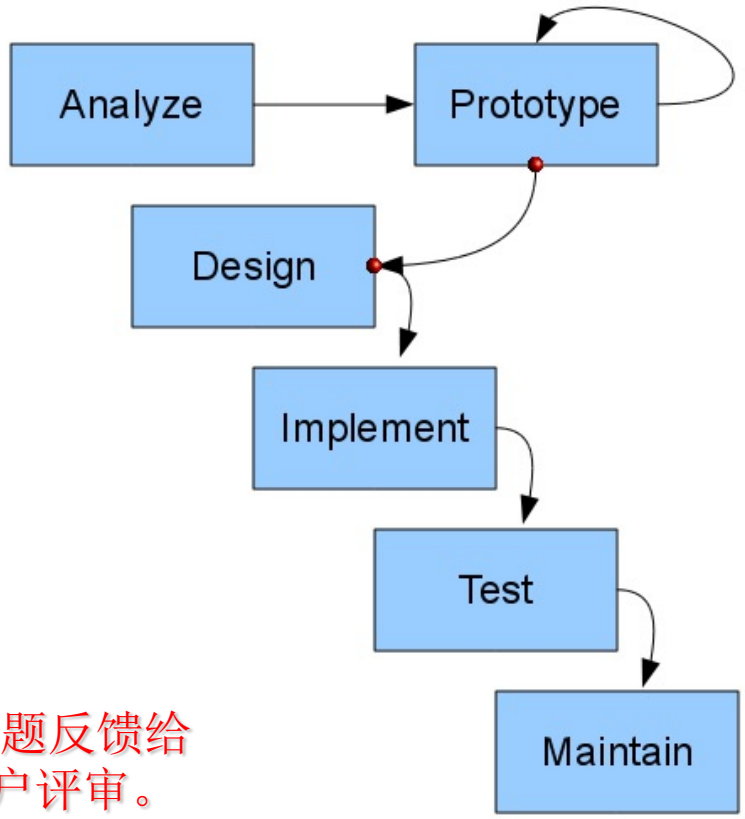
- **V-model represents a development process that may be considered an extension of the waterfall model.**
  - Instead of moving down in a linear way, the process steps are bent upwards after the coding phase, to form the typical V shape.
  - Demonstrates the relationships between each phase of the development 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.



Prototyping (iterative)

在原型上持续不断的迭代
发现用户变化的需求

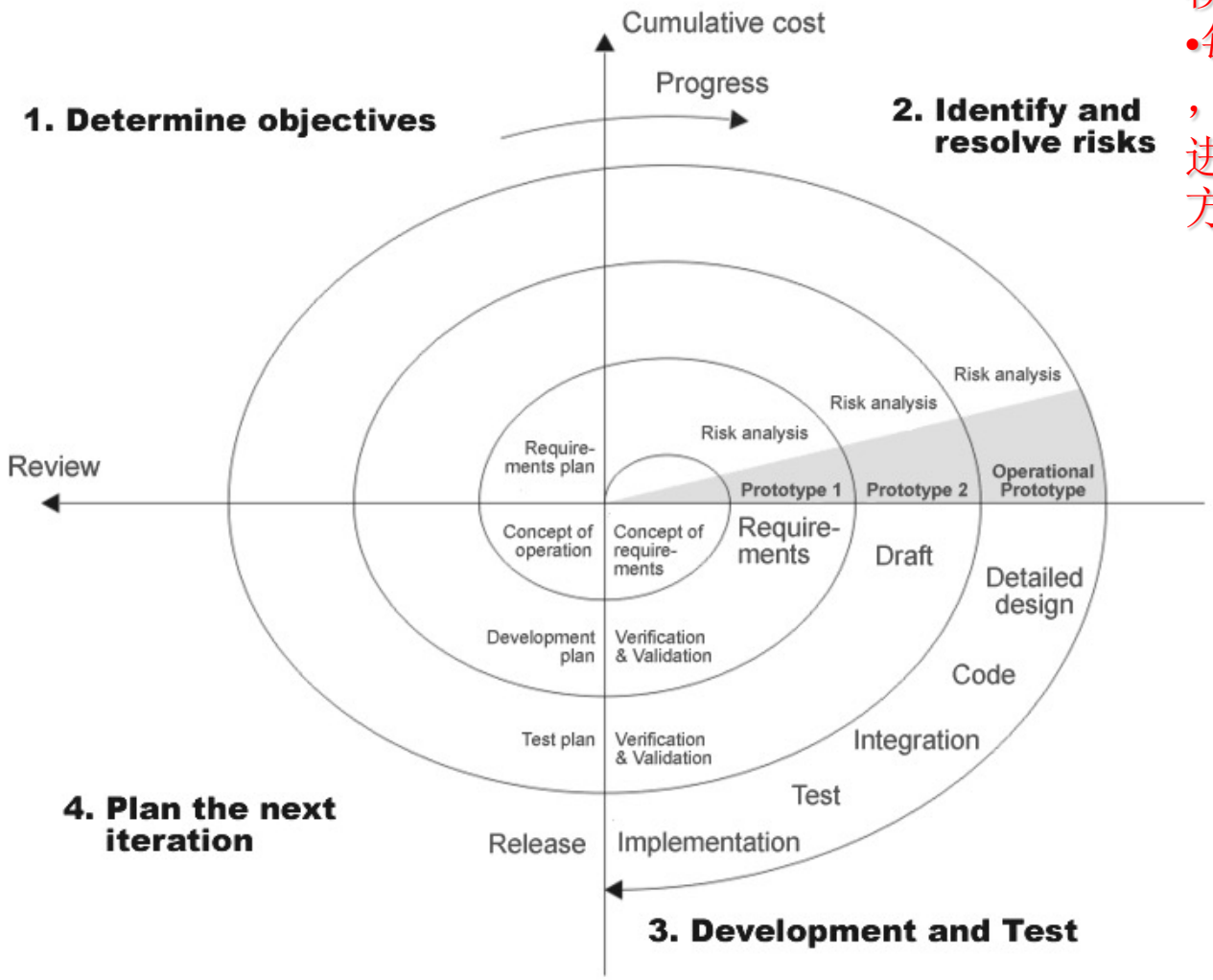
- **Benefits:**
  - The software designer and implementer can get valuable feedback from the users early in the project.
  - The client can compare if the software made matches the software specification, according to which the software program is built.
  - It also allows the software engineer some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



迭代：开发出来之后由用户试用/评审，发现问题反馈给开发者，开发者修改原有的实现，继续交给用户评审。

循环往复这个过程，直到用户满意为止。
时间代价高，但开发质量也高。

Spiral (iterative)



非常复杂的过程：

- 多轮迭代基本遵循瀑布模式
- 每轮迭代有明确的目标，遵循“原型”过程，进行严格的风险分析，方可进入下一轮迭代

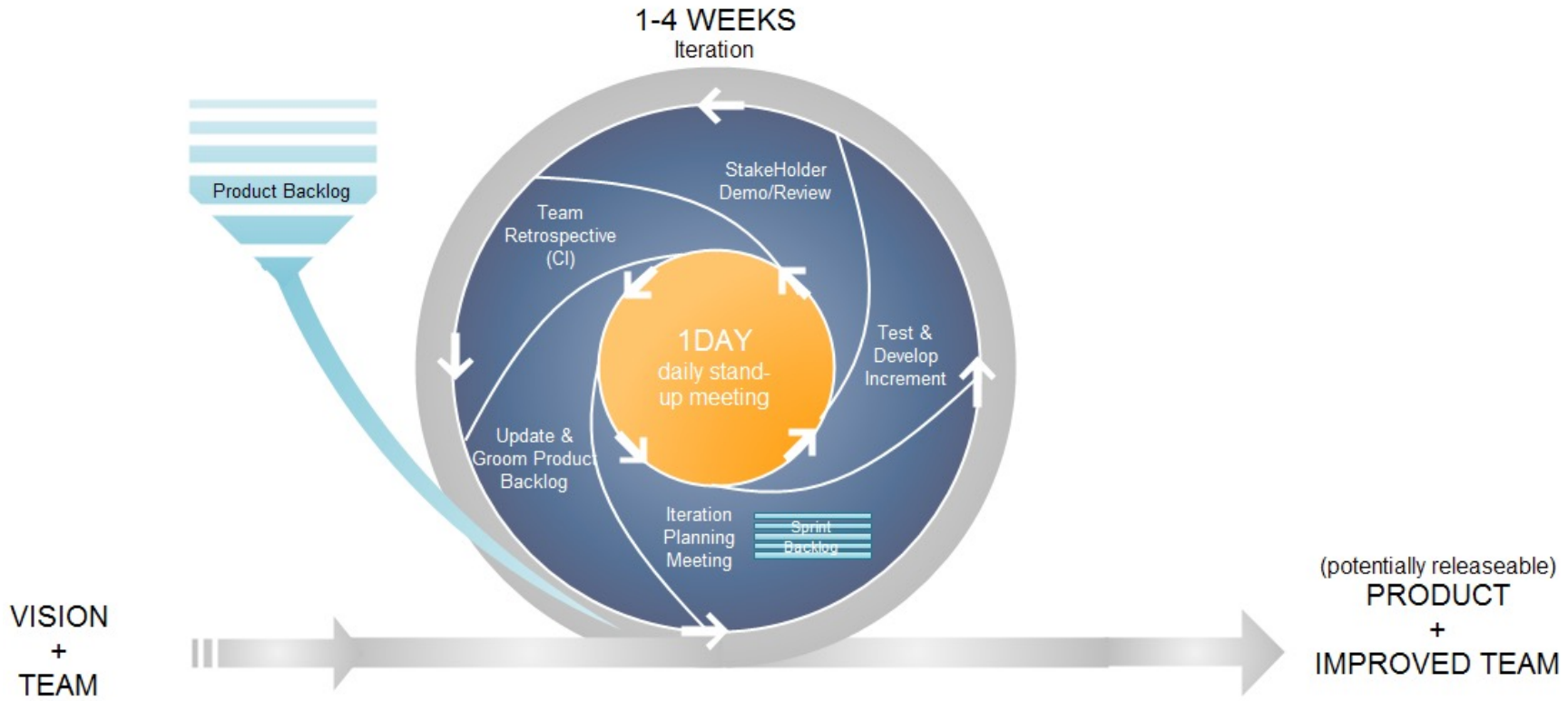
3 Agile Development

Agile development

- **It advocates adaptive planning, evolutionary development, early delivery, and continuous improvement, and it encourages rapid and flexible response to change.** 敏捷开发：通过快速迭代和小规模的持续改进，以快速适应变化。
- **Agile Manifesto** 敏捷宣言 **was firstly coined in 2001 by 17 famous "programmers".**

Agile development: rapid delivery

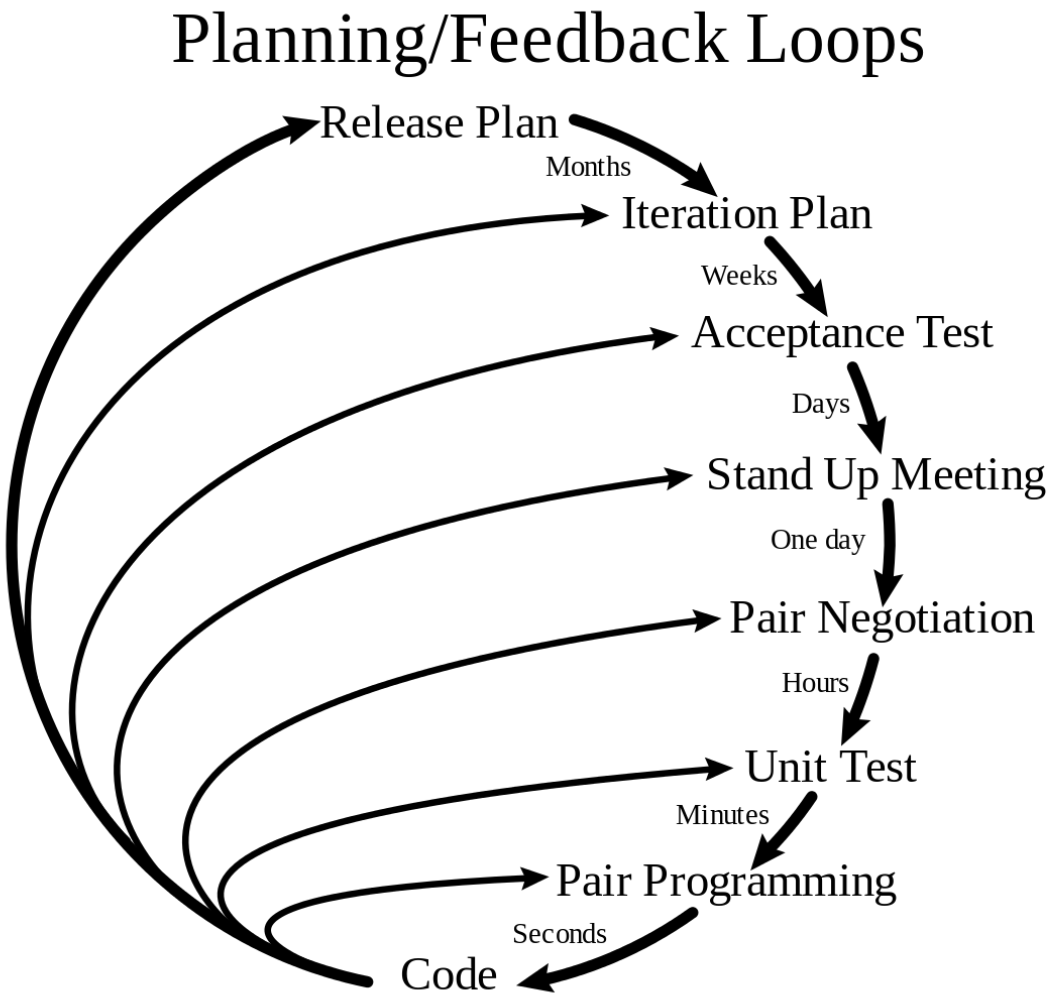
Agile = 增量 + 迭代
每次迭代处理一个小规模增量



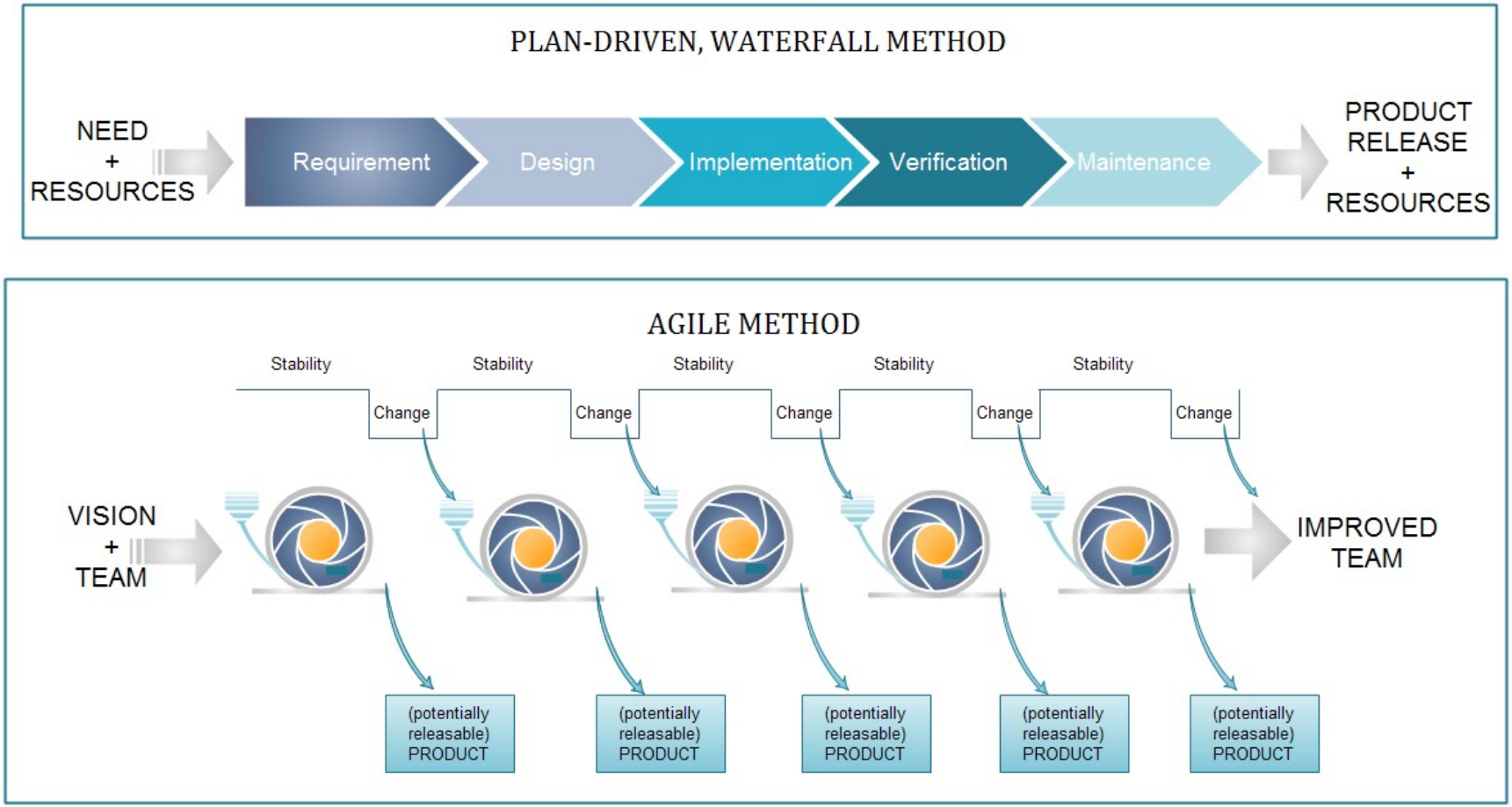
Agile development

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

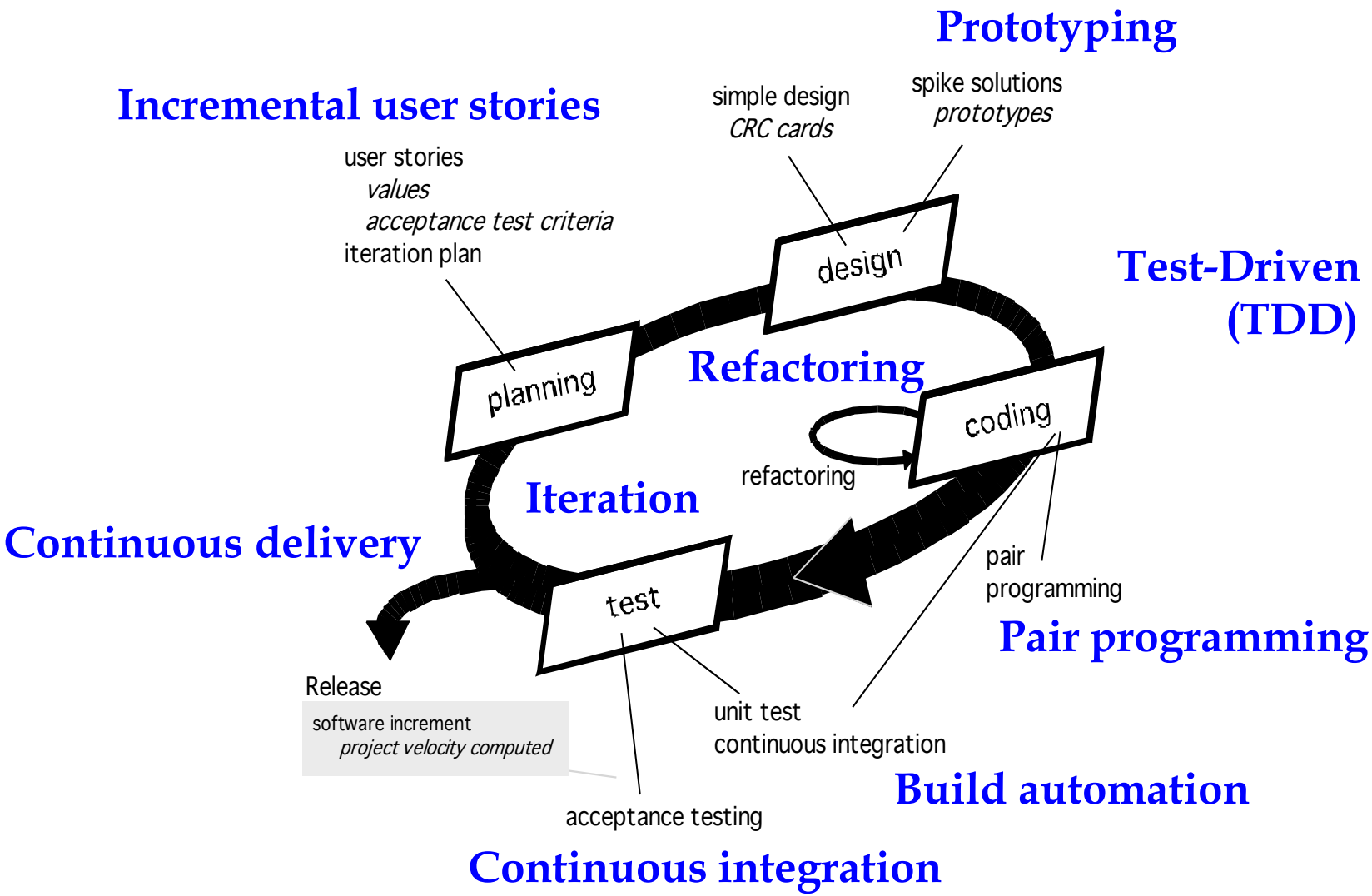
- 极限的用户参与
- 极限的小步骤迭代
- 极限的确认/验证



Waterfall vs. Agile

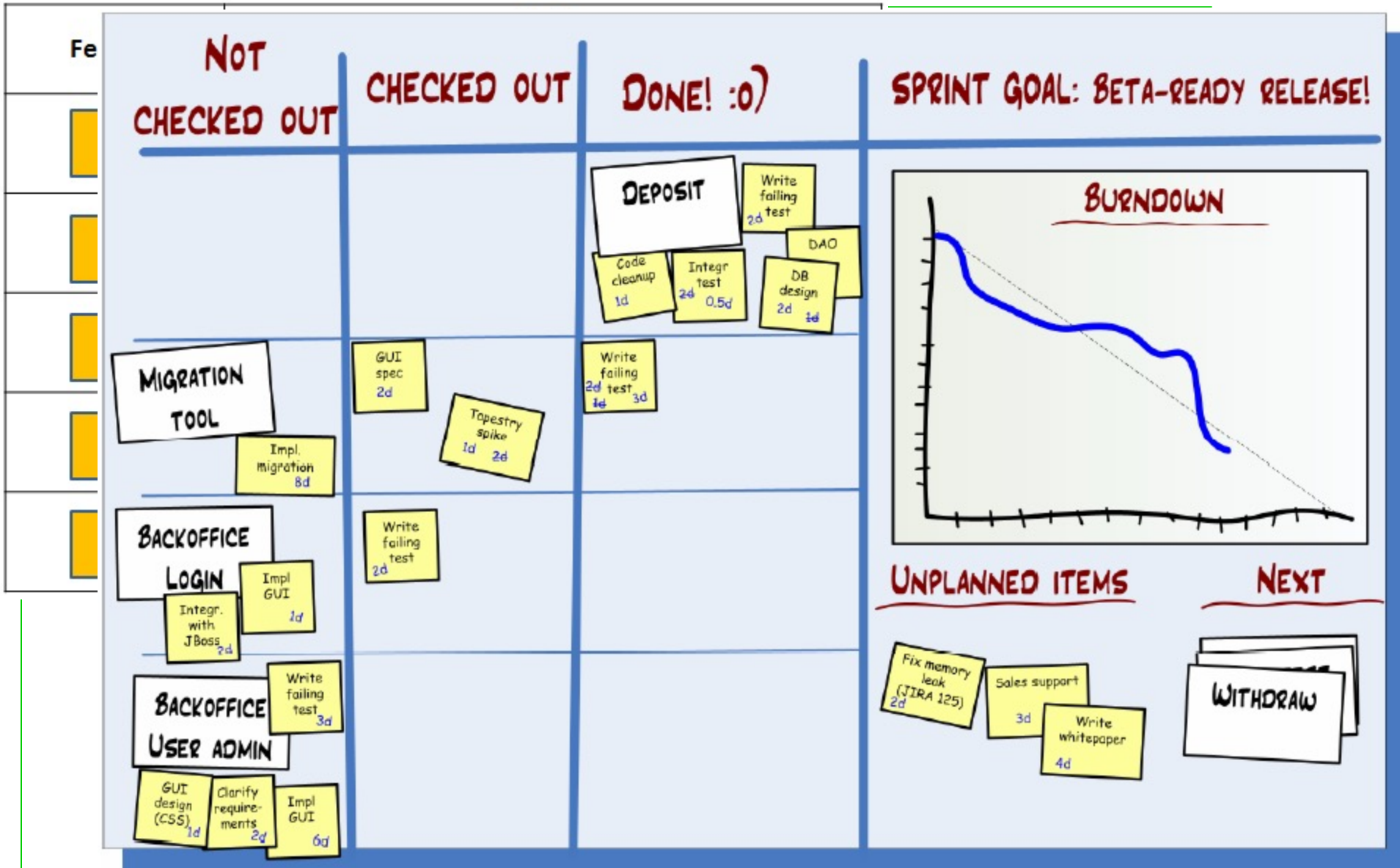


eXtreme Programming (XP，极限编程)



Pair Programming

Task board and progress monitoring



《硅谷》中的敏捷开发和task board

硅谷 第一季 Silicon Valley Season 1 (2014)

导演: 迈克·乔吉 / 亚力克·博格 / 玛姬·凯瑞 / 翠西亚·布洛克

编剧: 迈克·乔吉 / 约翰·奥茨舒勒 / 亚力克·博格 / 克雷·塔沃 / 戴夫·卡林斯基 / 罗恩·韦纳

主演: 托马斯·米德蒂奇 / T·J·米勒 / 乔什·布雷纳 / 马丁·斯塔尔 / 库梅尔·南贾尼 / 更多...

类型: 喜剧

豆瓣评分

9.1 ★★★★½ 42663人评价

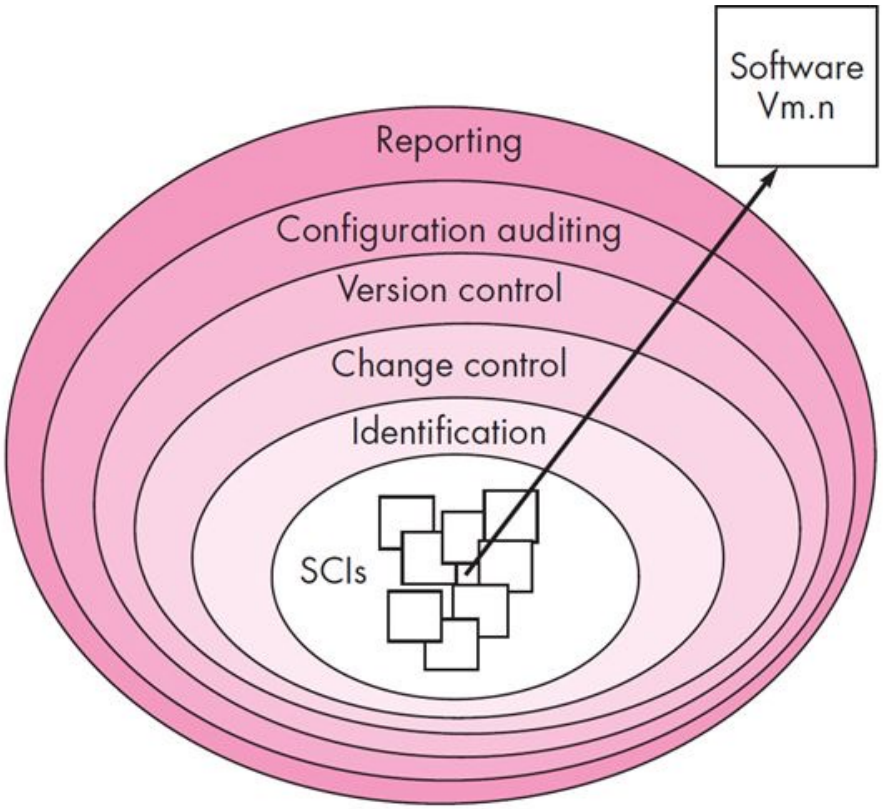
5星 63.9%
4星 30.1%
3星 5.3%
2星 0.5%

哈爾濱工業大學
HARBIN INSTITUTE OF TECHNOLOGY

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

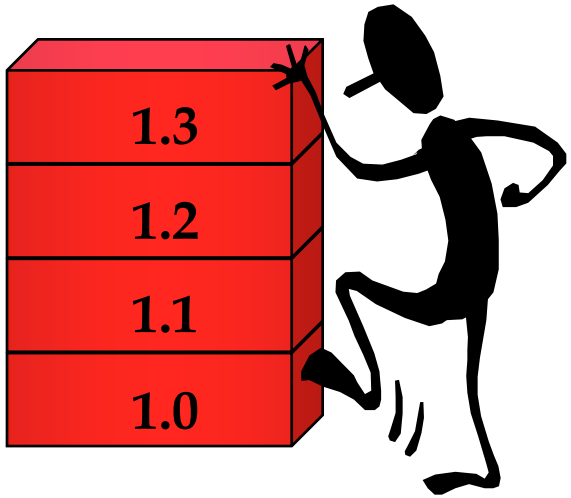
Software Configuration Mgmt. (SCM)

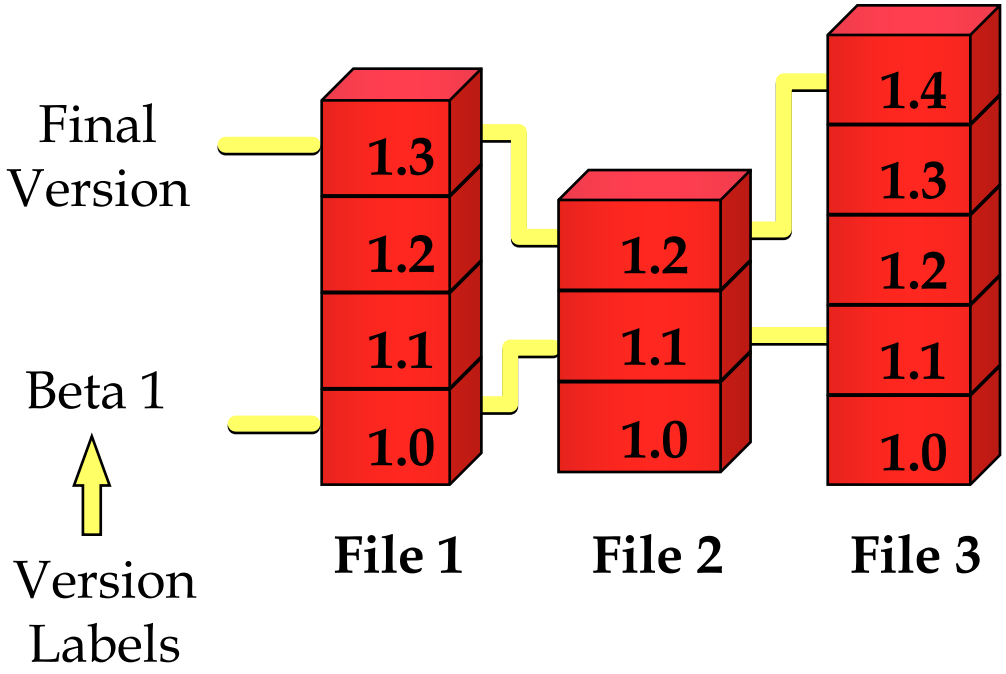
- **SCM is the task of tracking and controlling changes in the software.** 软件配置管理：追踪和控制软件的变化
- **SCM practices include revision control and the establishment of baselines.**



Life Cycle of a Configuration Item (CI)

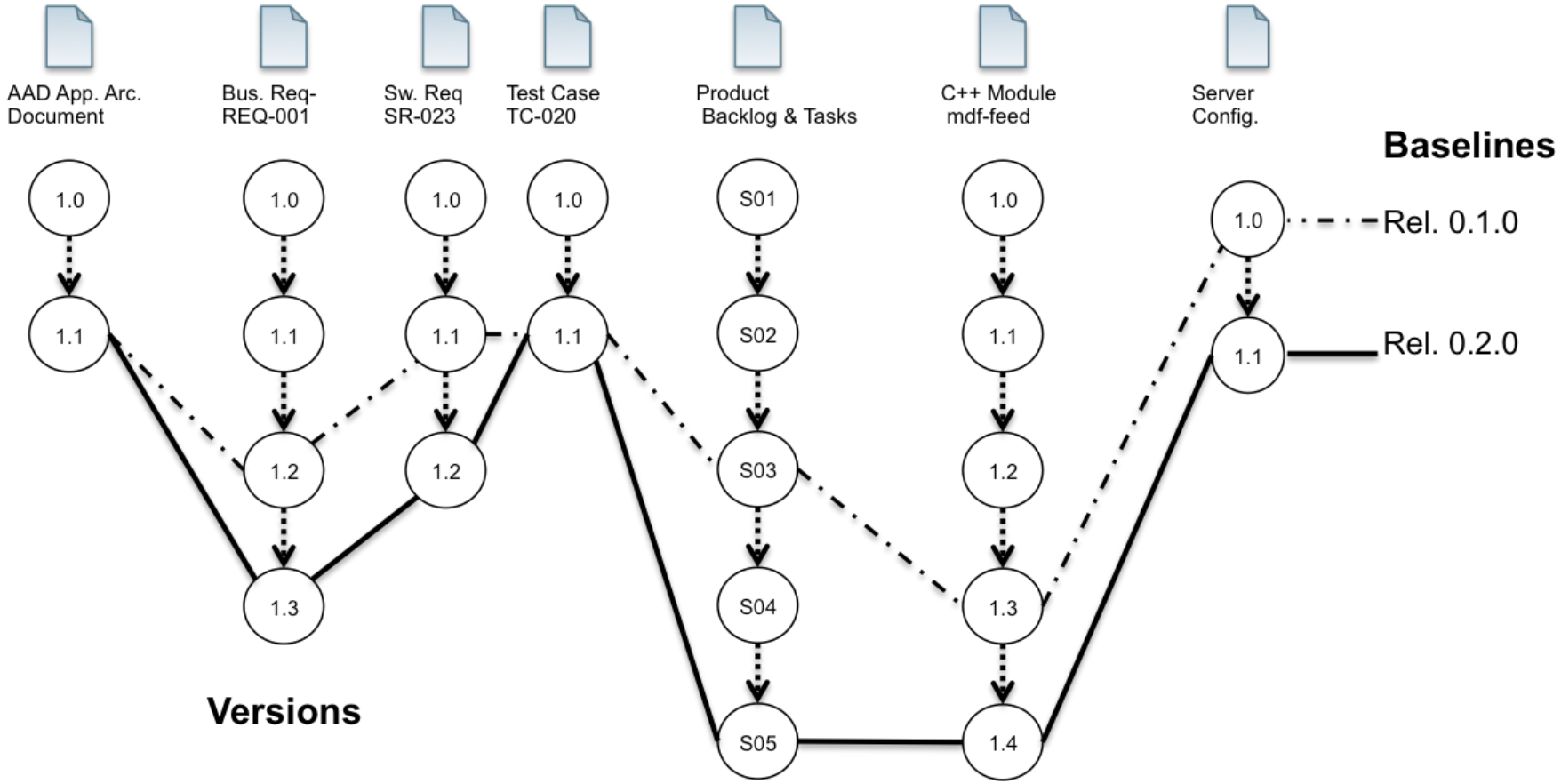
- **Any constituents of a software (source code, data, documents, hardware, various environments) may be updated along with the time in the life cycle of the software.**
- **Software Configuration Item (SCI): the fundamental structural unit of SCM. 软件配置项：软件中发生变化的基本单元（例如：文件）**



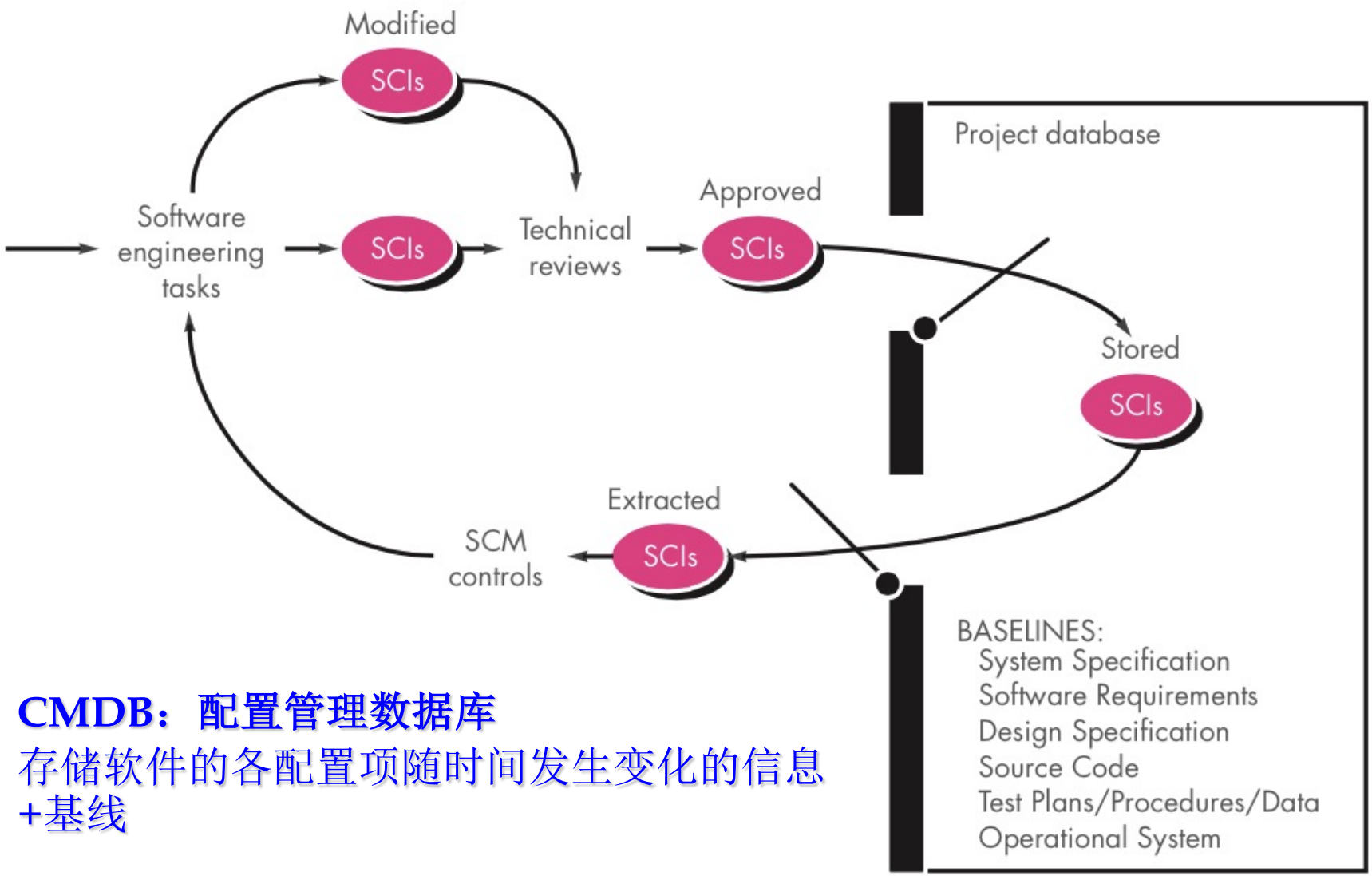


Configuration Items (SCI) and Baselines

- **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.**
  基线：软件持续变化过程中的“稳定时刻”（例如：对外发布的版本）



CMDB and Check-in/Check-out for Auditing

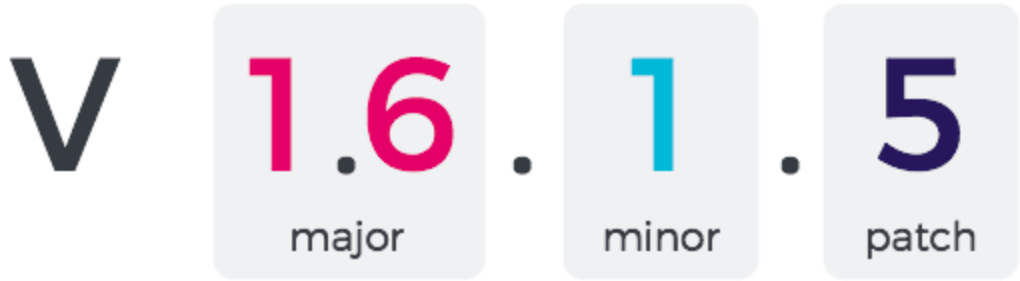


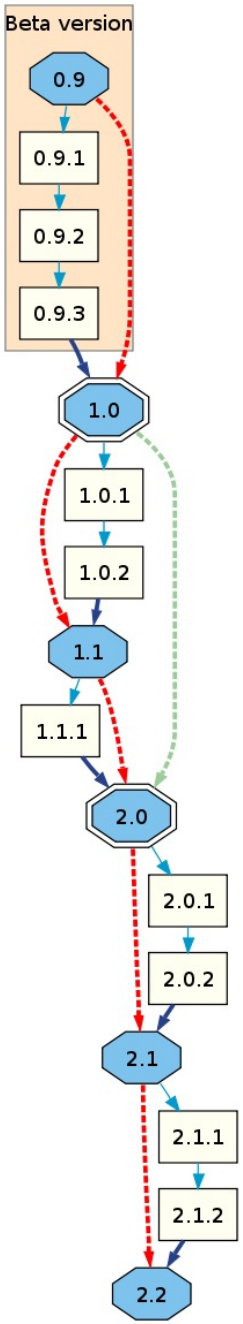
CMDB：配置管理数据库

存储软件的各配置项随时间发生变化的信息+基线

Versioning 版本控制

- **Versioning is the process of assigning either unique version names or unique version numbers to unique states of software.** 版本：为软件的任一特定时刻（Moment）的形态指派一个唯一的编号，作为“身份标识”
  - 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.



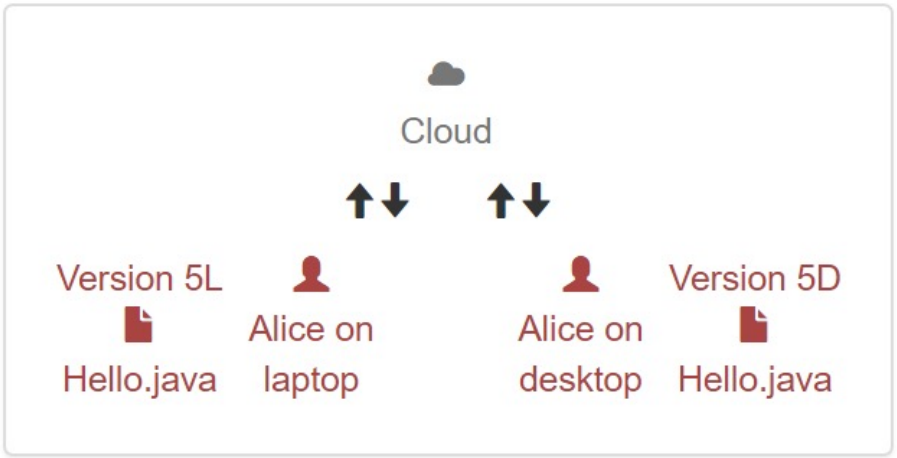


Version control systems you've already used

- 古老的版本控制方法：通过复制文件并修改文件名

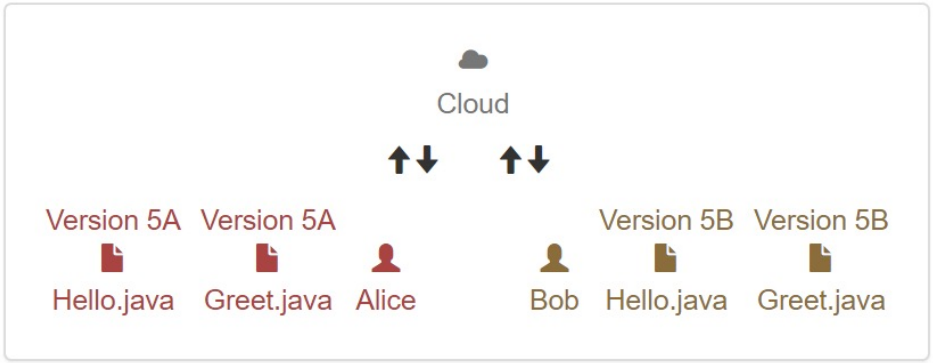
Why version control is required – for individuals

- *Reverting* to a past version 回滚到上一个版本
- *Comparing* two different versions 比较两个版本的差异
- *Pushing* full version history to another location 备份软件版本历史
- *Pulling* history back from that location 获取备份
- *Merging* versions that are offshoots of the same earlier version 合并

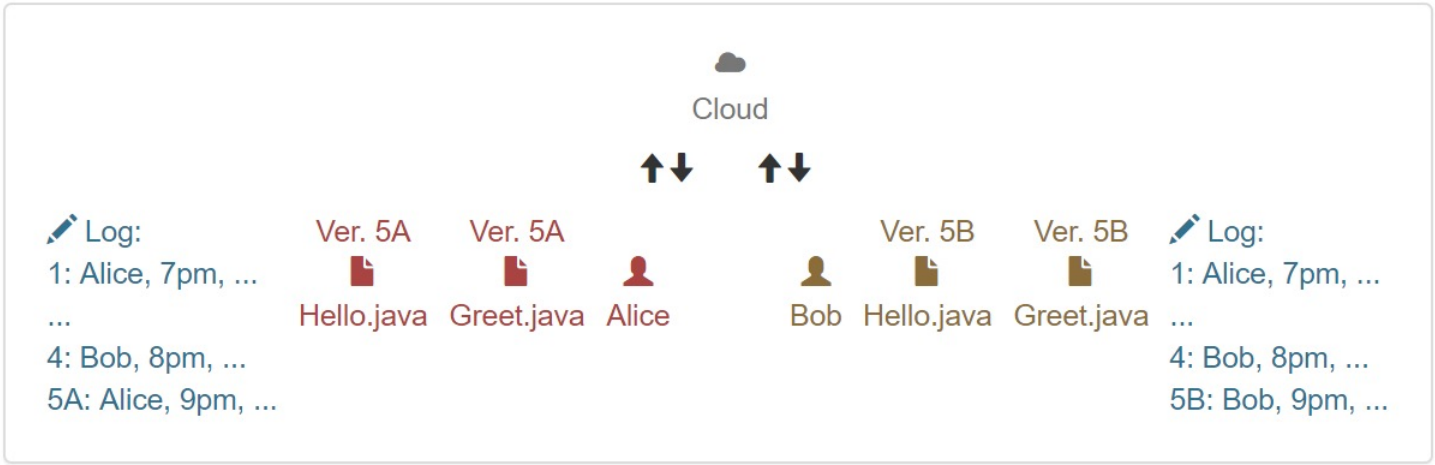


Why version control is required – for teamwork

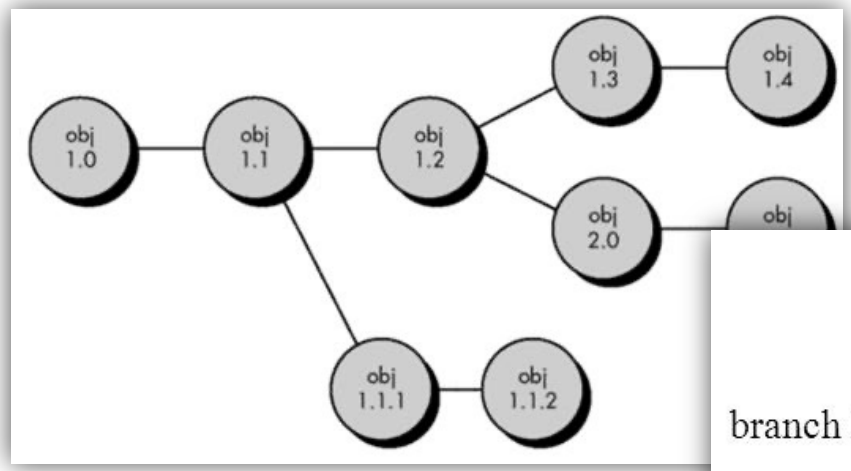
- **Communications and share/merge works among multiple developers** 在多个开发者之间共享和协作

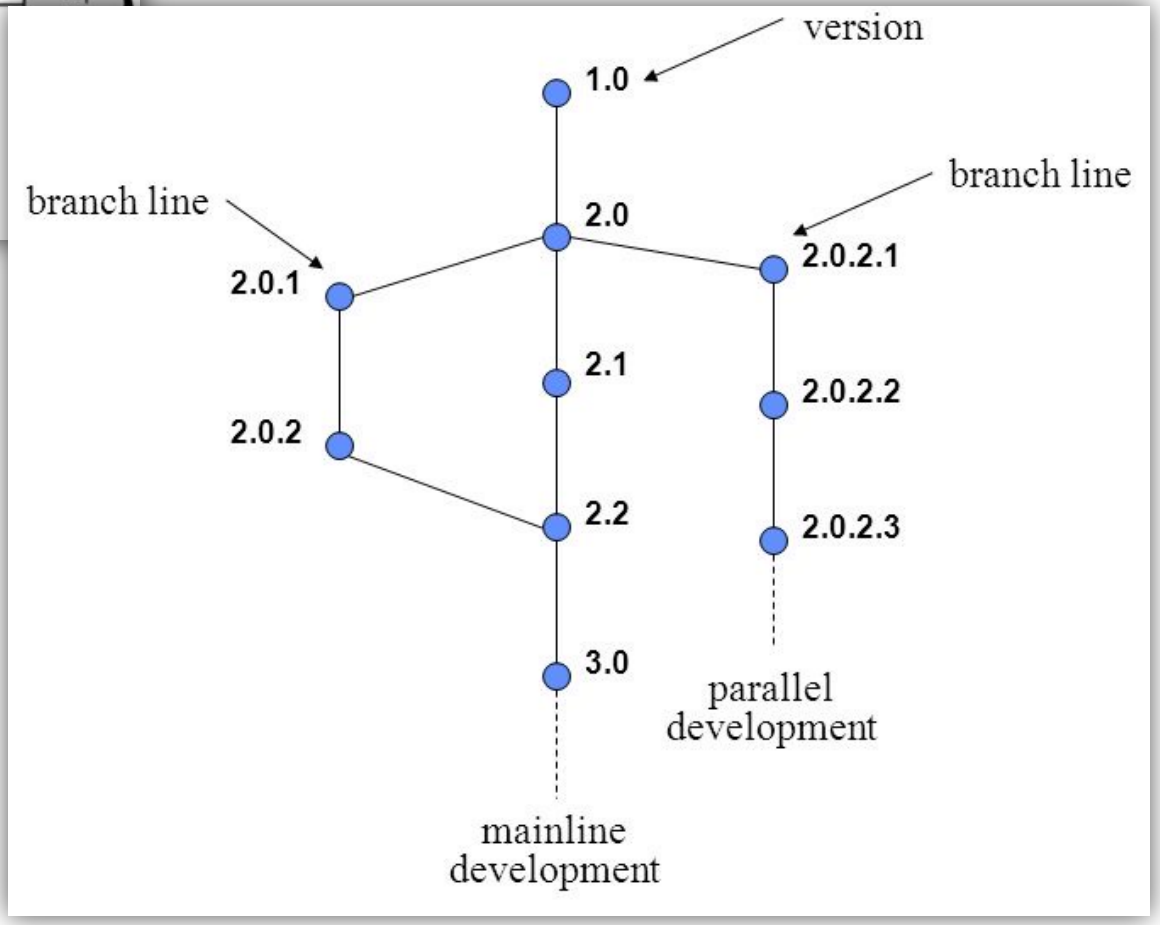


- **Logging individualized works of different developers for auditing** 记录每个开发者的动作，便于“审计”



History of an SCI





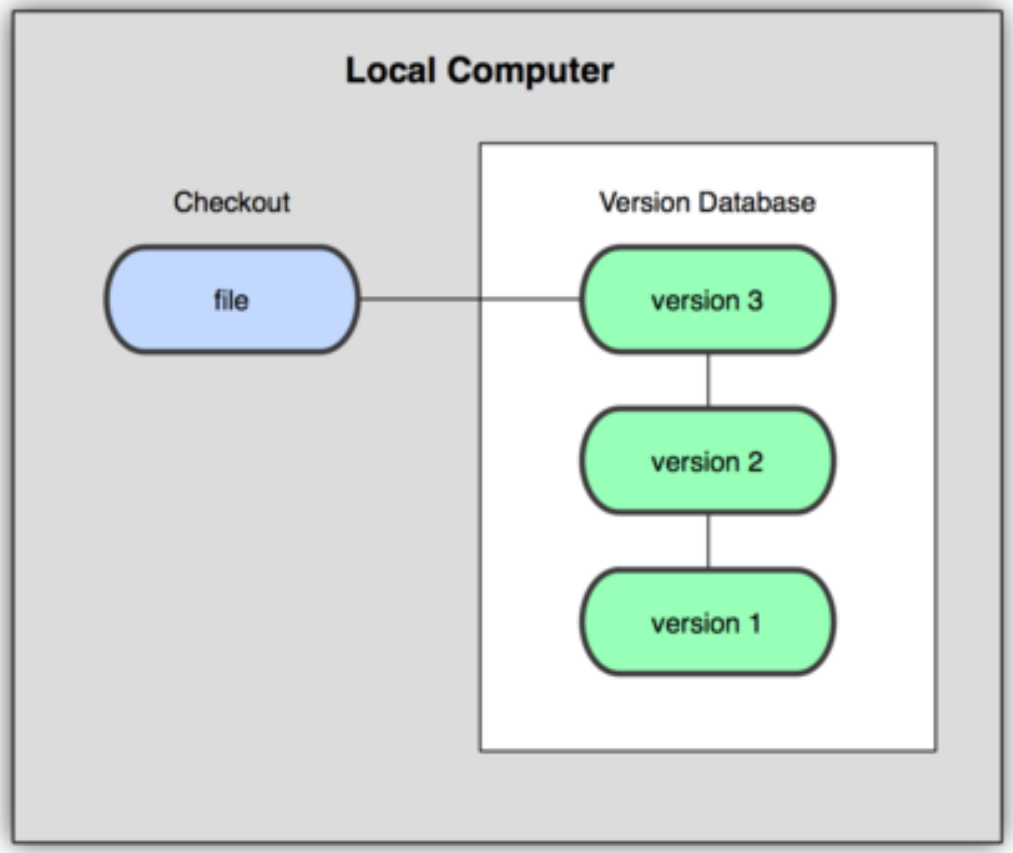
多个版本之间，形成线性或分支结构

Version control terminology

- **Repository**: a local or remote store of the versions in a project
  仓库：即于SCM中的CMDB
- **Working copy**: a local, editable copy of a project that we can work on
  工作拷贝：在开发者本地机器上的一份项目拷贝
- **File**: a single file in the project 文件：一个独立的配置项
- **Version** or **revision**: a record of the contents of the project at a point in time 版本：在某个特定时间点的所有文件的共同状态
- **Change** or **diff**: the difference between two versions 变化：即code churn，两个版本之间的差异
- **Head**: the current version HEAD：程序员正在其上工作的版本

Version Control System (VCS)

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

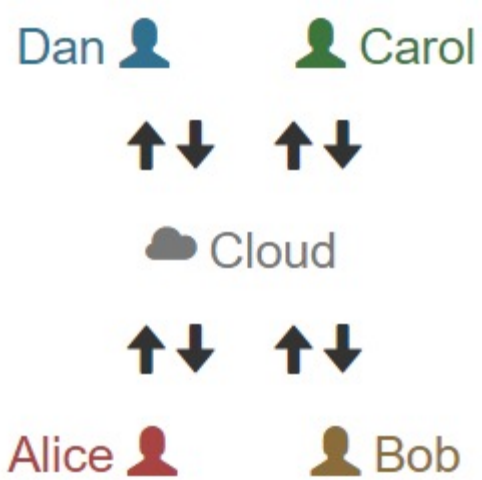


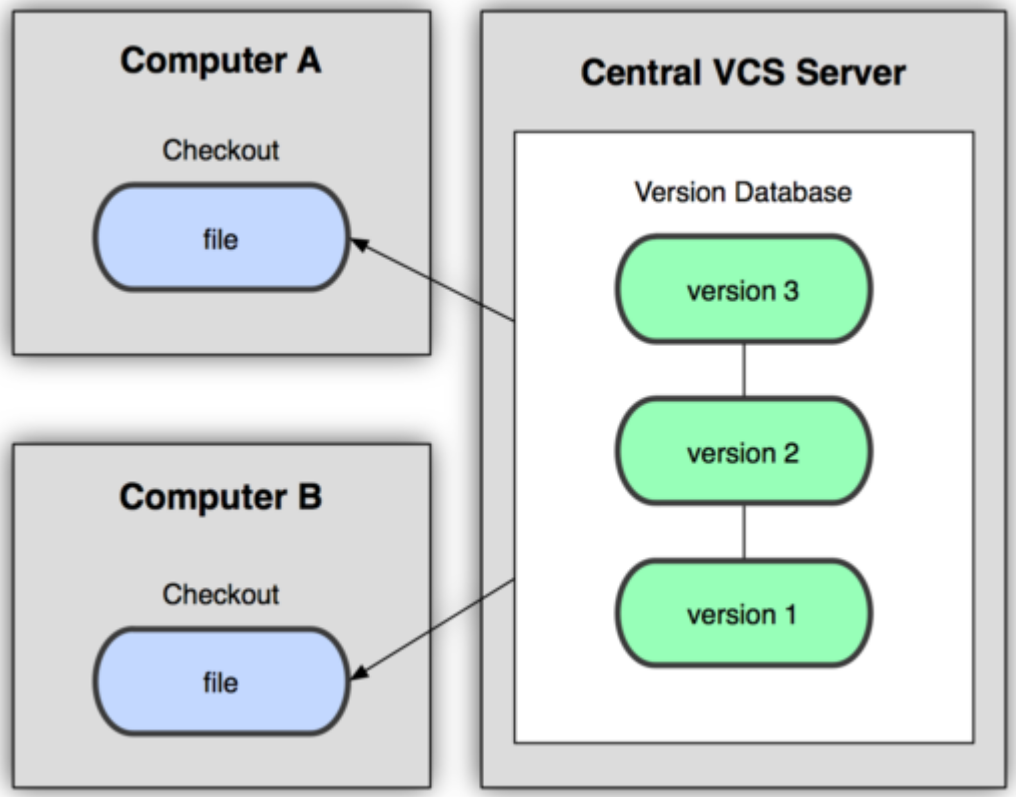
- **本地版本控制系统：**
  仓库存储于开发者本地机器
  无法共享和协作

Version Control System (VCS)

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

集中式版本控制系统：仓库存储于独立的服务器，支持多开发者之间的协作

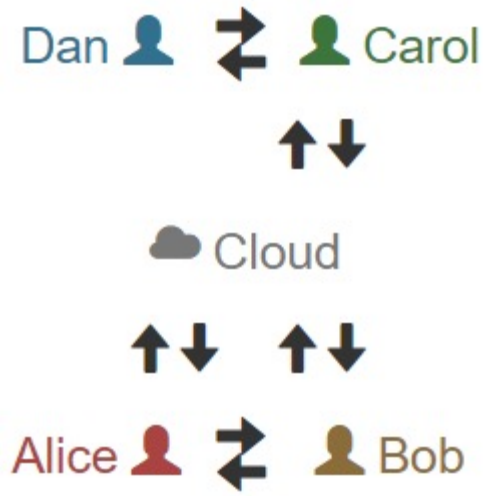


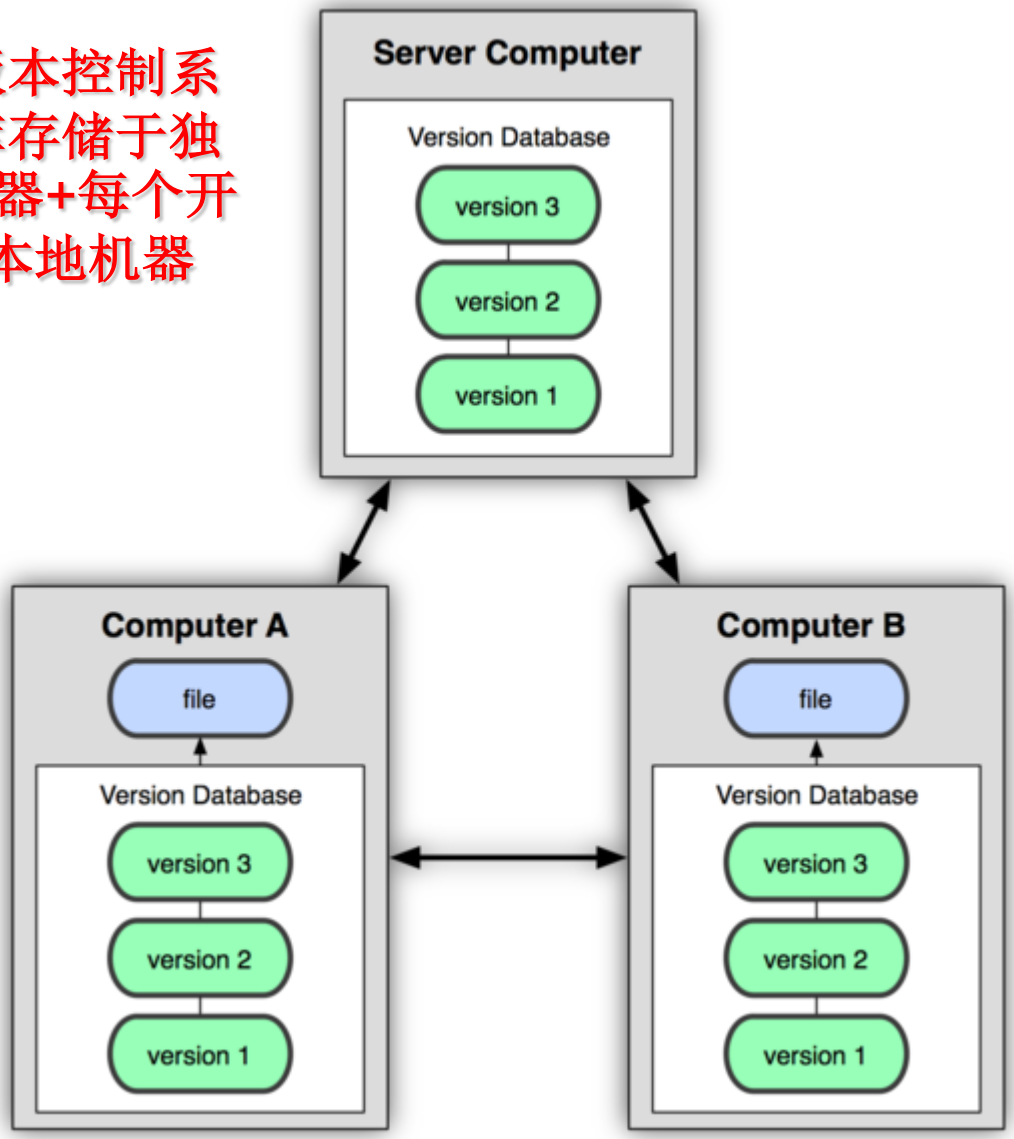


Version Control System (VCS)

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

分布式版本控制系统：仓库存储于独立的服务器+每个开发者的本地机器





5 Git as an example of SCM tool

What is Git?

Linux

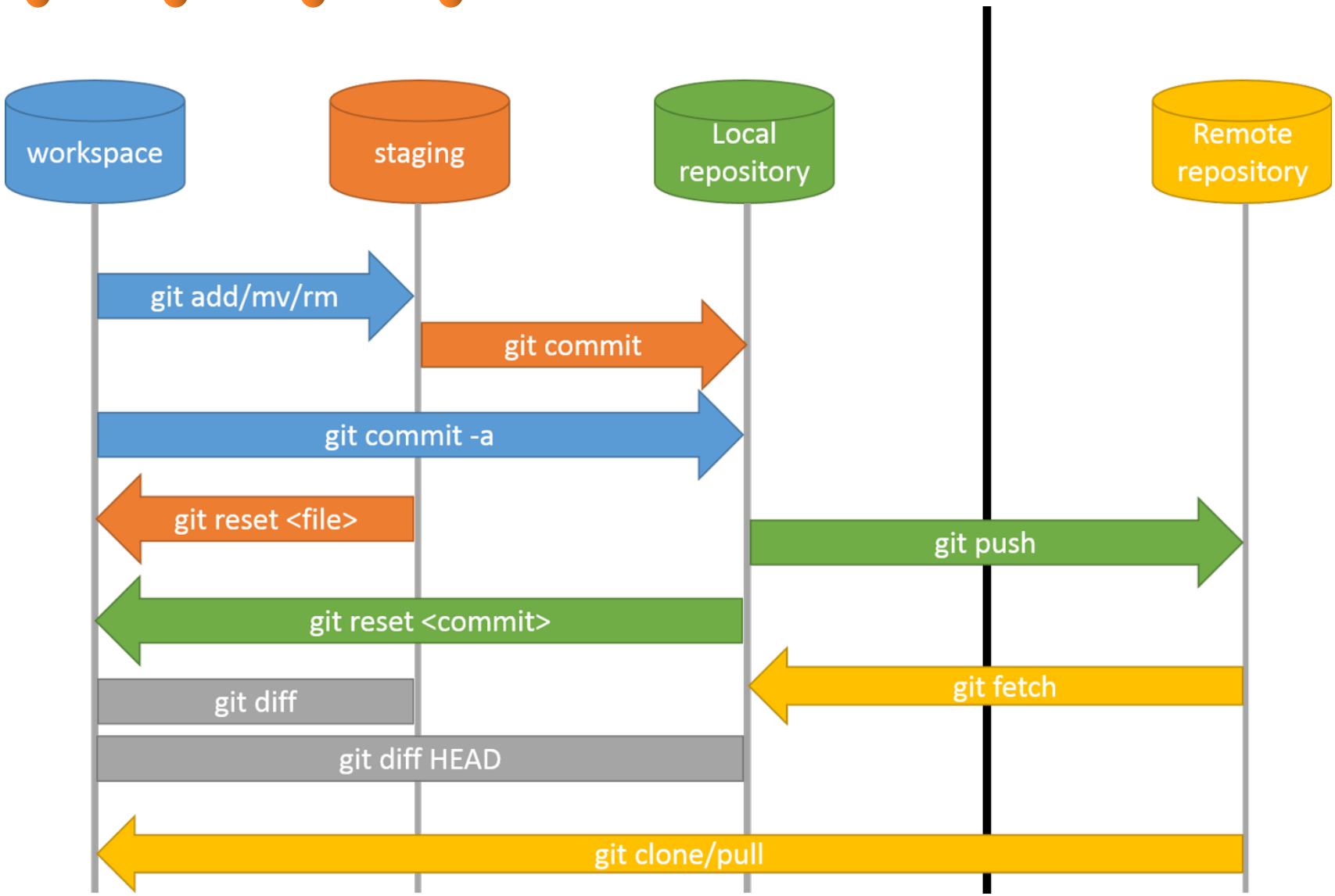
- **Initial release: 2005**
- **Initial Author: Linus Torvalds**
- **For development of the Linux kernel.**

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

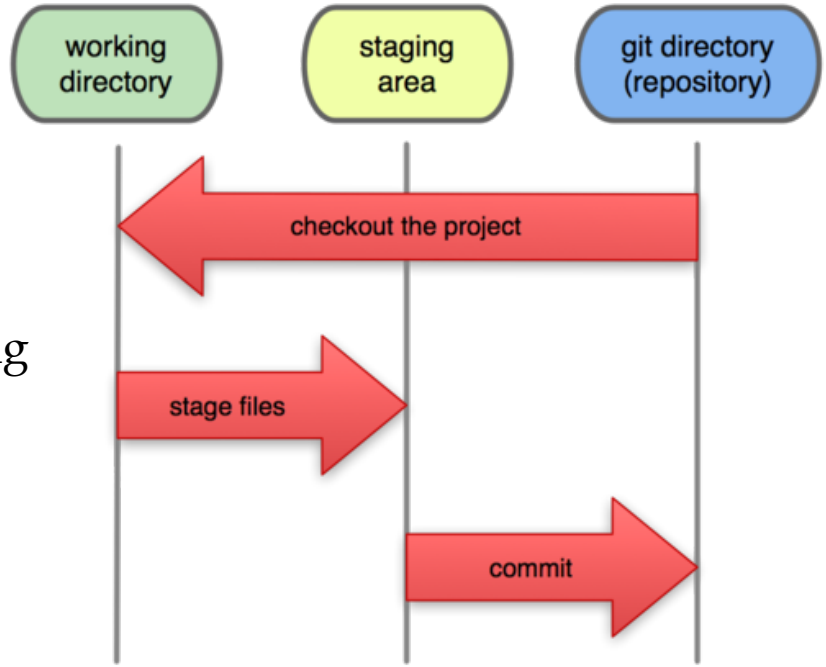
Legend: Old version | Older version, still supported | **Latest version** | Latest preview version

Managing changes in software evolution process



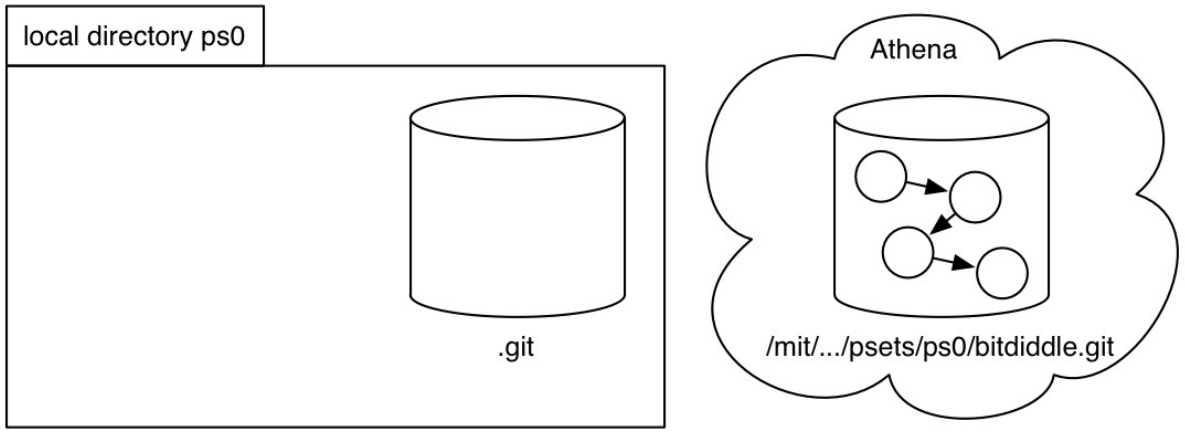
Git repository

- **A Git repository has three parts:**
  - .git directory (a repository storing all version control data) 本地的CMDB
  - Working directory (local file system) 工作目录：本地文件系统
  - Staging area (in memory) 暂存区：隔离工作目录和Git仓库
- **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) 已修改
  - Staged (the file is modified and has been added into the staging area) 已暂存
  - Committed (the file keeps same in working directory and git directory) 已提交



Object graph in Git

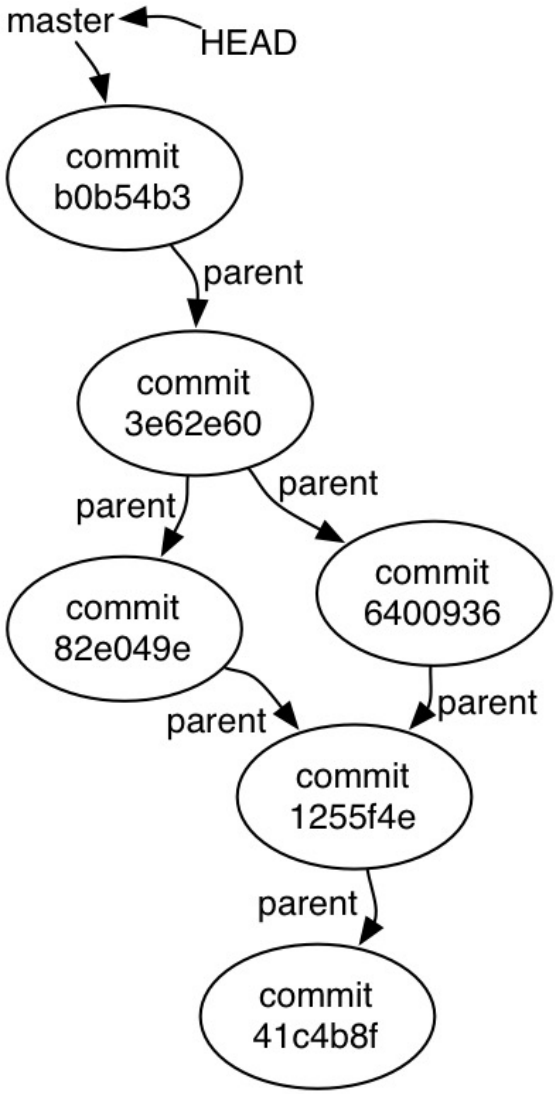
- **All of the operations we do with Git — clone, add, commit, push, log, merge, … — are operations on a graph data structure that stores all of the versions of files in the project, and all the log entries describing those changes.**
- **The Git object graph is stored in the .git directory of the repository.**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



What an Object Graph looks like?

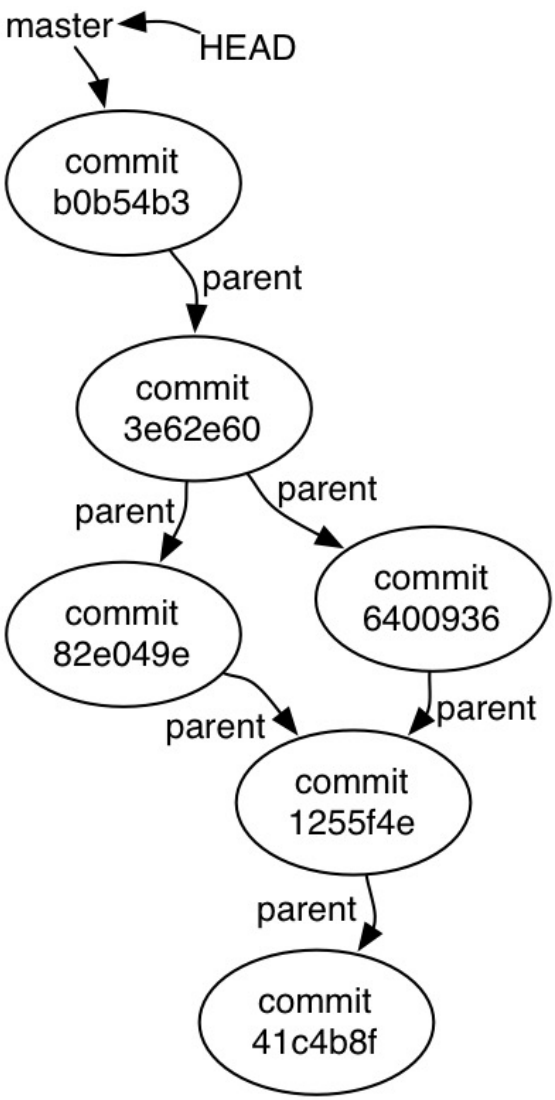
- **Object graph, being the history of a Git project, is a directed acyclic graph (DAG).**

```
* 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
```



- **Object Graph：版本之间的演化关系图，一条边A->B表征了“在版本B的基础上作出变化，形成了版本A”**

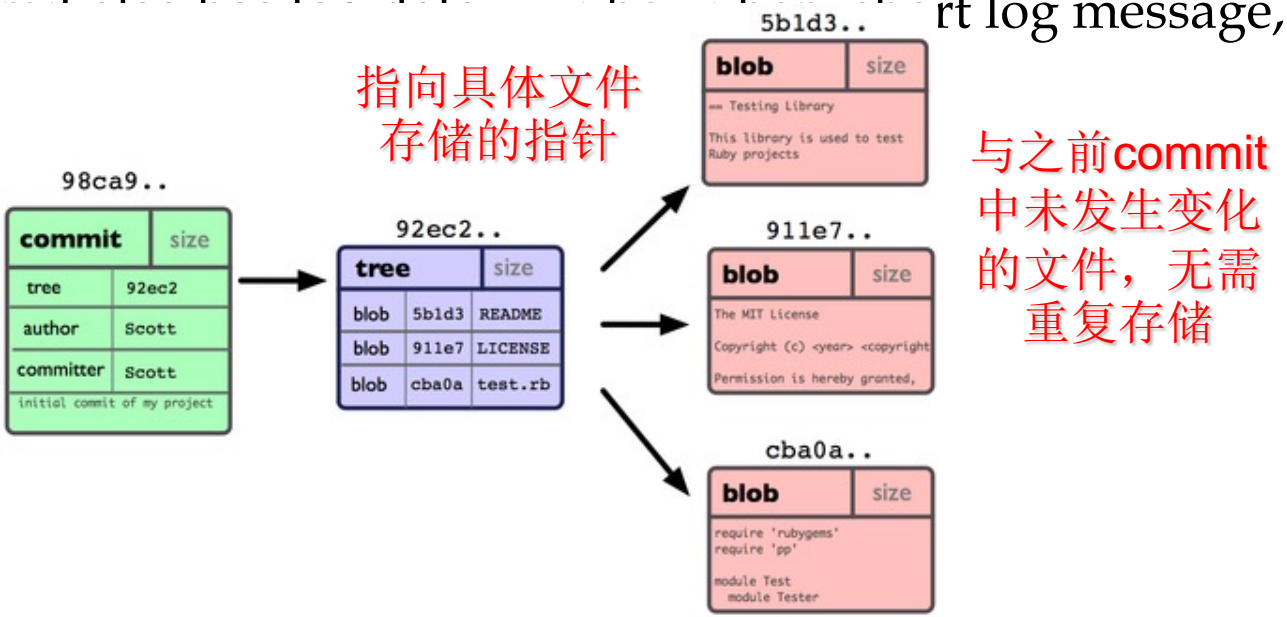
Commits: nodes in Object Graph



- **Each node in the history graph is a commit a.k.a. version a.k.a. revision of the project: a complete snapshot of all the files at that point in time.**
  - Except for the initial commit, each commit has a pointer to a parent commit. 每个commit指向一个父亲
  - Some commits have the same parent: they are versions that diverged from a common previous version. 多个commit指向同一个父亲：分支
  - Some commits have two parents: they are versions that tie divergent histories back together. 一个commit指向两个父亲：合并
- **A `branch` is just a name that points to a commit.**
- **`HEAD` points to the current commit.**
  - We need to remember which branch we're working on. So `HEAD` points to the current branch, which points to the current commit.

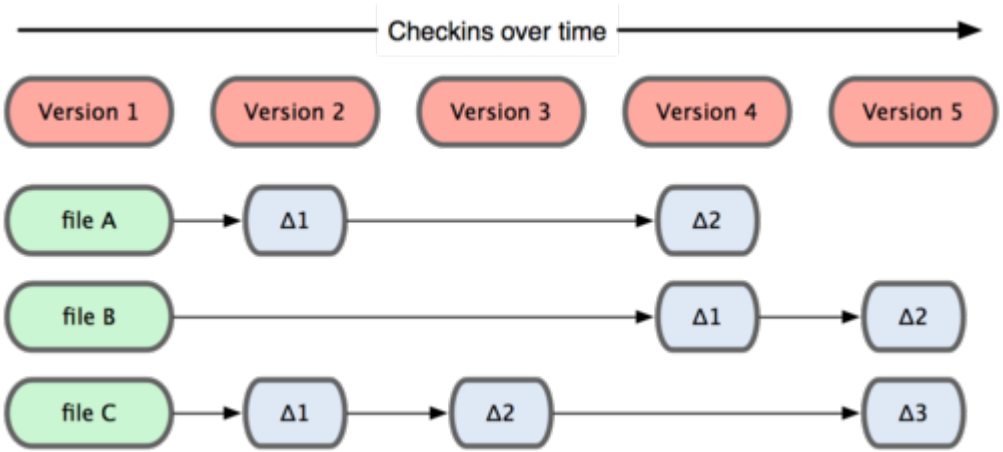
Commits: nodes in Object Graph

- Git represents a commit 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, Git object graph stores each version of an individual file *once*, and allows multiple commits to *share* that one copy.
  - Each com[illegible] rt log message, etc.



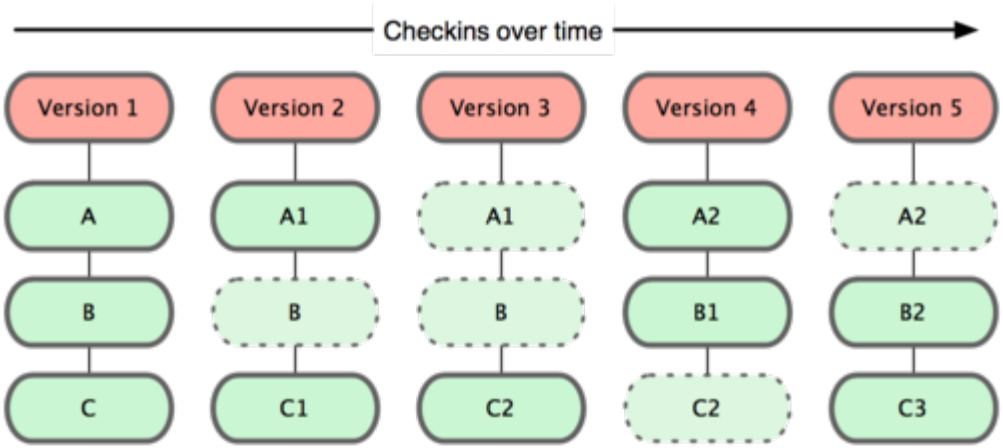
Managing changes in Git

- **Traditional VCS:**



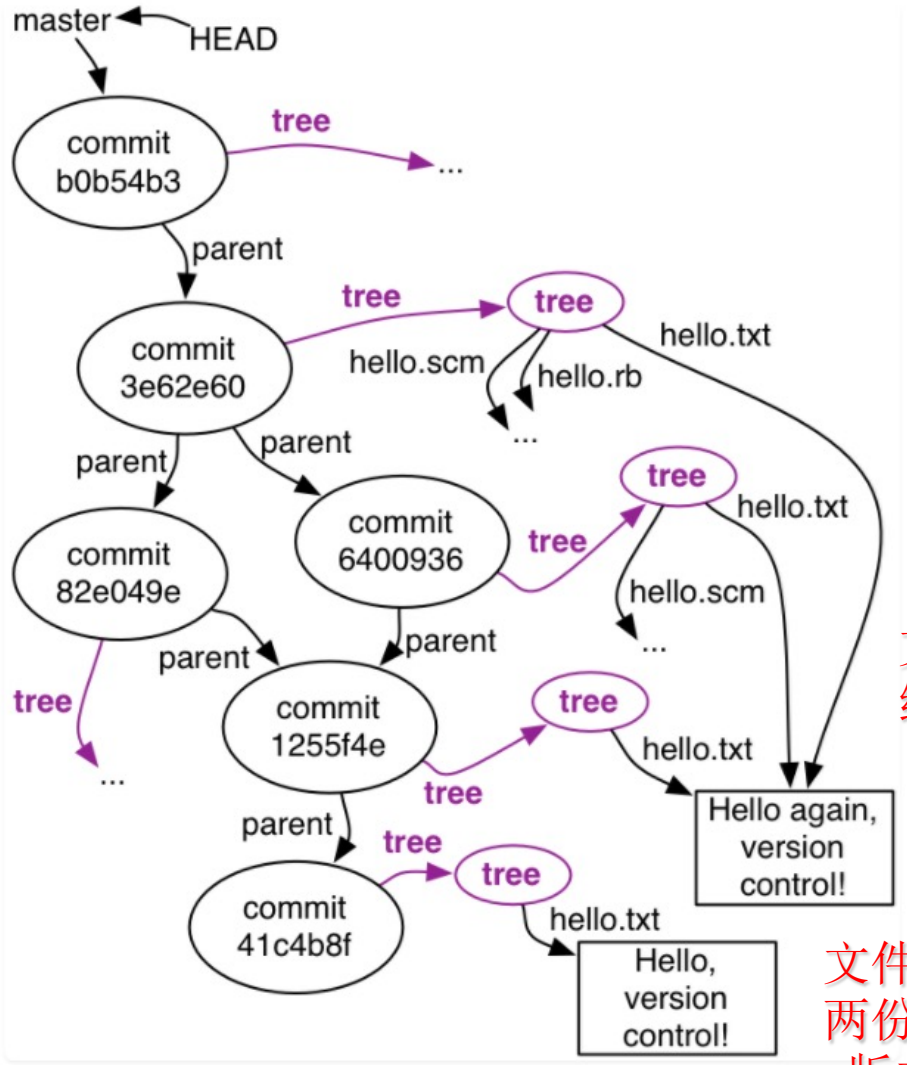
传统VCS存储版本之间的变化（行）

- **In Git:**



Git存储发生变化的文件（而非代码行），不变化的文件不重复存储

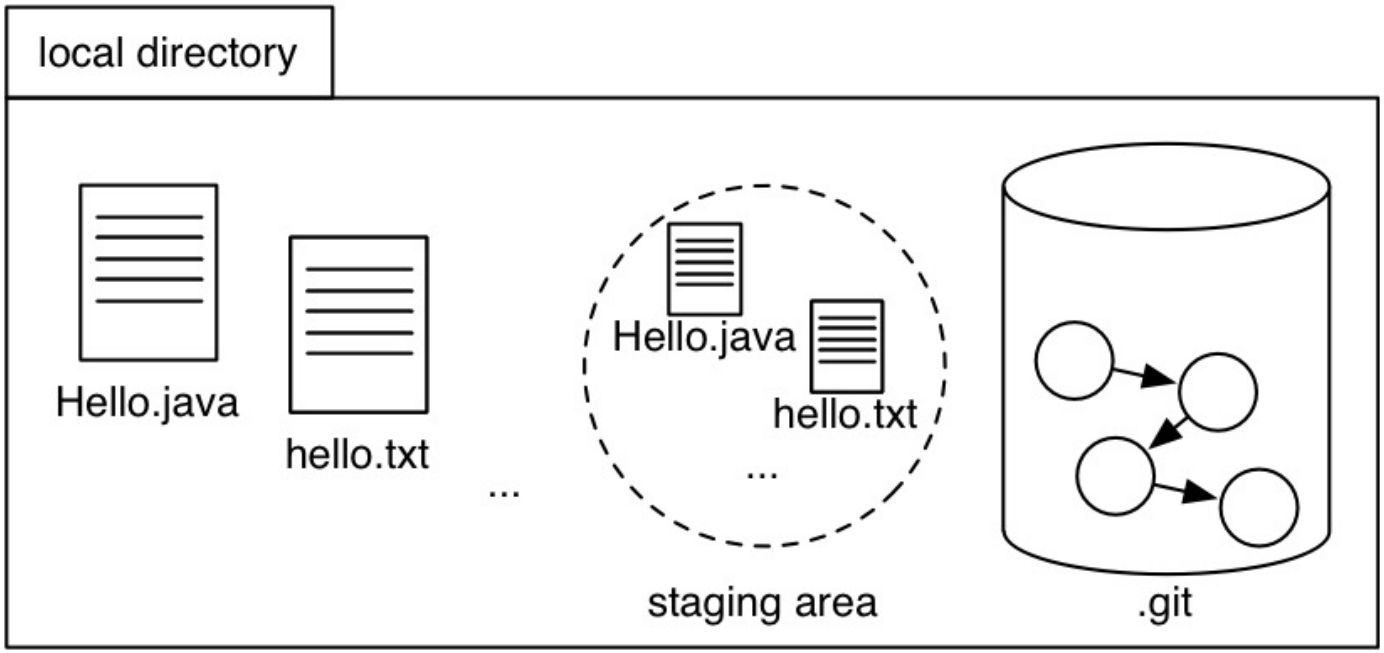
Commits: nodes in Object Graph



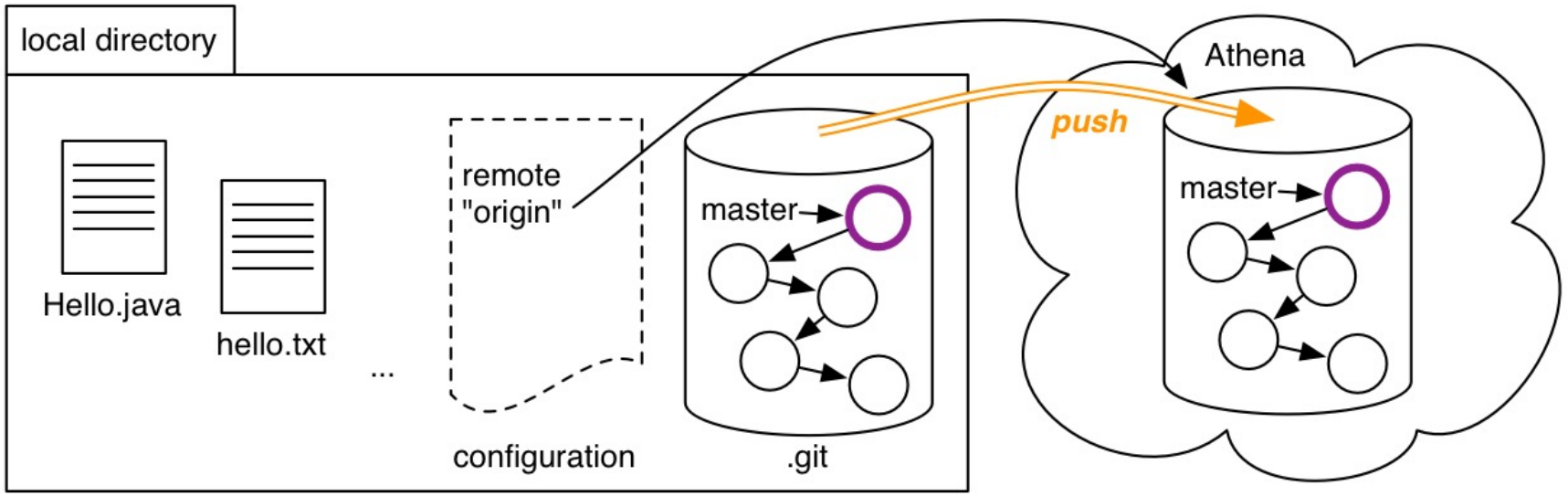
文件未发生变化，则后续多个版本始终指向同一个文件

文件发生变化了，存储两份不同的文件，两个版本指向不同的文件

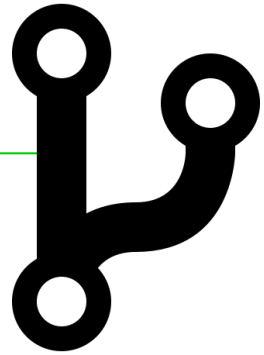
Add to the object graph with `git commit`



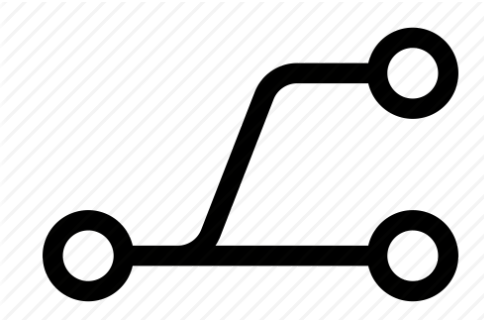
Send & receive object graphs with git push & git pull

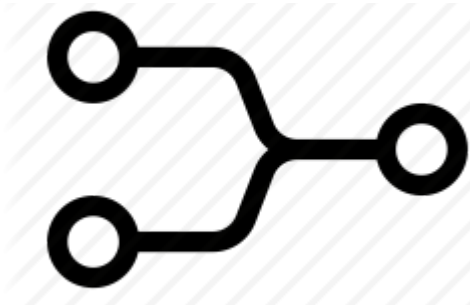


Git supports Branch and Merge 分支/合并

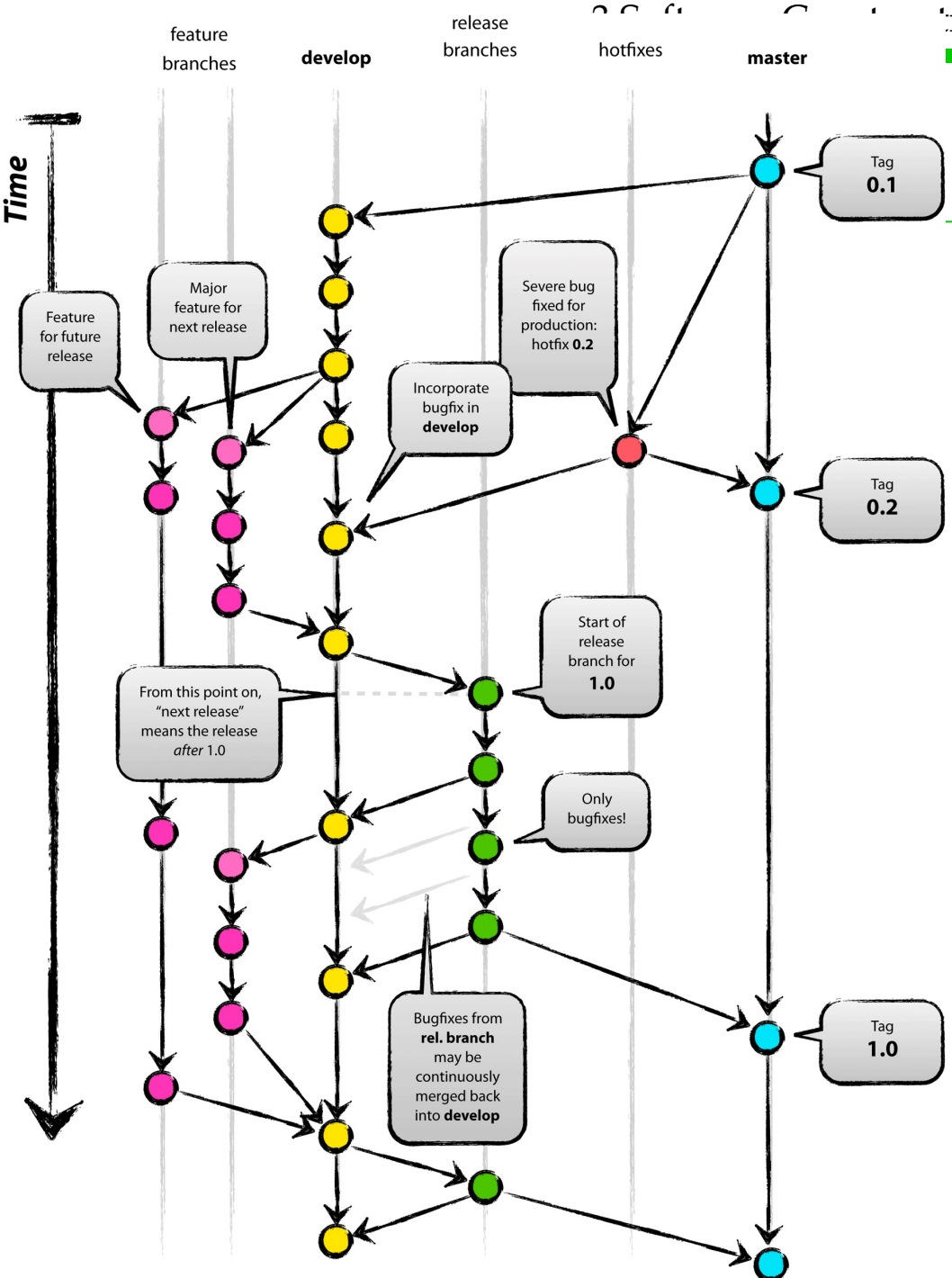


- **A branch is the duplication of an object under revision control so that modifications can happen in parallel along both branches.**
- **Merging two branches together.**

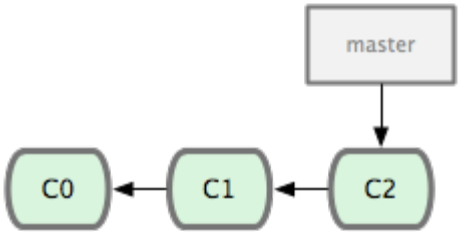


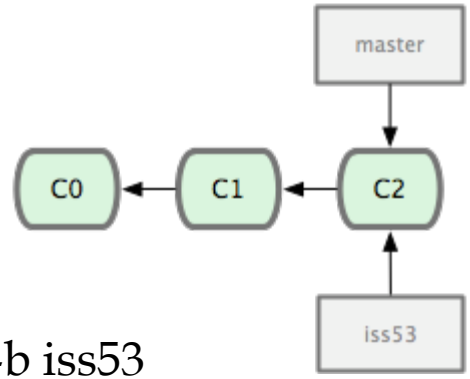


Branching

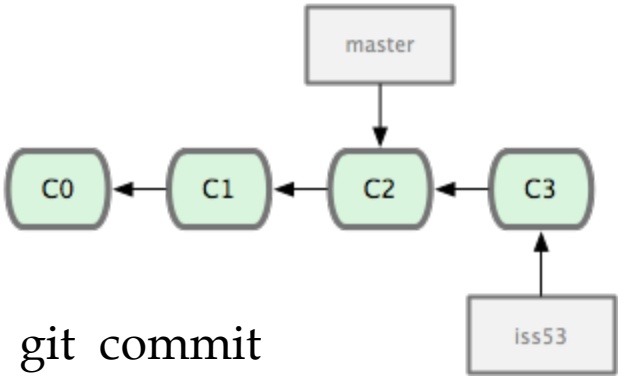

feature branches
develop
release branches
hotfixes
master
Time
Tag 0.1
Major feature for next release
Feature for future release
Severe bug fixed for production: hotfix 0.2
Incorporate bugfix in develop
Tag 0.2
Start of release branch for 1.0
From this point on, "next release" means the release after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

Creating and merging branches in Git

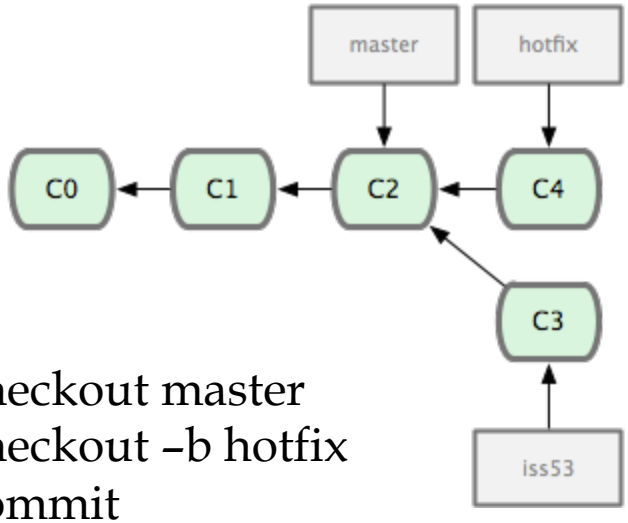




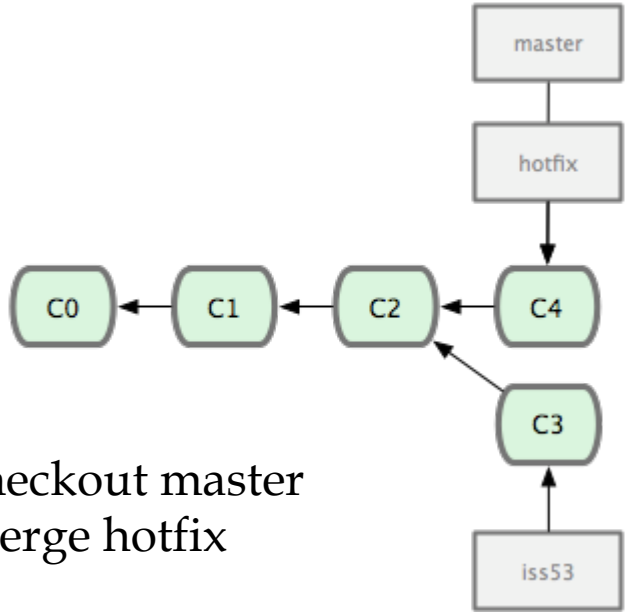
git checkout –b iss53



git commit

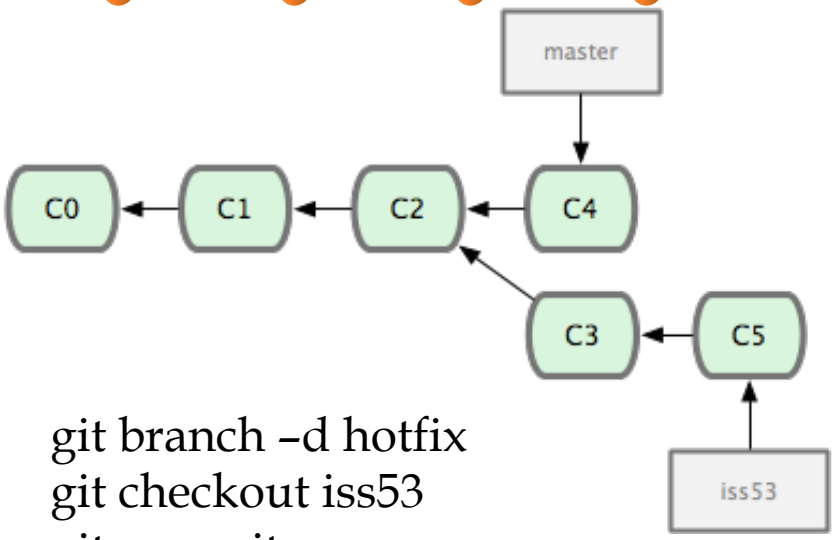


git checkout master
git checkout –b hotfix
git commit

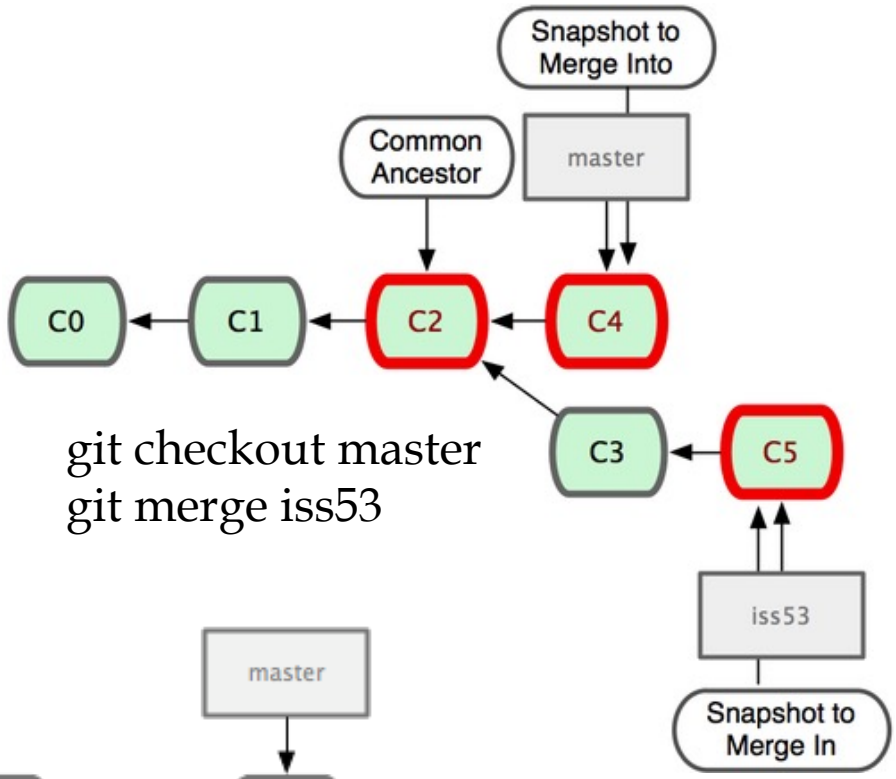


git checkout master
git merge hotfix

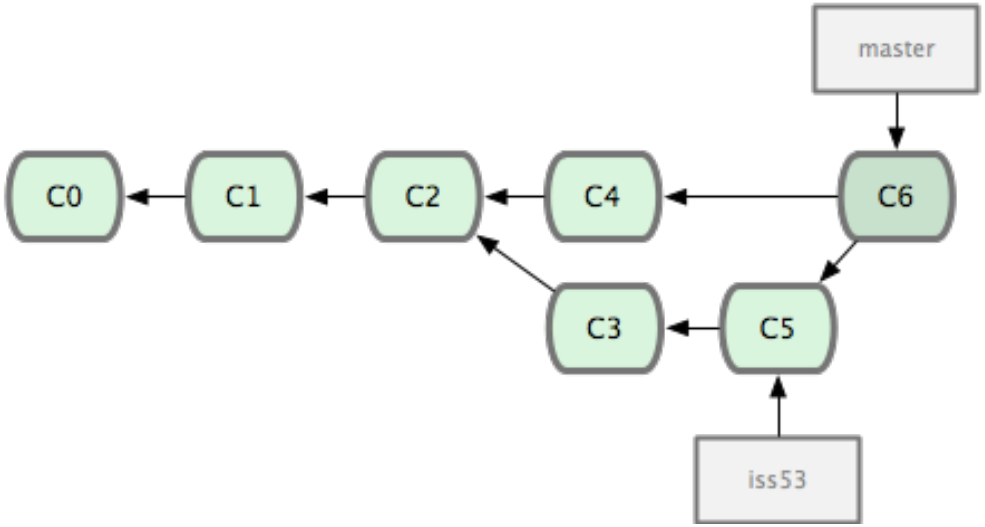
Creating and merging branches in Git



git branch –d hotfix
git checkout iss53
git commit

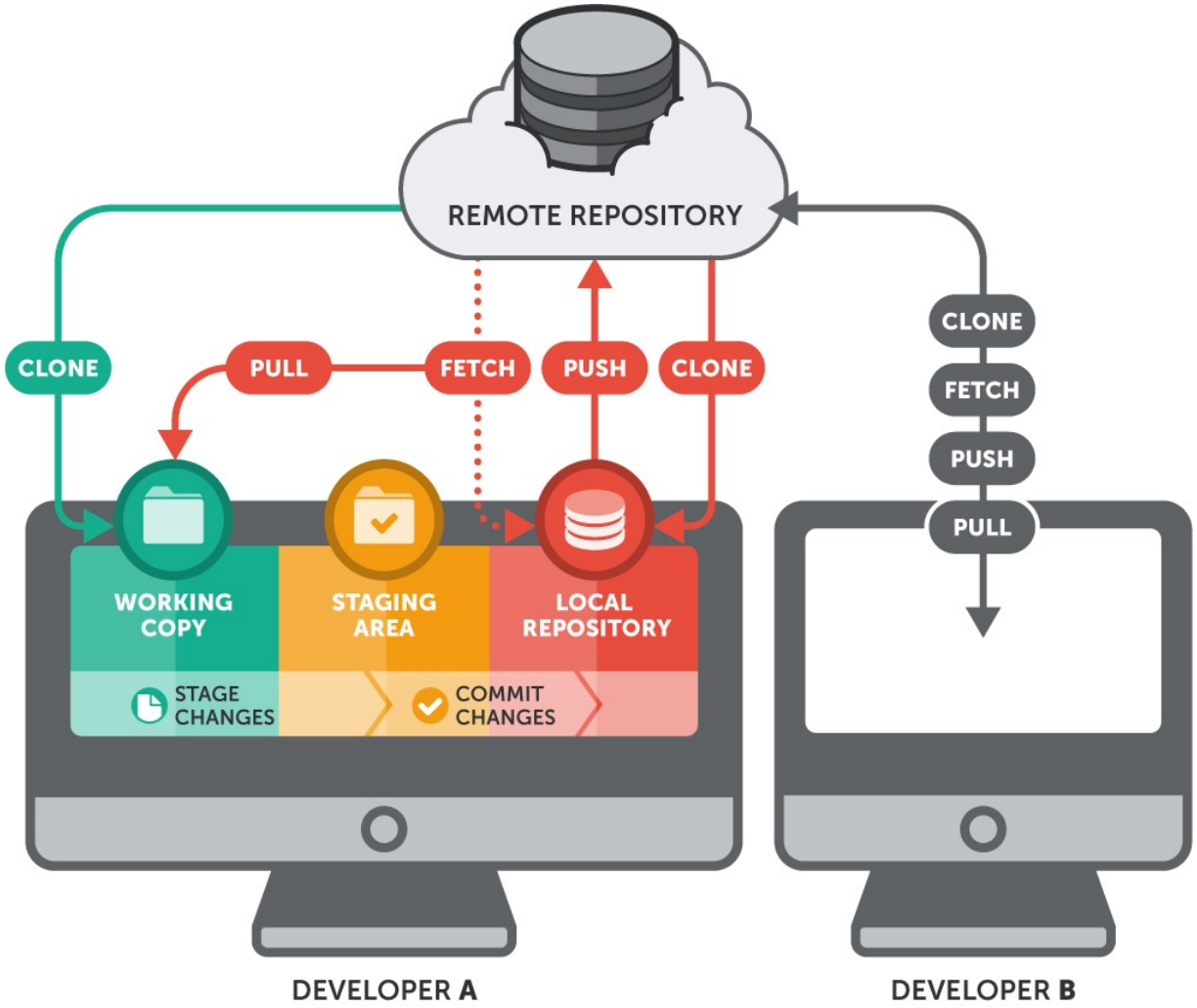


git checkout master
git merge iss53

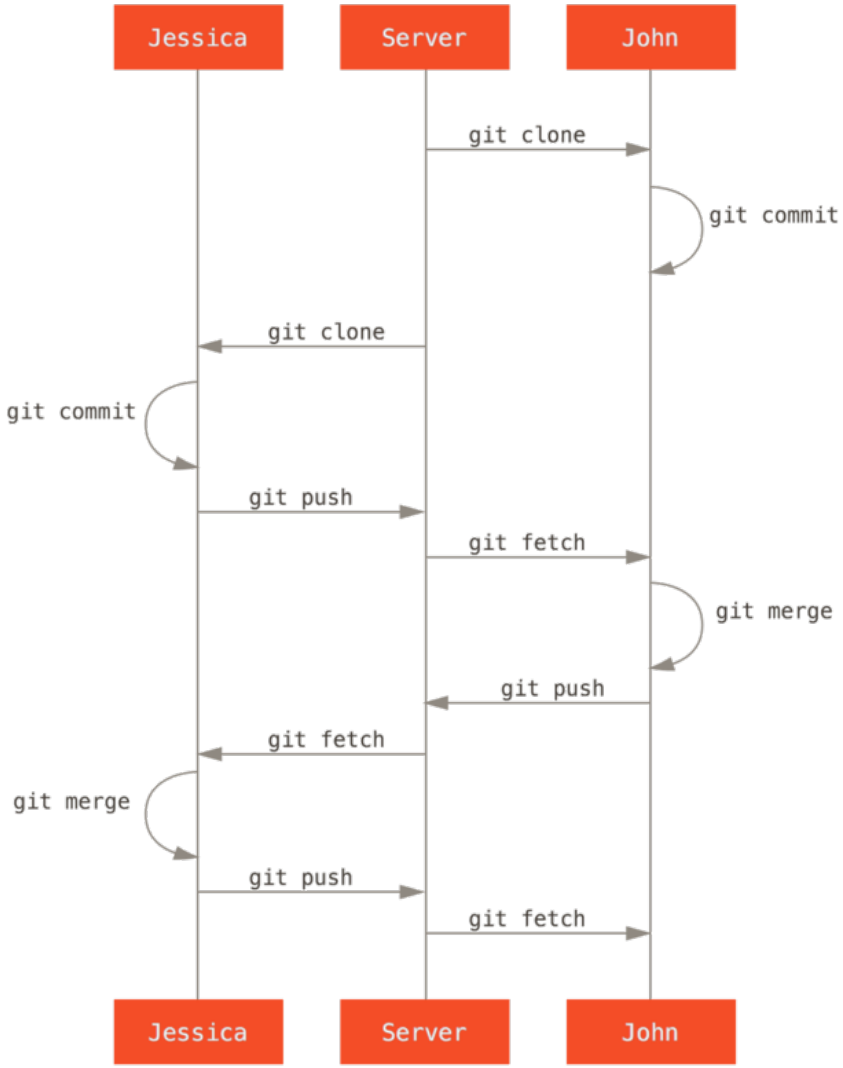


Git supports collaboration

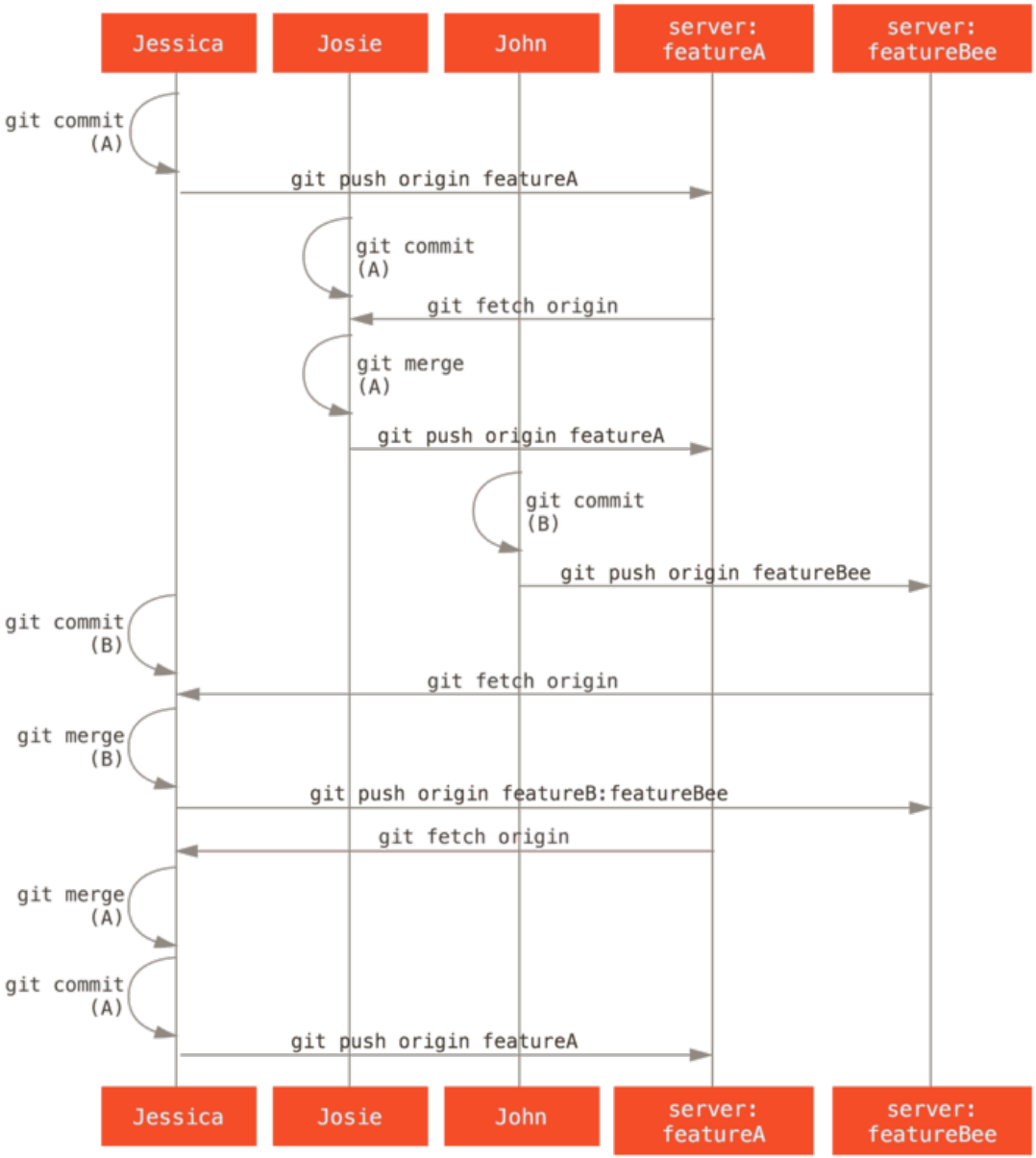
- **Local repository and Remote Repository**



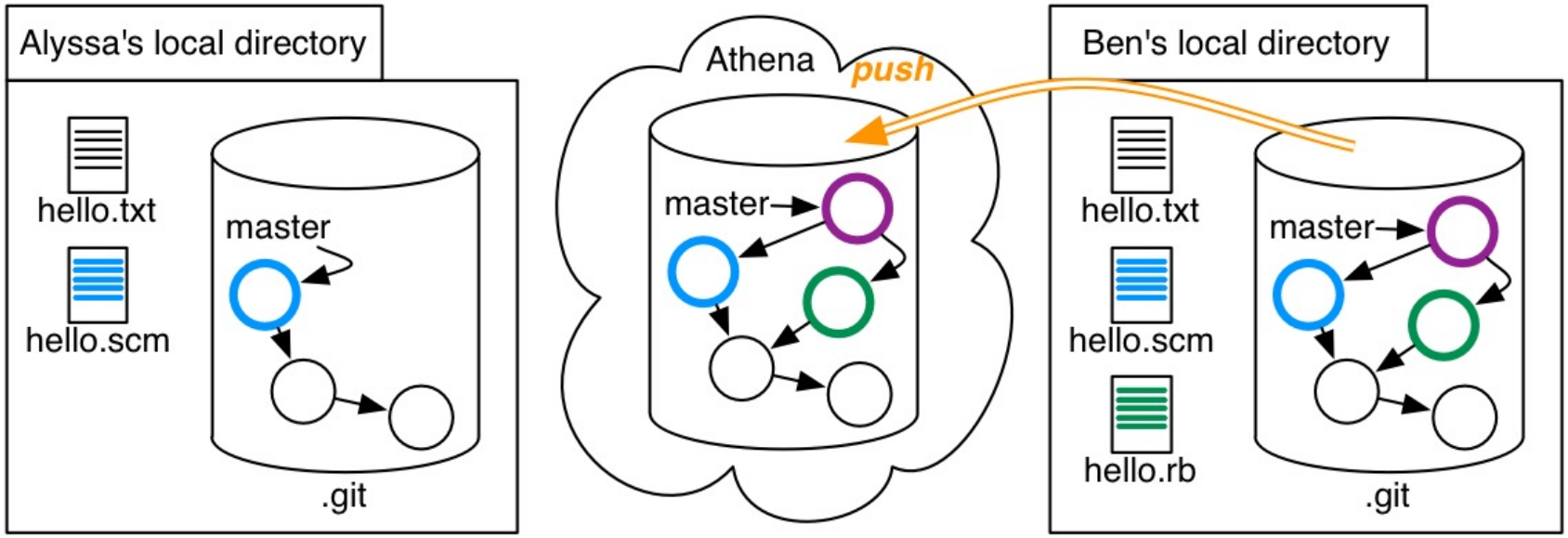
Git supports collaboration



Git supports collaboration

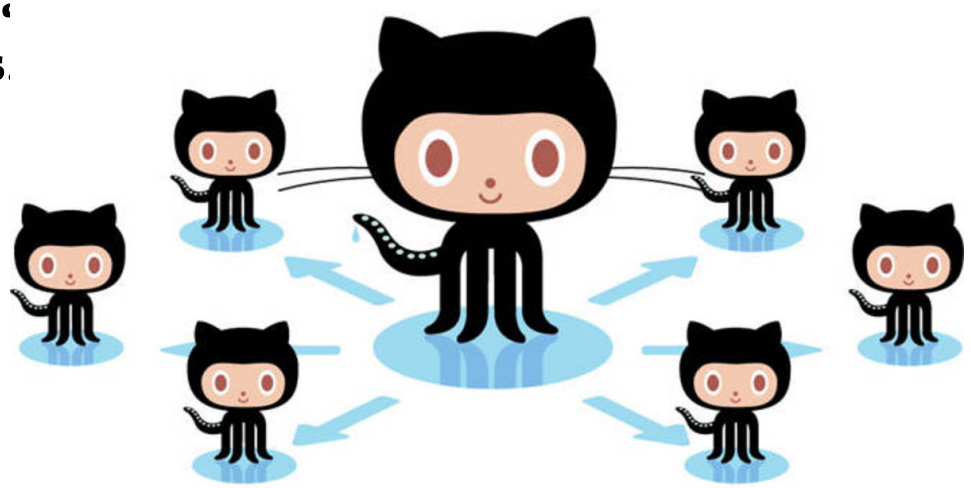


Git supports collaboration



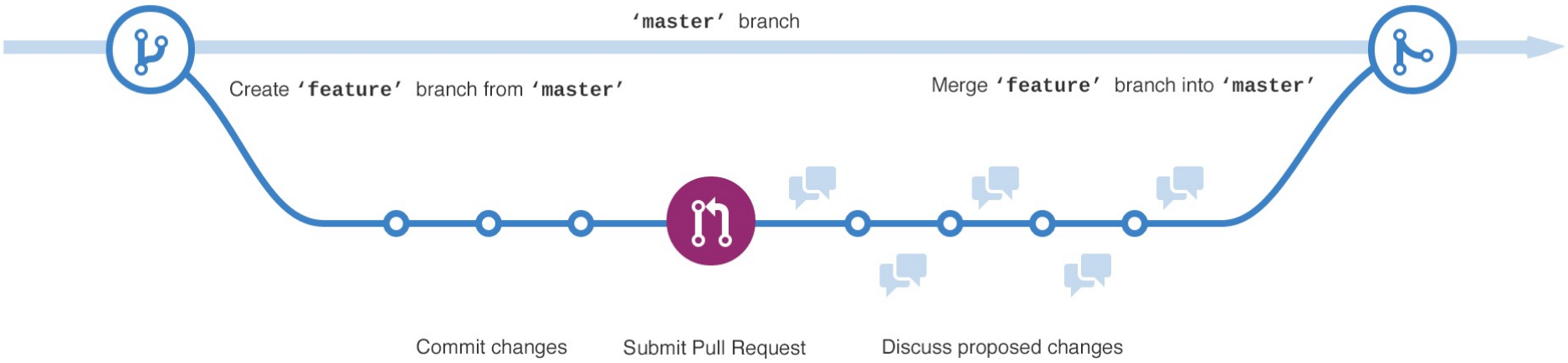
GitHub is how people build software

- **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 2019, it ha[illegible] million repositories.**

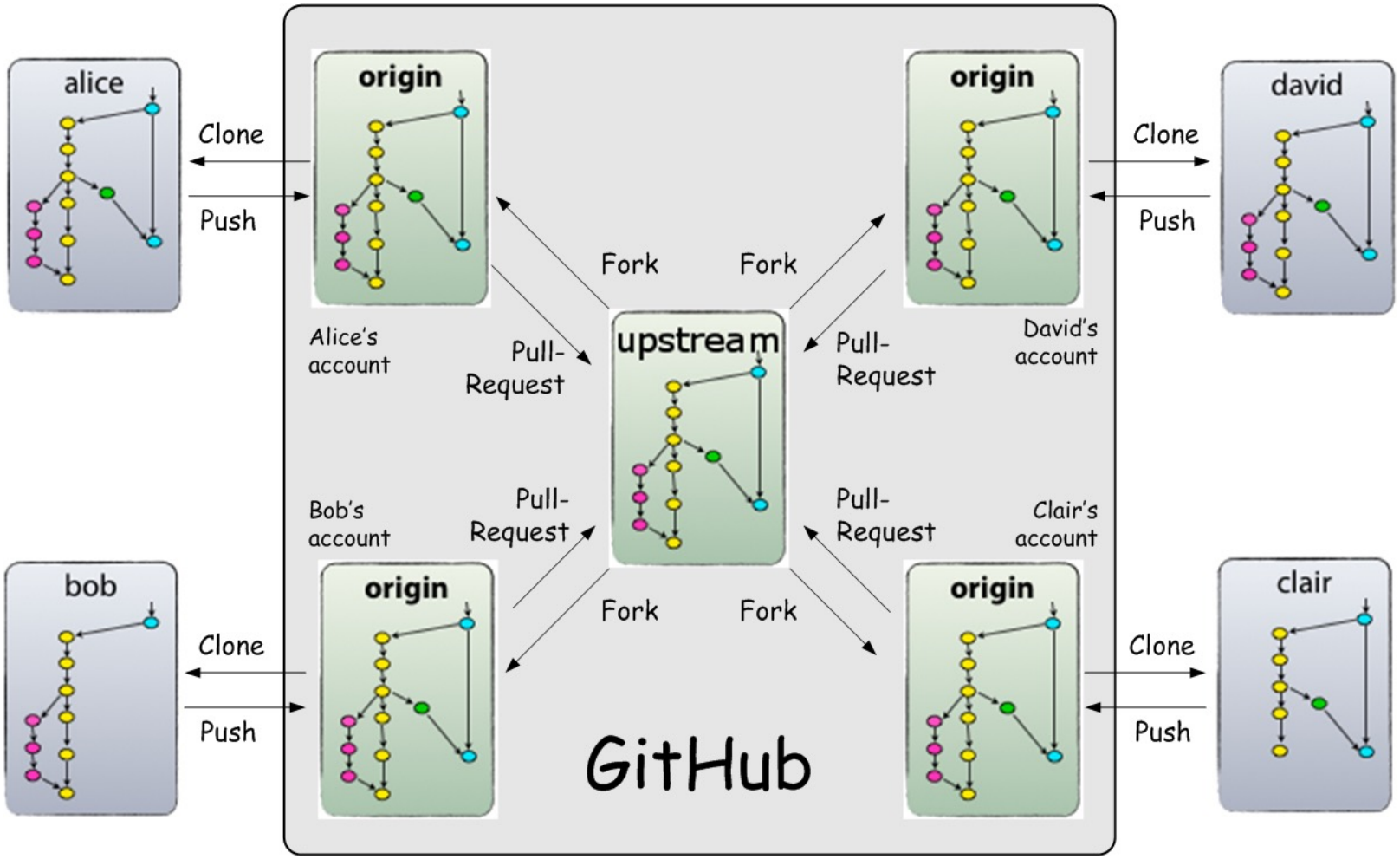


GitHub working process

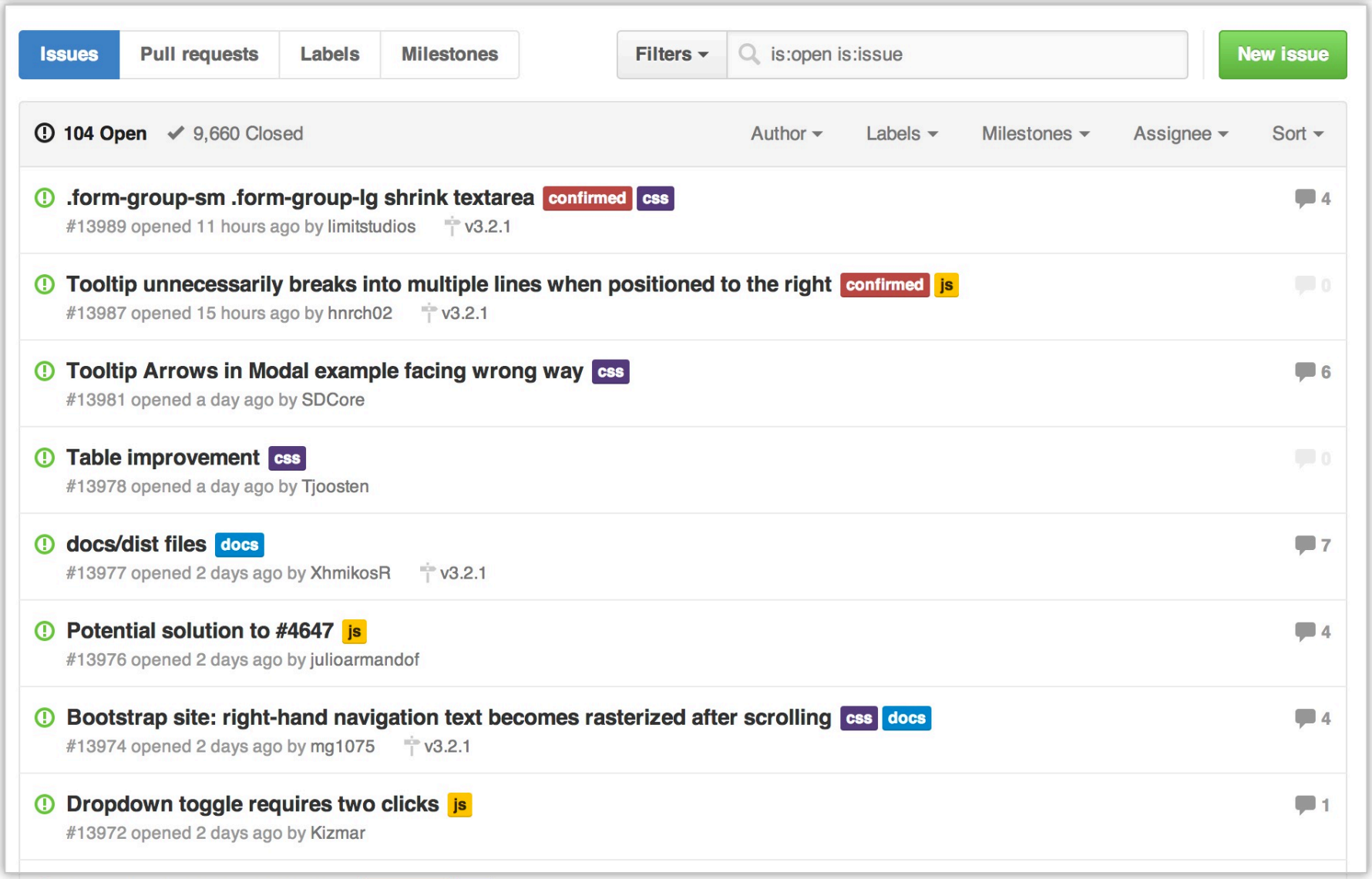
- **Basic process: commit, branch and merge**
- **Collaboration process: fork and pull request**



GitHub working process

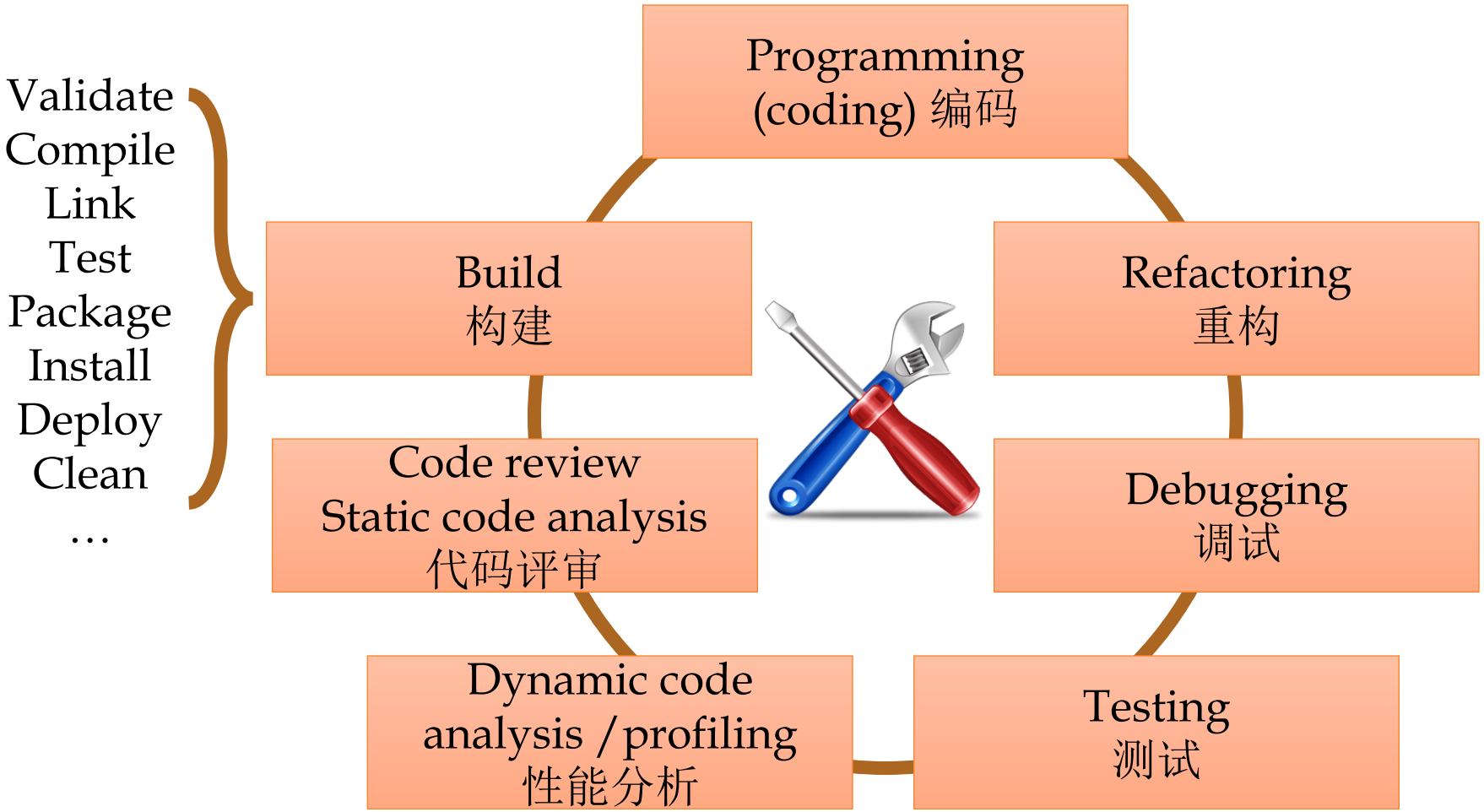


Issue Tracking and Pull Request



6 General process of software construction

General process of software construction



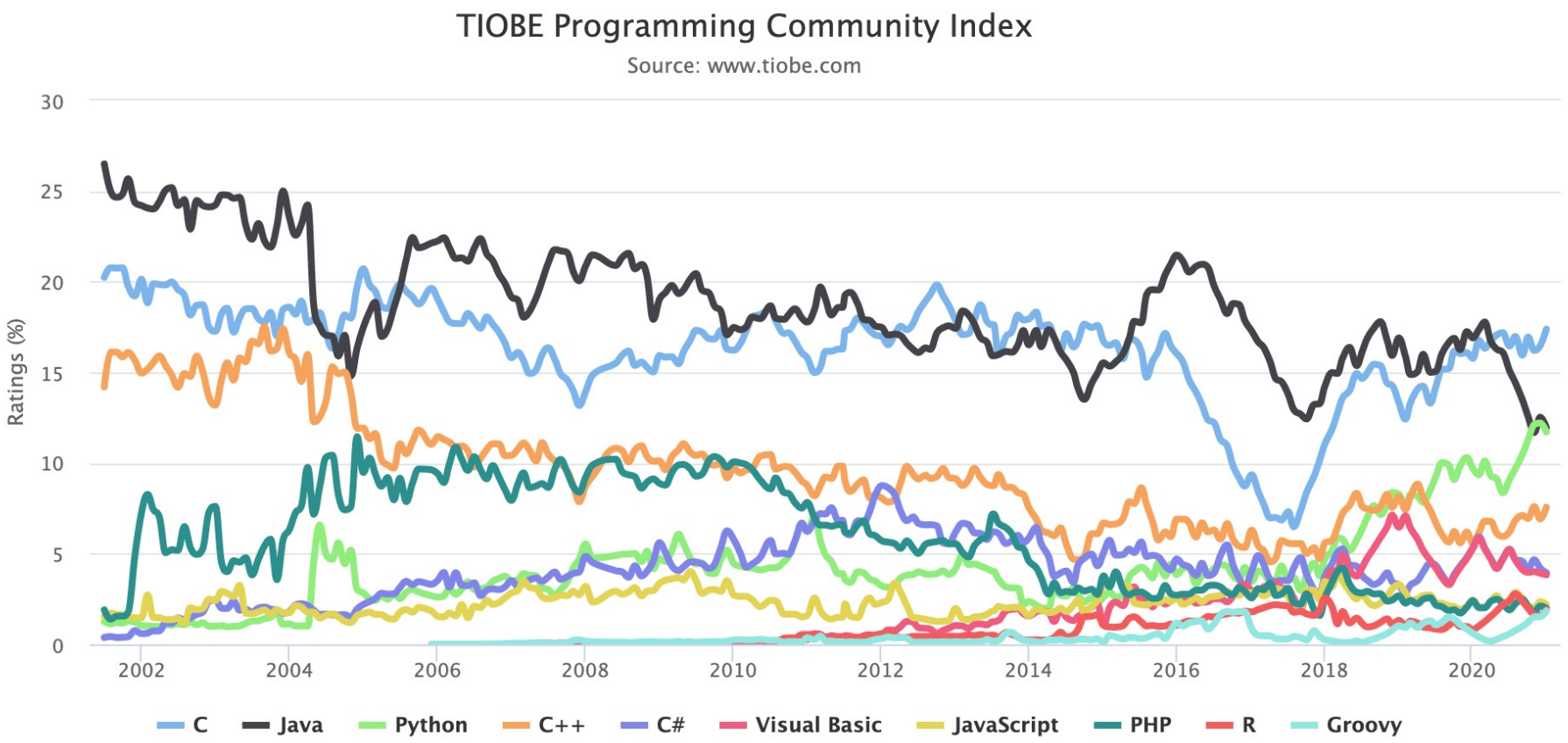
(1) Programming

Construction languages

- **Purpose of the languages** 从用途上划分
  - Programming languages (e.g., C, C++, Java, Python) 编程语言
  - Modeling languages (e.g., UML) 建模语言
  - Configuration languages (e.g., XML) 配置语言
  - Build languages (e.g., XML) 构建语言

- **Forms of the languages** 从形态上划分
  - Linguistic-based 基于语言学的构造语言
  - Mathematics-based (formal) 基于数学的形式化构造语言
  - Graphics-based (visual) 基于图形的可视化构造语言

(1) Programming Languages 编程语言

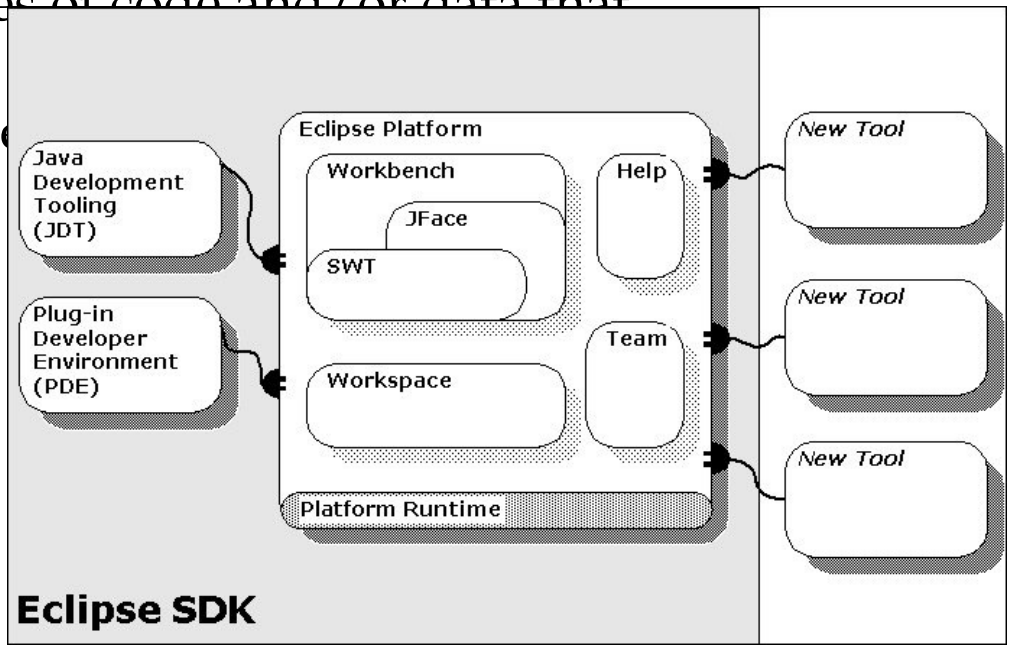


Programming tools

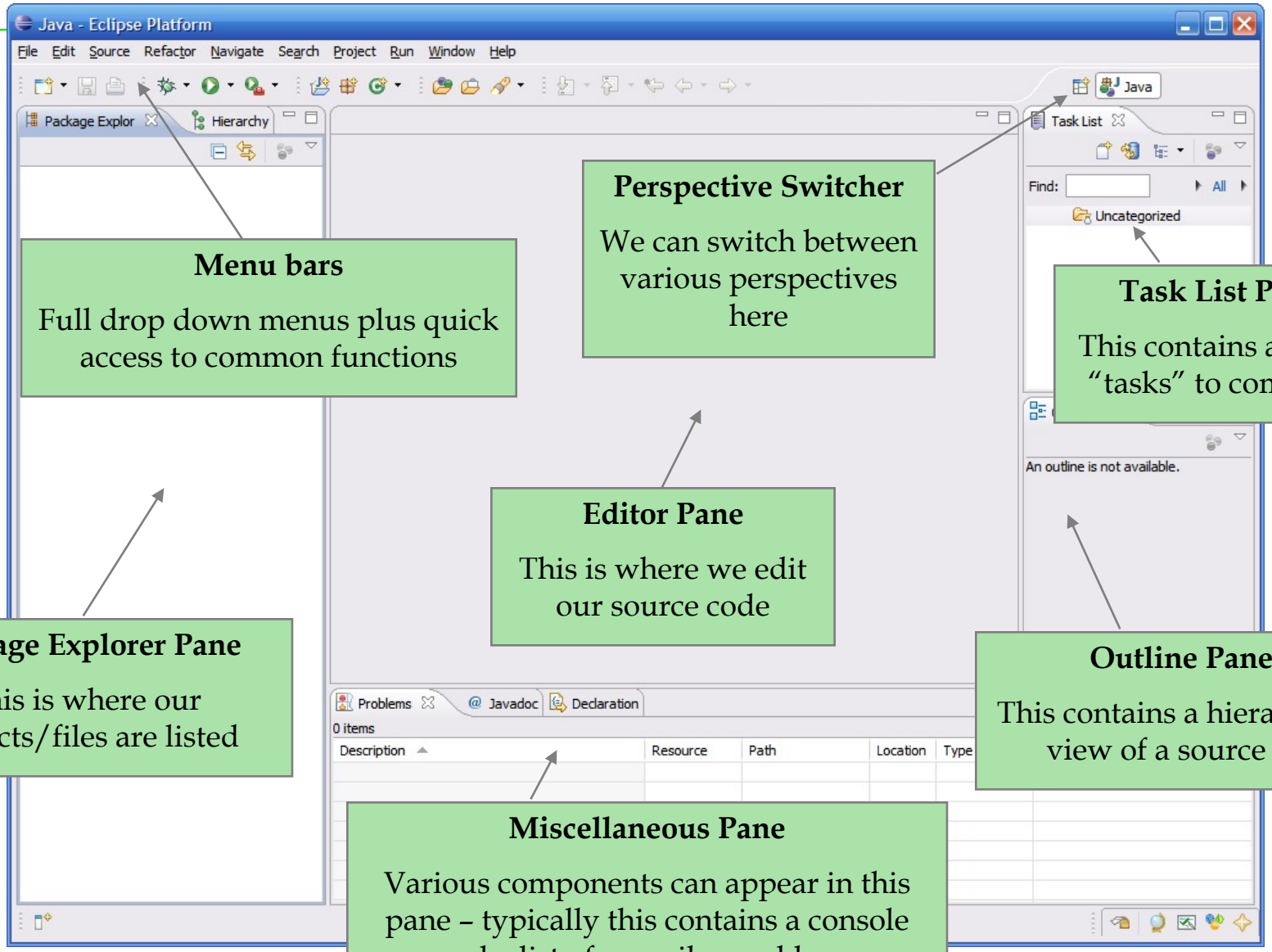
- **Integrated development environment (IDE):** comprehensive facilities to programmers for software development. 集成开发环境
- **An IDE normally consists of:**
  - Source code editor with intelligent code completion, code refactoring tool 源代码编辑器、智能代码补全工具、代码重构工具
  - File management tool 文件管理
  - Library management tool 库管理
  - Class browser, object browser, class hierarchy diagram 软件逻辑实体可视化
  - Graphical User Interface (GUI) builder 图形化用户界面构造器
  - Compiler, interpreter 编译器、解释器
  - Build automation tools 自动化build工具
  - Version control system 版本控制系统
  - Extensible by more external third-party tools 外部的第三方工具

Eclipse as an IDE example

- **Eclipse IDE: an open source IDE for Java, but not limited to, C/C++, PHP, Python, etc, started as a proprietary IBM product (Visual age for Smalltalk/Java)**
  - It contains a base workspace with tools for coding, building, running and debugging applications, and an extensible plug-in system for customizing the environment.
  - Plug-ins are structured bundles of code and/or data that contribute functionality to the Functionality can be contribute in the form of code libraries, platform extensions, or even documentation.
  - Plug-ins can define extension points, well-defined places where other plug-ins can add functionality.



Eclipse IDE Components



Menu bars

Full drop down menus plus quick access to common functions

Perspective Switcher

We can switch between various perspectives here

Task List Pane

This contains a list of "tasks" to complete

Editor Pane

This is where we edit our source code

Package Explorer Pane

This is where our projects/files are listed

Outline Pane

This contains a hierarchical view of a source file

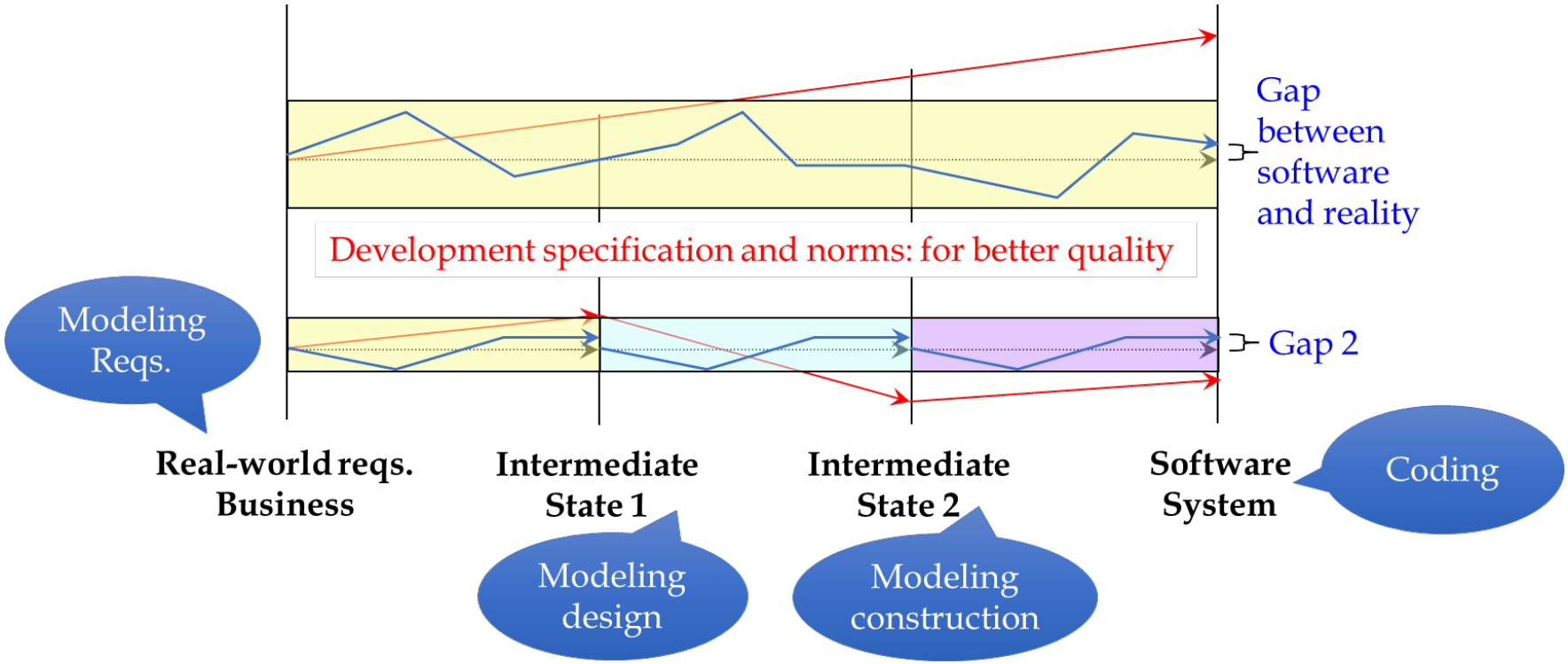
Miscellaneous Pane

Various components can appear in this pane – typically this contains a console and a list of compiler problems

(2) Modeling languages 建模语言

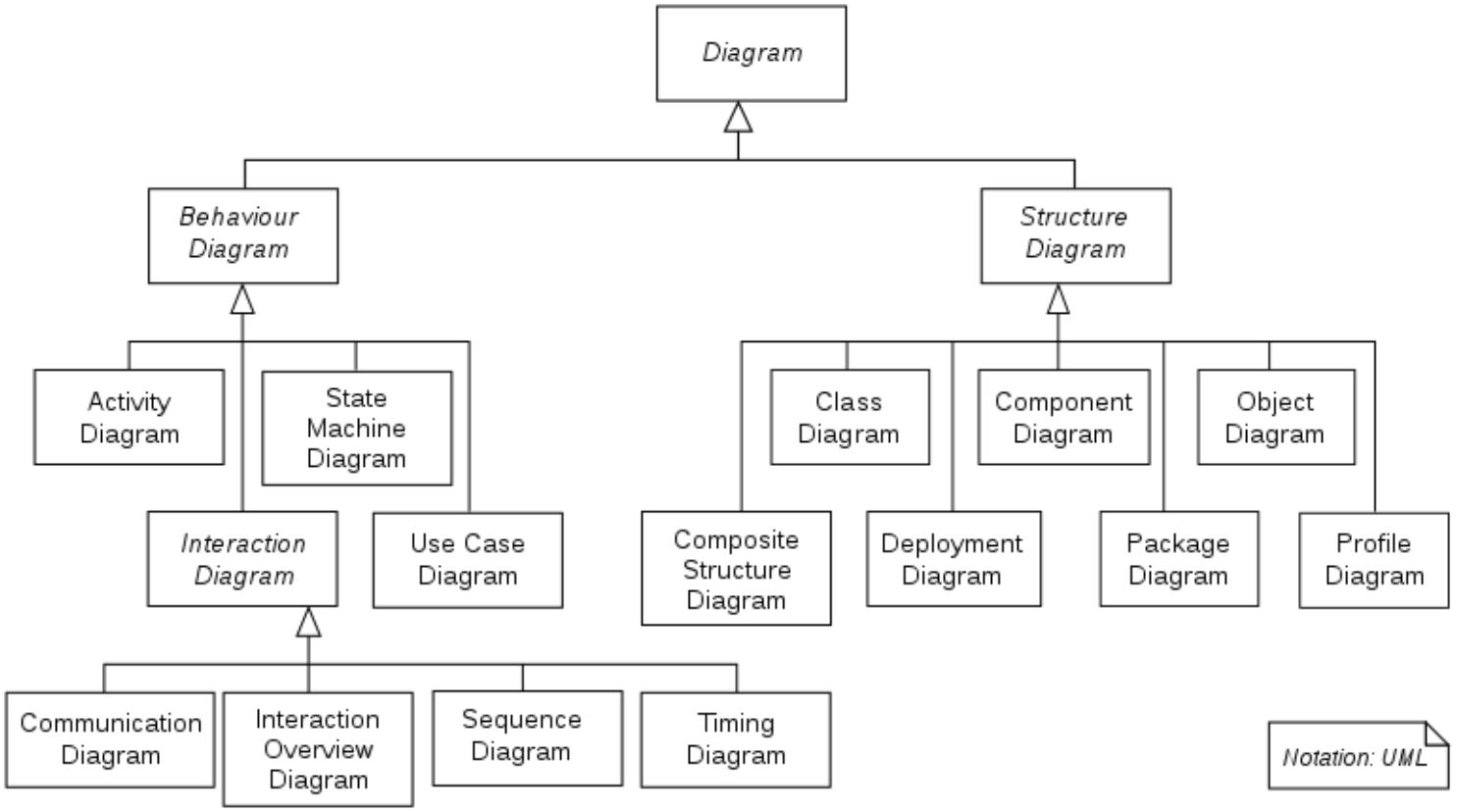
http://modeling-languages.com

- **A modeling language** is any artificial language that can be used to express information or knowledge or systems in a structure that is defined by a consistent set of rules, with the objective of **visualizing, reasoning, verifying and communicating the design of a system**.



UML as a modeling language and tool example

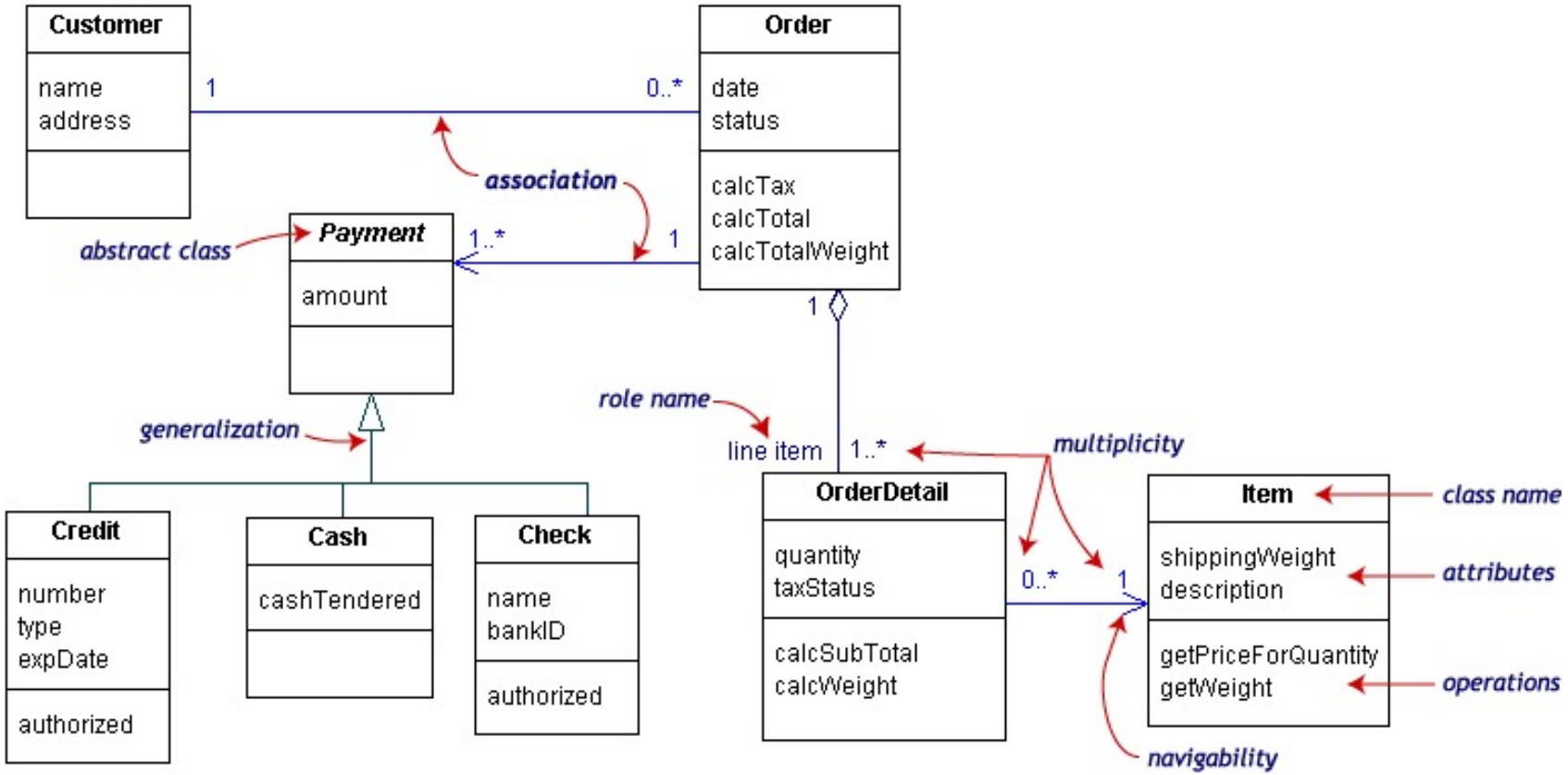
- **UML: Unified Modeling Language**



to learn UML in "Software Process and Tools" in the 3rd year

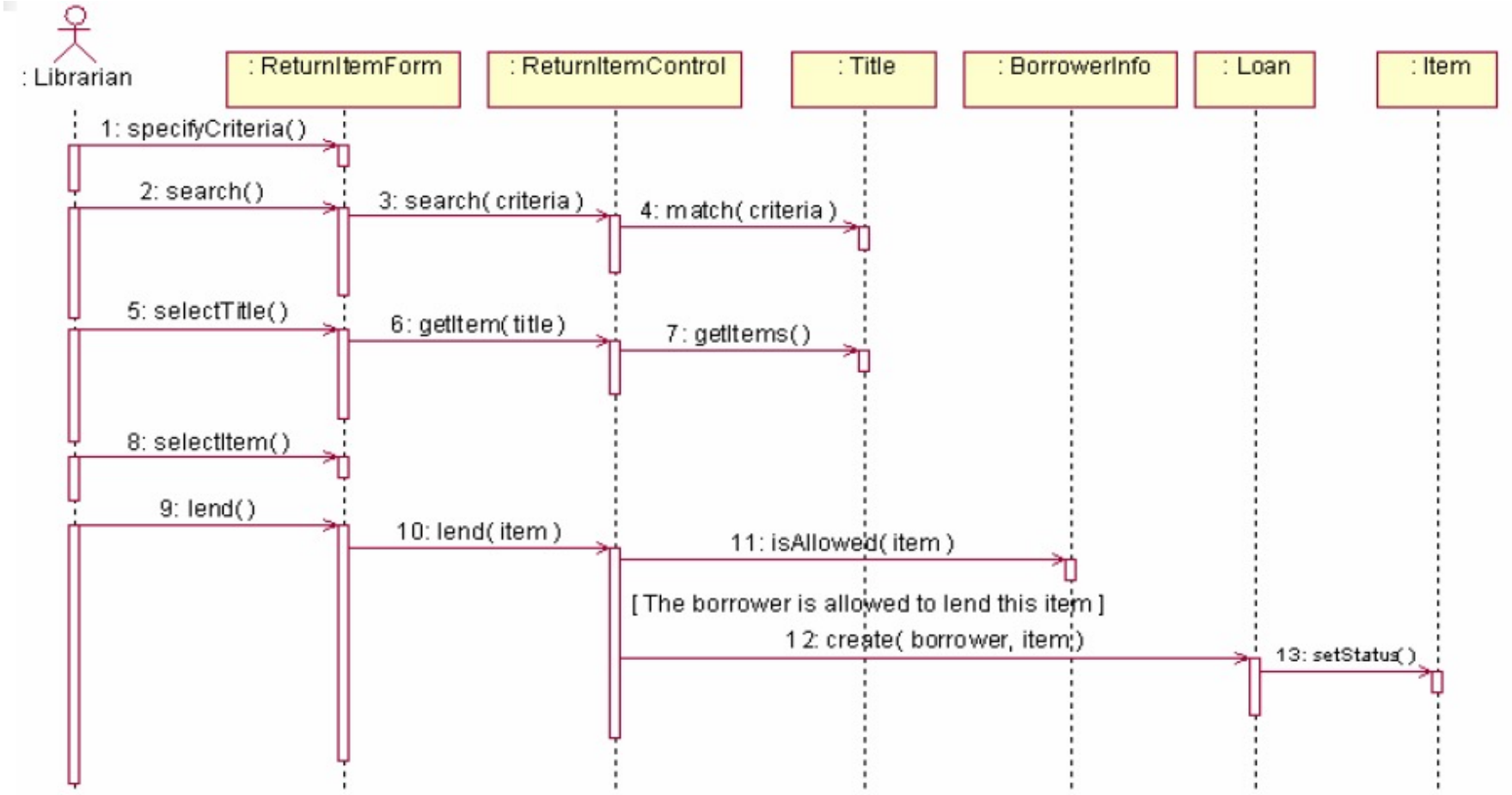
UML as a modeling language and tool example

- **UML Class Diagram**



UML as a modeling language and tool example

- **UML Sequence Diagram**



(3) Configuration languages 配置语言

- **Configuration files configure the parameters and initial settings for programs.**
  - Applications should provide tools to create, modify, and verify the syntax of their configuration files;
  - Some computer programs only read their configuration files at startup. Others periodically check the configuration files for changes.
- **Purpose example:**
  - Deployment environment settings
  - Variants of application features
  - Variants of connections between components
- **Configuration language examples:**
  - Key-Value texts (.ini, .properties, .rc, etc)
  - XML, YAML, JSON

To change behaviors of software at run-time

To separate stable and unstable parts

Configuration languages

```
- <employees>
  - <person id="1392">
      <name>John Smith</name>
      <dob>1974-07-25</dob>
      <start-date>2004-08-01</start-date>
      <salary currency="USD">35000</salary>
    </person>
  - <person id="1395">
      <name>Clara Tennison</name>
      <dob>1968-03-15</dob>
      <start-date>2003-05-16</start-date>
      <salary currency="USD">27000</salary>
    </person>
  </employees>
```

```
{
  "configuration": {
    "name": "Default",
    "properties": {
      "property": [...]
    },
    "appenders": {
      "Console": {"name": "Console-Appender"...},
      "File": {"name": "File-Appender"...},
      "RollingFile": {"name": "RollingFile-Appender"...}
    },
    "loggers": {
      {"name": "guru.springframework.blog.log4j2json"...},
      "level": "debug"...}
```

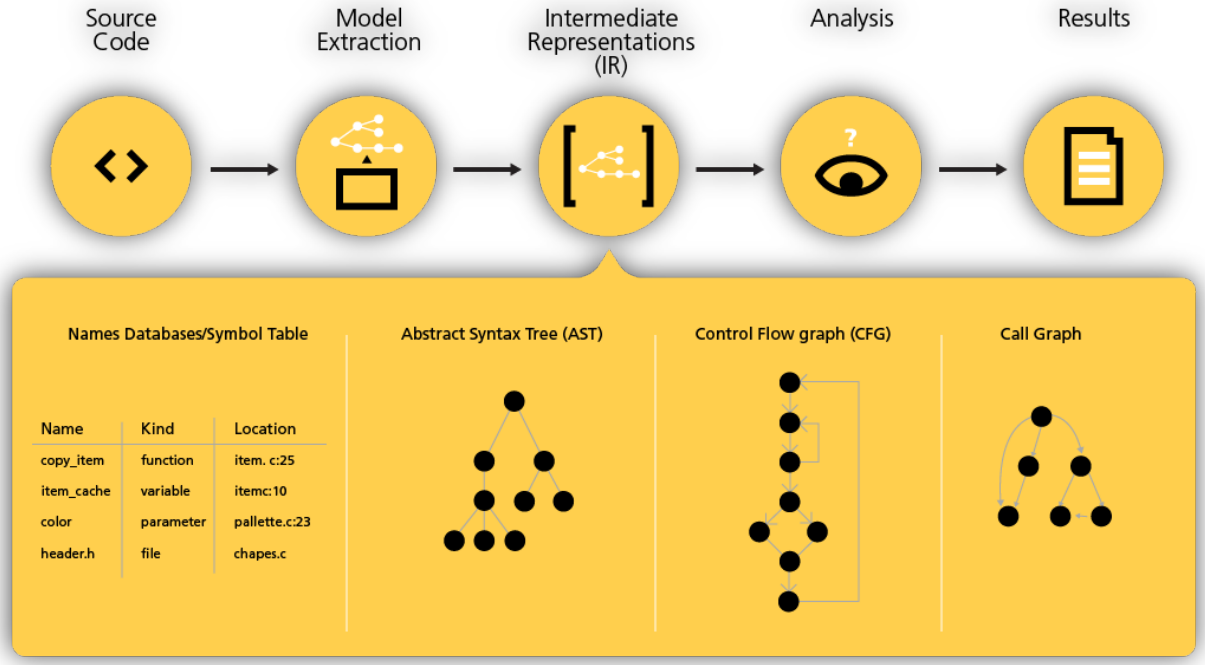

```
<?xml version="1.0"?>
<!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD Struts Configuration 1.1//EN"
        "http://jakarta.apache.org/struts/dtds/struts-config_1_1.dtd">
<struts-config>
    <!-- ========== ActionForm Definitions ============================== -->
    <form-beans>
        <form-bean name="userForm" type="org.apache.struts.action.DynaActionForm">
            <form-property name="property1" type="java.lang.String"/>
        </form-bean>
    </form-beans>

    <!-- ========== Global Forward Definitions ========================== -->
    <global-forwards type="org.apache.struts.action.ActionForward">
        <forward name="next" path="/forwardedPage.jsp" />
    </global-forwards>

    <!-- ========== Action Mapping Definitions ========================== -->
    <action-mappings type="org.apache.struts.action.ActionMapping">
        <!-- Process a user logon -->
        <action path="/user" type="org.rollerjm.presentation.UserAction" name="userForm" scope="session">
            <forward name="next" path="/forwardedPage.jsp"/>
        </action>
    </action-mappings>

    <!-- ========== Applications resources ============================== -->
    <message-resources parameter="org.rollerjm.presentation.struts.resources.ApplicationResources"/>
</struts-config>
```

```
{ "users":[
            {
                "firstName":"Ray",
                "lastName":"Villalobos",
                "joined": {
                    "month":"January",
                    "day":12,
                    "year":2012
                }
            },
            {
                "firstName":"John",
                "lastName":"Jones",
                "joined": {
                    "month":"April",
                    "day":28,
                    "year":2010
                }
            }
    ]}
```

(2) Review and static code analysis

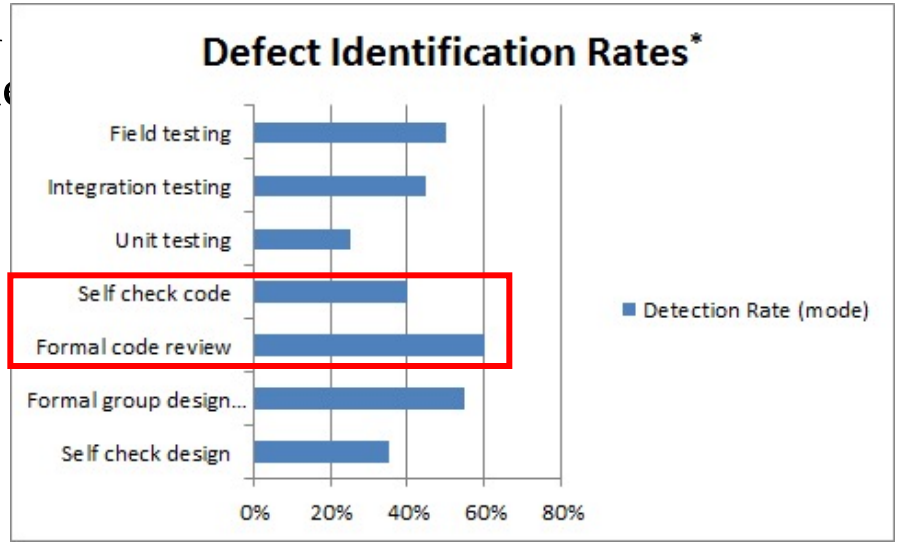
Review and static analysis/checking 代码评审

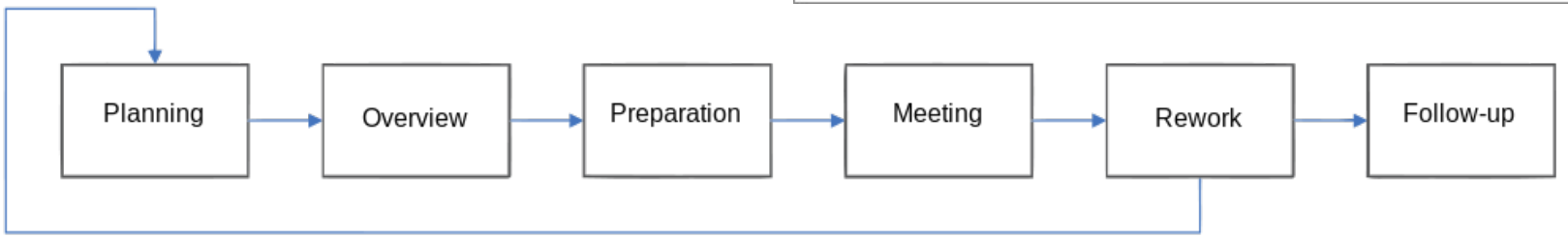
- **Code review is systematic examination (peer review) of source code.**
  - Intended to find mistakes overlooked in the initial development phase, improving the overall quality.
  - Reviews are done in various forms such as pair programming, informal walkthroughs, and formal inspections.
  - 结对编程
  - 走查
  - 正式评审会议
  - 自动化评审



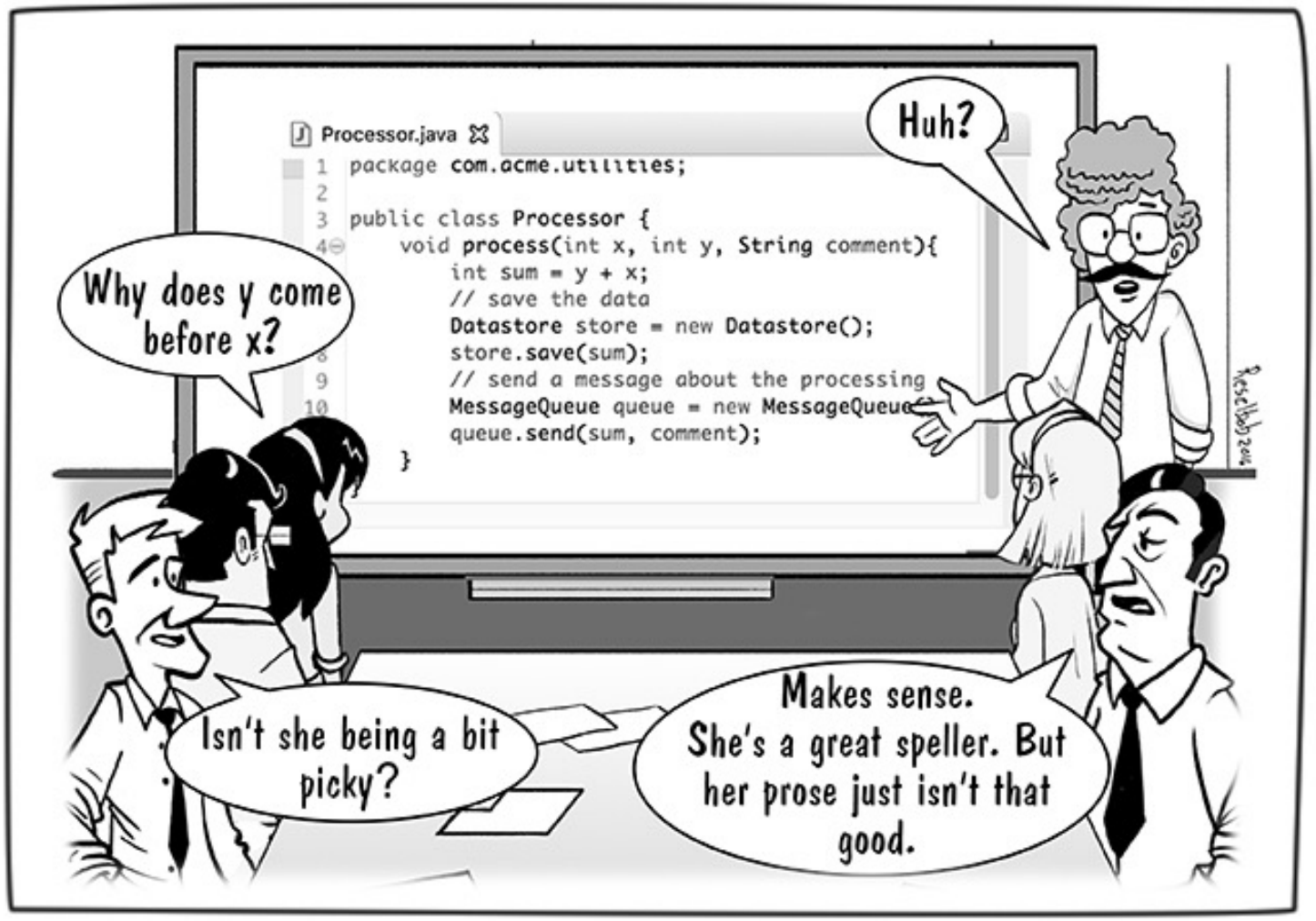
Formal code review 正式的代码评审会议

- **Formal code review, such as a Fagan inspection, involves a careful and detailed process with multiple participants and multiple phases.**
  - Formal code reviews are the traditional method of review, in which software developers attend series of meetings and review code line by line, usually using printed copies of the material.
  - Formal inspections are extremely thorough and have been proven effective at finding defects in the code under review.





Formal code review 正式的代码评审会议

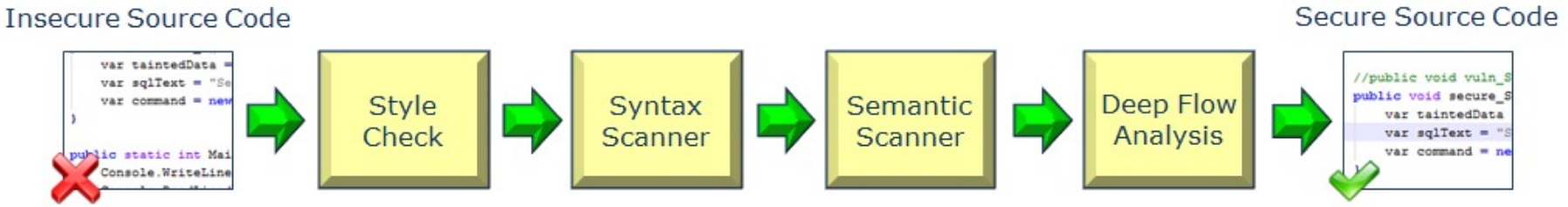


Lightweight code review 轻量级的代码评审

- **Lightweight code review typically requires less overhead than formal code inspections, though it can be equally effective when done properly.**
- **Lightweight reviews are often conducted as part of the normal development process:**
  - *Over-the-shoulder* – one developer looks over the author's shoulder as the latter walks through the code.
  - *Email pass-around* – source code management system emails code to reviewers automatically after checkin is made.
  - *Pair programming* – two authors develop code together at the same workstation, as it is common in Extreme Programming.
  - *Tool-assisted code review* – authors and reviewers use software tools, informal ones such as pastebins and IRC, or specialized tools designed for peer code review.

Static code analysis 利用工具进行的静态代码分析

- **Static code analysis is the analysis of computer software that is performed without actually executing programs (analysis performed on executing programs is known as dynamic analysis).**
- **The process provides an understanding of the code structure, and can help to ensure that the code adheres to industry standards.**
- **Automated tools can assist programmers and developers in carrying out static analysis.**
  - e.g., **CheckStyle, SpotBugs, PMD** for Java



Purpose of Code Review

- **Code review really has two purposes:**
  - Improving the code. Finding bugs, anticipating possible bugs, checking the clarity of the code, and checking for consistency with the project's style standards.
  - Improving the programmer. Code review is an important way that programmers learn and teach each other, about new language features, changes in the design of the project or its coding standards, and new techniques. In open source projects, particularly, much conversation happens in the context of code reviews.
- **Code review is widely practiced in open source projects like Apache and Mozilla.**
- **Code review is also widely practiced in industry.**
  - At Google, you can't push any code into the main repository until another engineer has signed off on it in a code review.

Some concrete examples of Code Review

Bugs or potential bugs.

- Repetitive code (remember DRY, Don't Repeat Yourself).
- Disagreement between code and specification.
- Off-by-one errors.
- Global variables, and other too-large variable scopes.
- Optimistic, undefensive programming.
- Magic numbers.
- …

Unclear, messy code.

- Bad variable or method names.
- Inconsistent indentation.
- Convoluted control flow (if and while statements) that could be simplified.
- Packing too much into one line of code, or too much into one method.
- Failing to comment obscure code.
- Having too many trivial comments that are simply redundant with the code.
- Variables used for more than one purpose.
- …

Some concrete examples of Code Review

Misunderstandings of Java.

Misuse of == or `.equals()`
Inadvertent use of integer division instead of floating-point division.
Use of a fixed-size array where a variable-length List would be more appropriate.
…

Code review is usually based on "programming experience".

Misusing (or failing to use) essential design concepts.

Incomplete or incorrect specification for a method or class.
Representation exposure for a data abstraction.
Immutable datatypes that expose themselves to change.
Invariants that aren't really invariant, or aren't even stated.
Failure to implement the `Object` contract correctly (`equals` and `hashCode`).
…

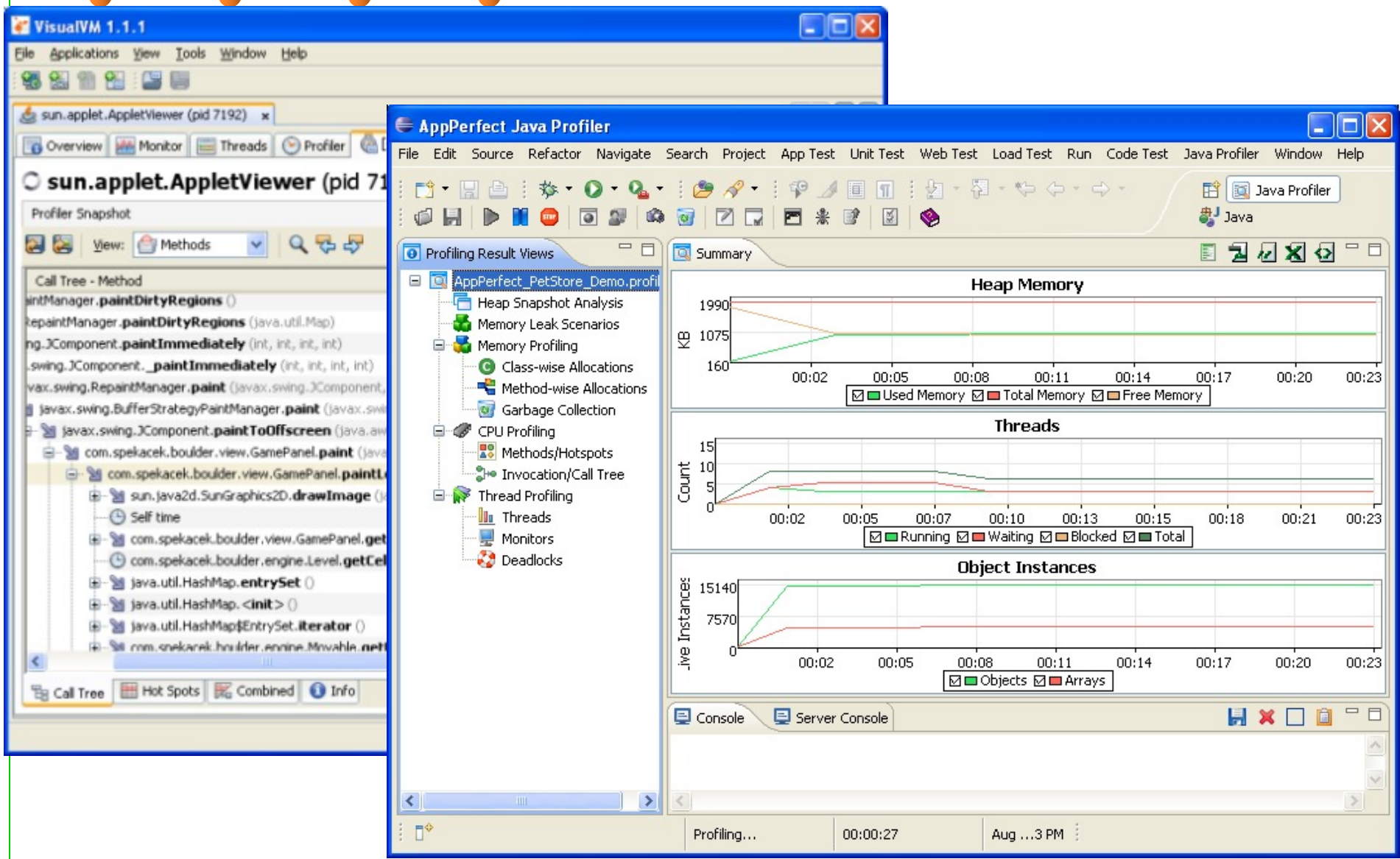
(3) Dynamic code analysis / profiling

Dynamic code analysis / profiling

- **Dynamic program analysis is the analysis of software that is performed by executing programs. 动态分析：要执行程序并观察现象、收集数据、分析不足**
- **The target program must be executed with sufficient test inputs to produce interesting behavior.**
- **Use of software testing measures such as code coverage helps ensure that an adequate slice of the program's set of possible behaviors has been observed.**

- **Profiling ("program profiling", "software profiling") is a form of dynamic program analysis that measures the space (memory) or time complexity of a program, the usage of particular instructions, or the frequency and duration of function calls.**
  对代码的运行时状态和性能进行度量，发现代码中的潜在问题

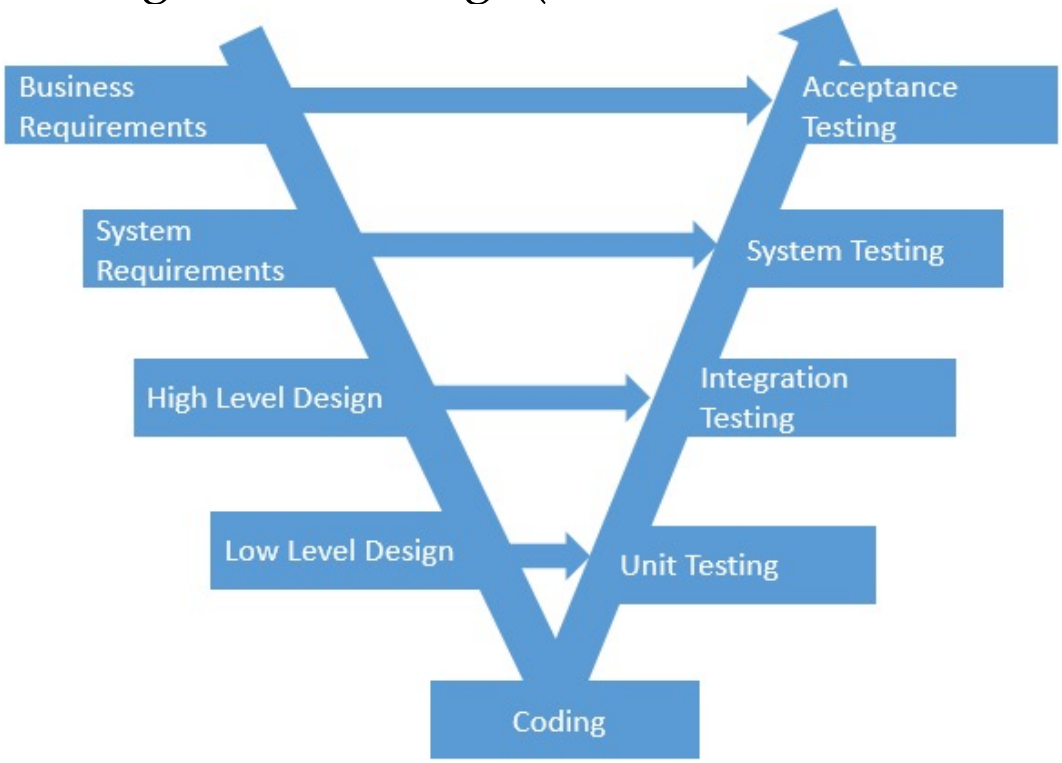
Dynamic code analysis / profiling



(4) Debugging and Testing

What is testing? 测试：发现程序是否有错误

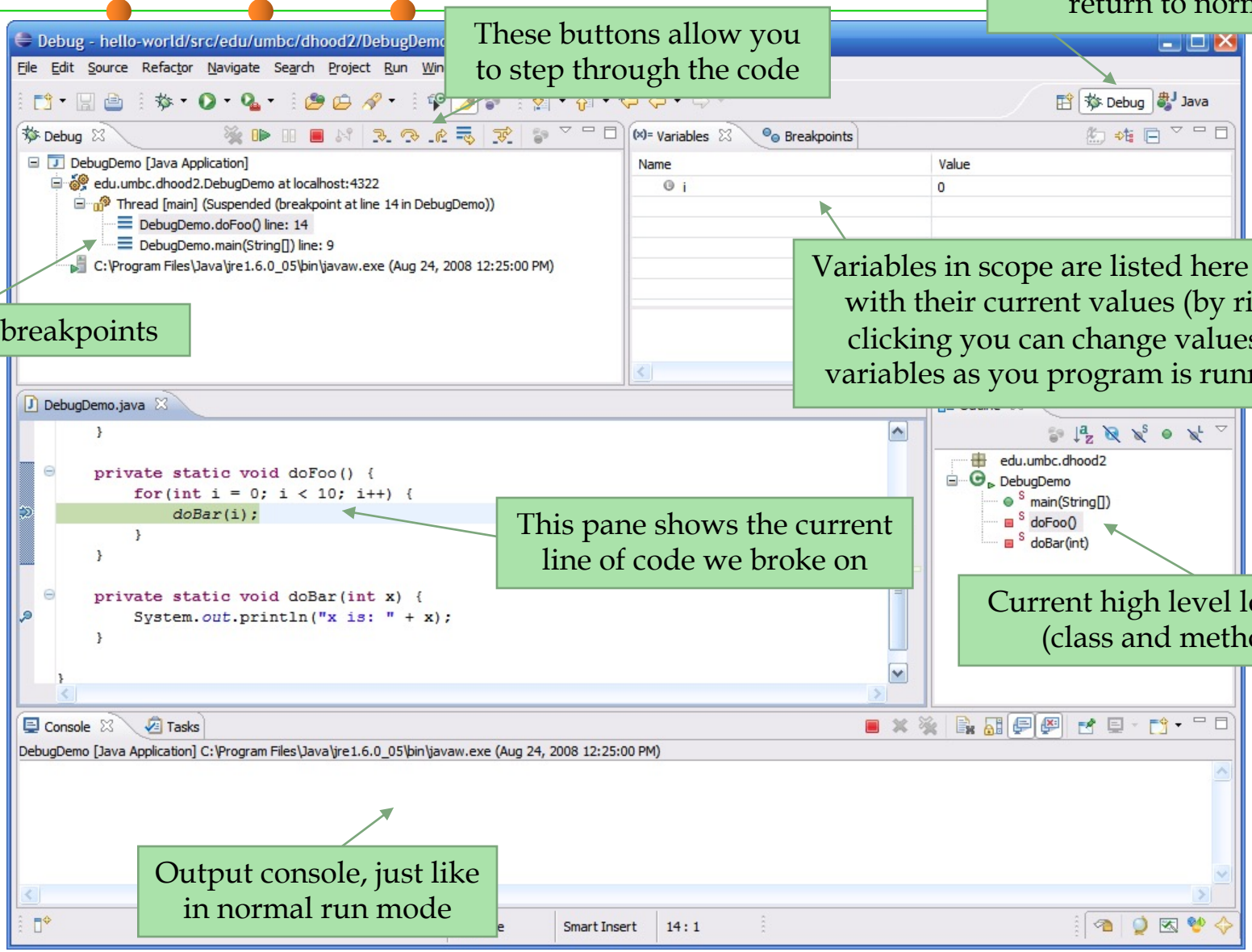
- **Software testing** is an investigation conducted to provide stakeholders with information about the quality of the product or service under test.
- **Test techniques** include the process of executing a program or application with the intent of finding software bugs (errors or other defects), and verifying that the software product is fit for use.
- **Software testing** involves the execution of a software component or system component to evaluate one or more properties of interest.



What is Debugging? 调试：定位错误、发现错误根源

- **Debugging is the process of identifying the root cause of an error and correcting it.**
- **It contrasts with testing, which is the process of detecting the error initially, debugging occurs as a consequence of successful testing.**
  - On some projects, debugging occupies as much as 50 percent of the total development time.
  - For many programmers, debugging is the hardest part of programming.
- **Like testing, debugging isn't a way to improve the quality of your software, but it's a way to diagnose defects.**
  - Software quality must be built in from the start. The best way to build a quality product is to develop requirements carefully, design well, and use high-quality coding practices.
  - Debugging is a last resort.

Debug perspective in Eclipse

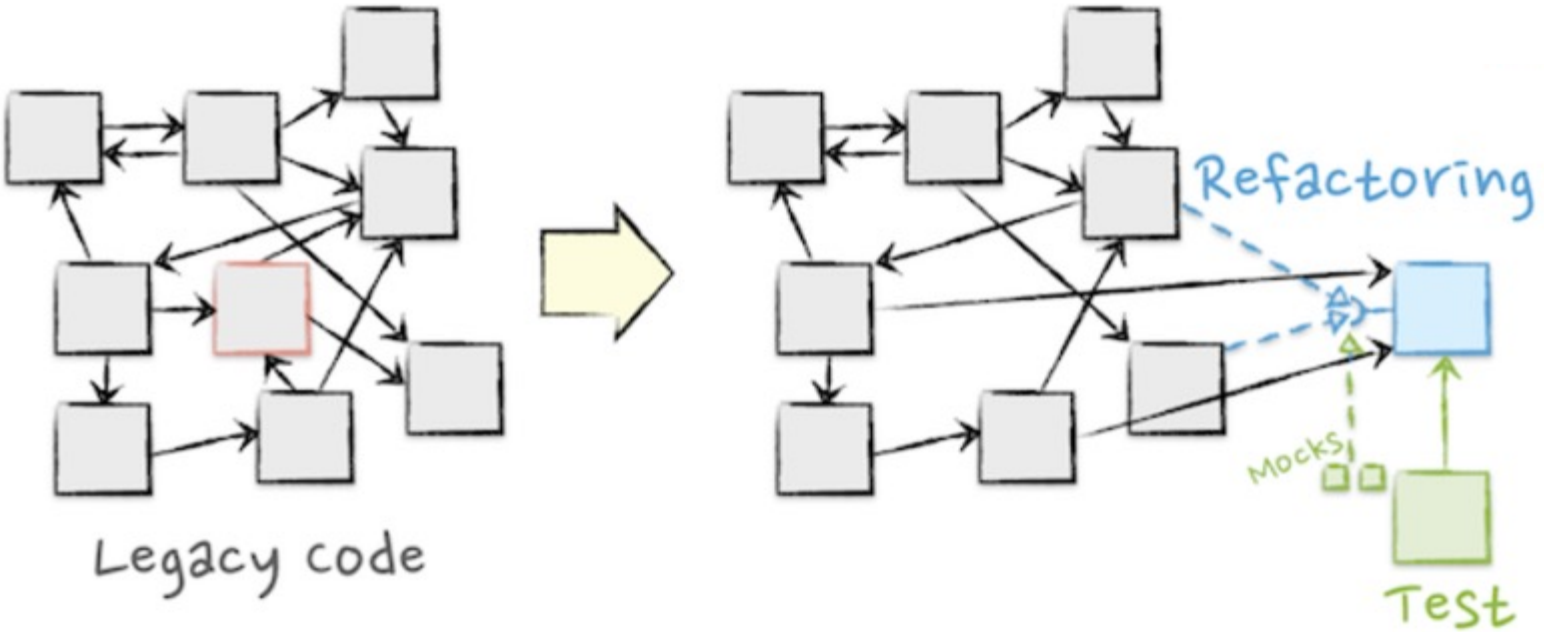


(5) Refactoring

Refactoring 重构：在不改变功能的前提下优化代码

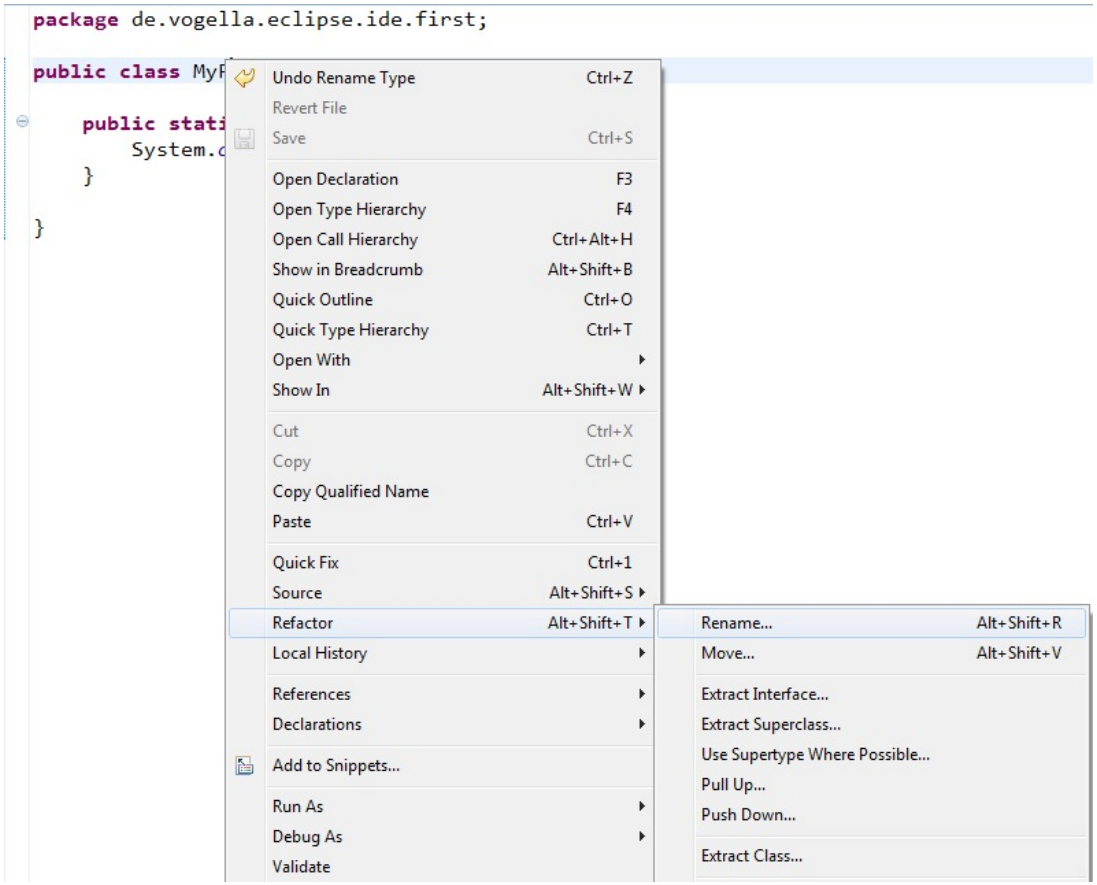
- **Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure.**
  - Incurs a short-term time/work cost to reap long-term benefits, and a long-term investment in the overall quality of your system.
- **Refactoring is:**
  - restructuring (rearranging) code...
  - ...in a series of small, semantics-preserving transformations...
  - ...in order to make the code easier to maintain and modify
- **Refactoring is not just any old restructuring**
  - You need to keep the code working
  - You need small steps that preserve semantics
  - You need to have unit tests to prove the code works

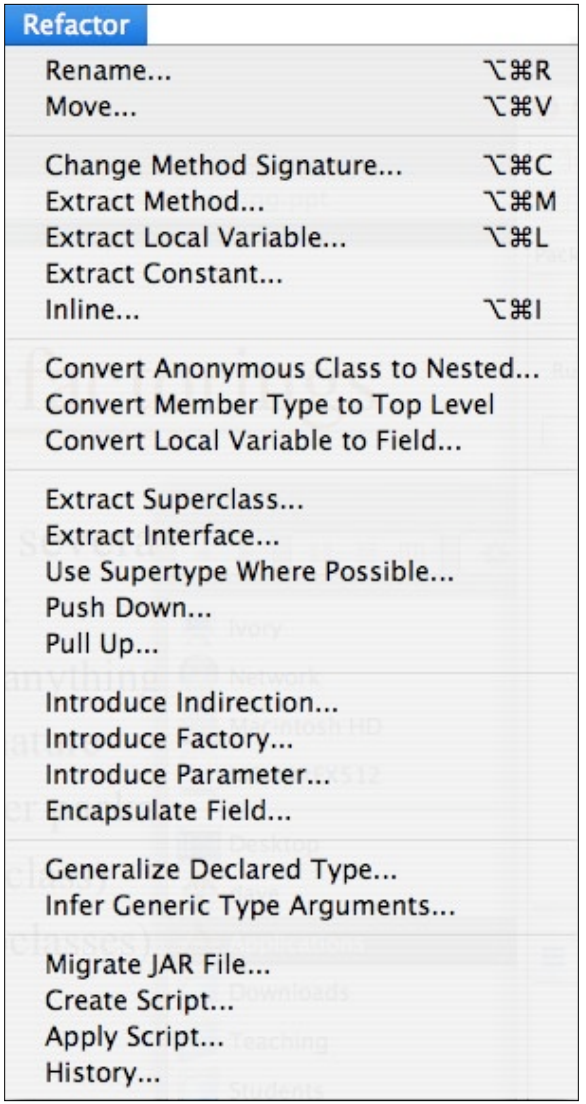
Refactoring



Eclipse IDE support for refactoring

- **Eclipse (and some other IDEs) provide significant support for refactoring**





Eclipse IDE support for refactoring

```
45    if (exists code = request.parameter("code")) {
46        value clientRequest
47                = ClientRequest {
48            uri = gitHubAuth;
49            Parameter("client_id", clientId),
50            Parameter("client_secret", clientSecret),
51            Parameter("code", code),
52
53        };
54        clientRequest.setHeader("Accept", "application/json");
55        clientRequest.setHeader("Cache-Control", "no-cache");
56
57        value json = clientRequest.execute().contents;
58        assert (is JsonObject result = parse(json),
```

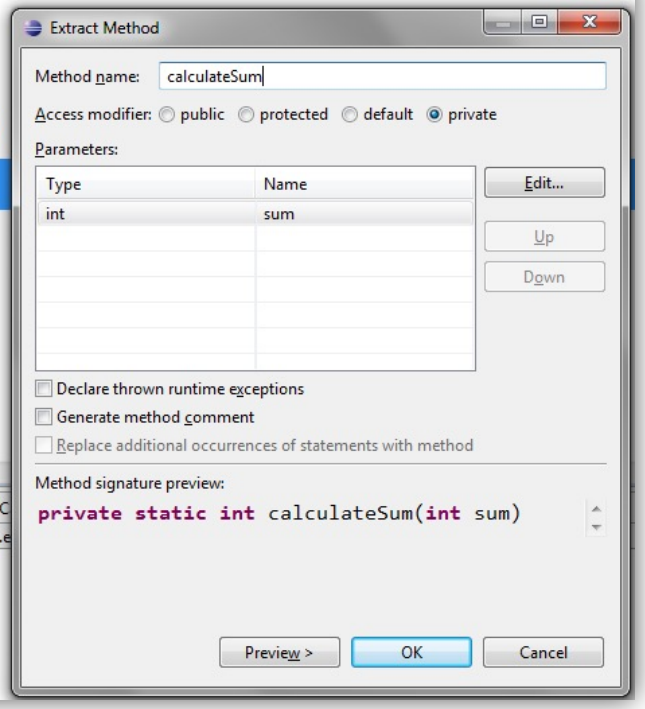
Enter new name for 4 occurrences of 'clientRequest' ⋯ ↩

```
public class MyFirstClass {

    public static void main(String[] args) {
        System.out.println("Hello Eclipse!");
        int sum = 0;
        for (int i = 0; i <= 100; i++) {
            sum += i;
        }
        System.out.println(sum);
    }
}
```

Problems | Javadoc | Declaration | Console | Error Log

<terminated> MyFirstClass [Java Application] C:\Program Files\Java\jre7\bin\javaw.e

```
Hello Eclipse!
5050
```



http://help.eclipse.org/luna/index.jsp?topic=%2Forg.eclipse.jdt.doc.user%2Freference%2Fref-menu-refactor.htm

External tools for refactoring

☆ 6 ○ 0

Install

Compositional Refactoring

This Eclipse bundle provides new Quick Assist actions to support refactorings in a compositional paradigm. In the compositional paradigm, the tool automates small, predictable steps of a refactorin...

Editor, Code Management, J2EE Development Platform, Source Code Analyzer, Application Development Frameworks

Last Updated on Monday, November 7, 2016 - 14:12 by Mohsen Vakilian

☆ 4 ○ 6

Install

AutoRefactor

AutoRefactor is an Eclipse plugin to automatically refactor Java code bases. The aim is to fix language/API usage in order to deliver smaller, more maintainable and more expressive code bases.

Source Code Analyzer, Tools, Languages, IDE, Code Management

Last Updated on Monday, November 7, 2016 - 14:17 by Jean-Noel Rouvignac

☆ 30 ○ 2

Install

Spartan Refactoring

Automatically find and correct fragments of code to make your Java's source code more efficient, shorter and more readable More info on https://www.spartan.org.il

Source Code Analyzer, Tools, Languages

Last Updated on Wednesday, February 22, 2017 - 07:26 by Daniel Mittelman

☆ 8 ○ 0

Install

Refactory

Refactory is a set of Eclipse plugins for creating refactorings for arbitrary models. It consists of a SDK to define new generic refactorings which then can be reused for any metamodel you like. Fu...

Modeling Tools, Languages, Modeling, IDE, Tools

Last Updated on Monday, November 7, 2016 - 14:20 by Jan Reimann

7* Narrow-sense process of software construction (Build)

粗略理解build：build-time → run-time

借助于工具，将软件构造各阶段的活动“自动化”

(编译、打包、静态分析、测试、生成文档、部署、…)

尽可能脱离“手工作业”，提高构造效率

Typical BUILD scenarios 使用build的典型场景

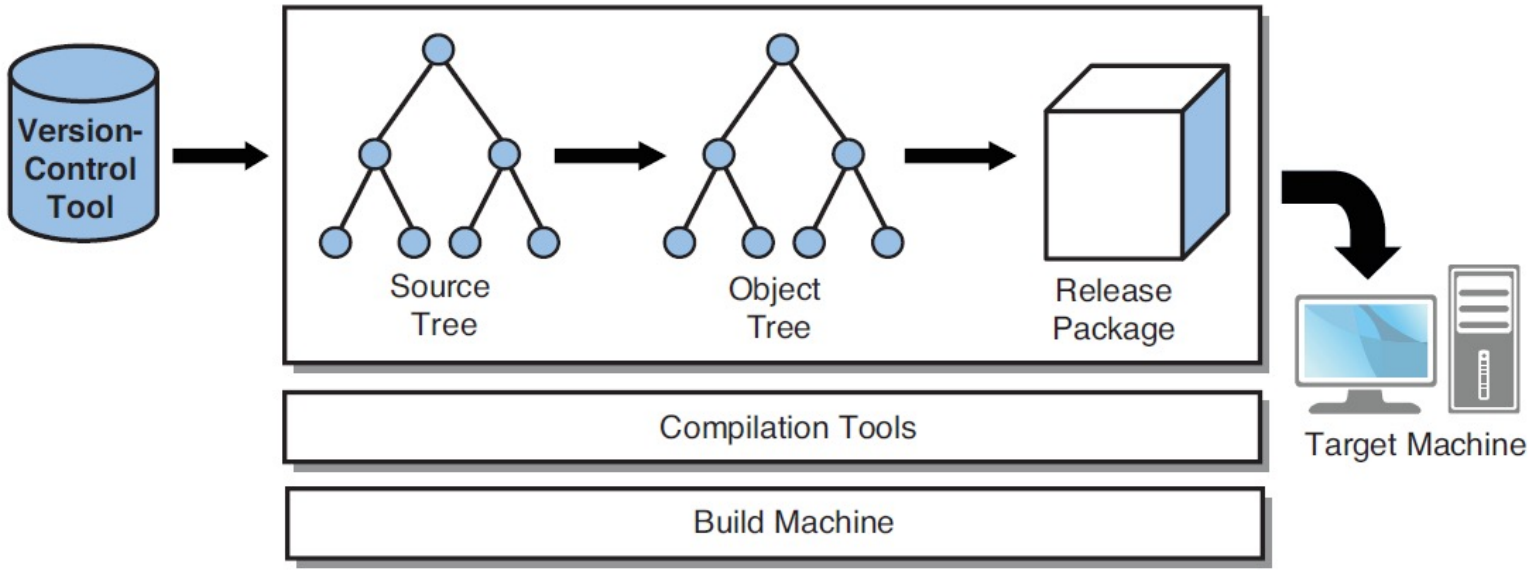
- **The compilation of software written in traditional compiled languages, such as C, C++, Java and C#.**
- **The packaging and testing of software written in interpreted languages such as Perl and Python.**
- **The compilation and packaging of web-based applications.**
  - These include static HTML pages, source code written in Java or C#, hybrid files written using JSP (JavaServer Pages), ASP (Active Server Pages), or PHP (Hypertext Preprocessor) syntax, along with numerous types of configuration file.

Typical BUILD scenarios 使用build的典型场景

- **The execution of unit tests to validate small portions of the software in isolation from the rest of the code.**
- **The execution of static analysis tools to identify bugs in a program's source code. The output from this build system is a bug report document rather than an executable program.**
- **The generation of PDF or HTML documentation. This type of build system consumes input files in a range of different formats but generates human-readable documentation as the output.**

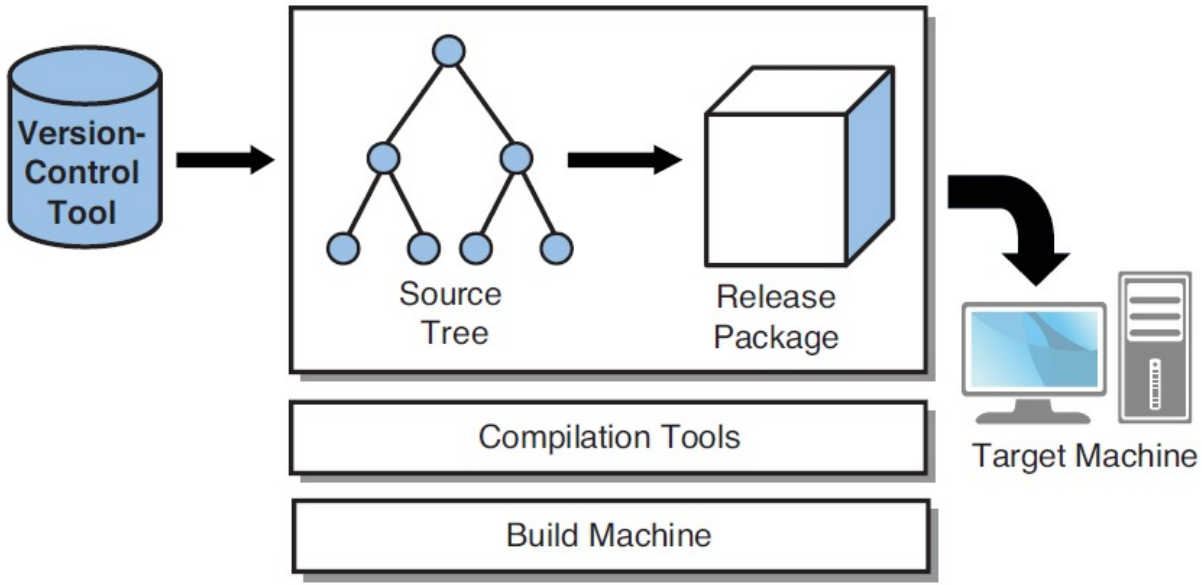
Compiled Languages

- **Compiled languages such as C, C++, Java, and C#. In this model, source files are compiled into object files, which are then linked into code libraries or executable programs.**
- **The resulting files are collected into a release package that can be installed on a target machine.**



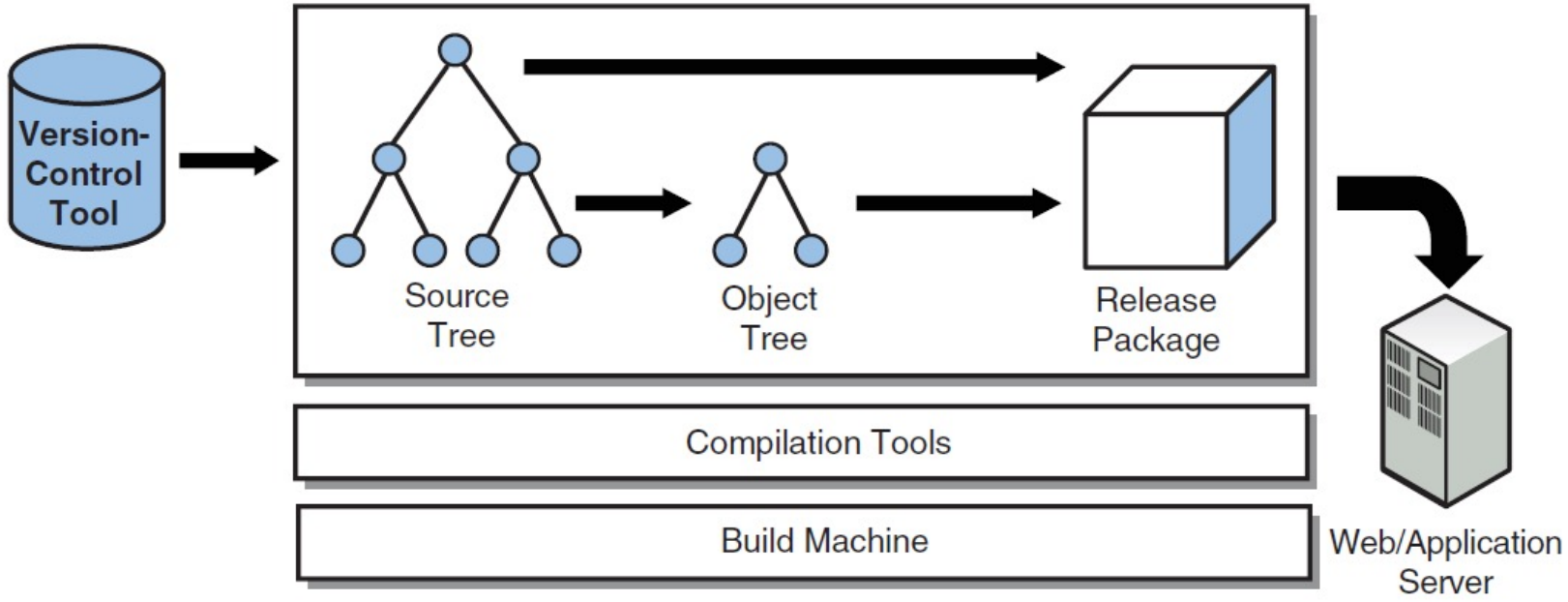
Interpreted Languages

- **Interpreted source code** isn't compiled into object code, so there's no need for an object tree. The source files themselves are collected into a release package, ready to be installed on the target machine.
- **Compilation** tools focus on transforming source files and storing them in the release package.
- Compilation into machine code is not performed at build time, even though it may happen at runtime.

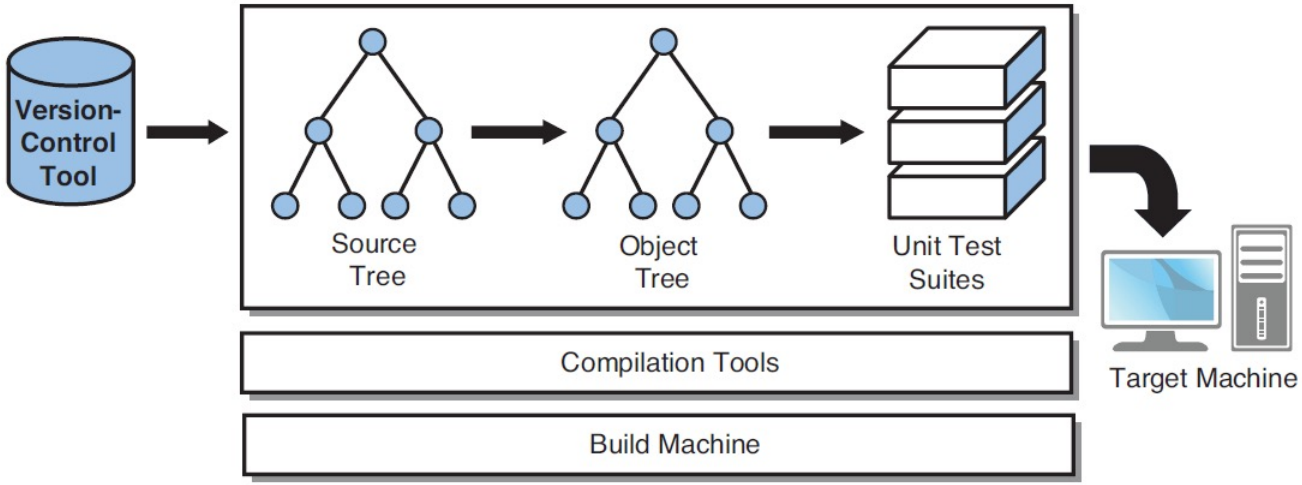


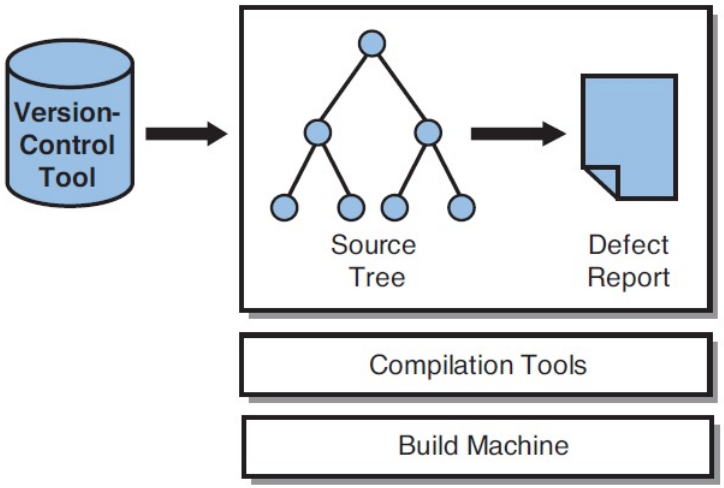
Web-Based Applications

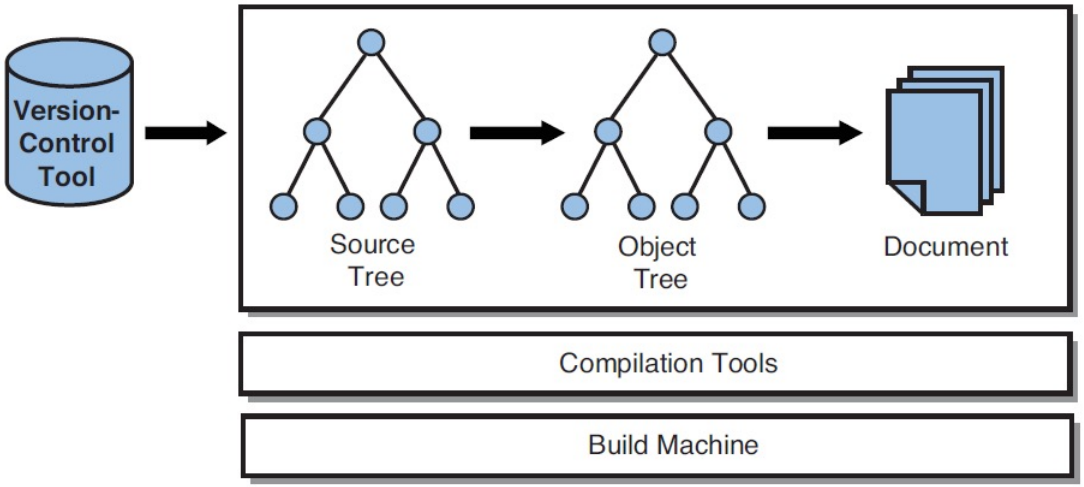
- The build system for a web-based application is a mix of compiled code, interpreted code, and configuration or data files.
- Some files (such as HTML files) are copied directly from the source tree to the release package, whereas others (such as Java source files) are first compiled into object code.



Unit Testing, Static Analysis, etc



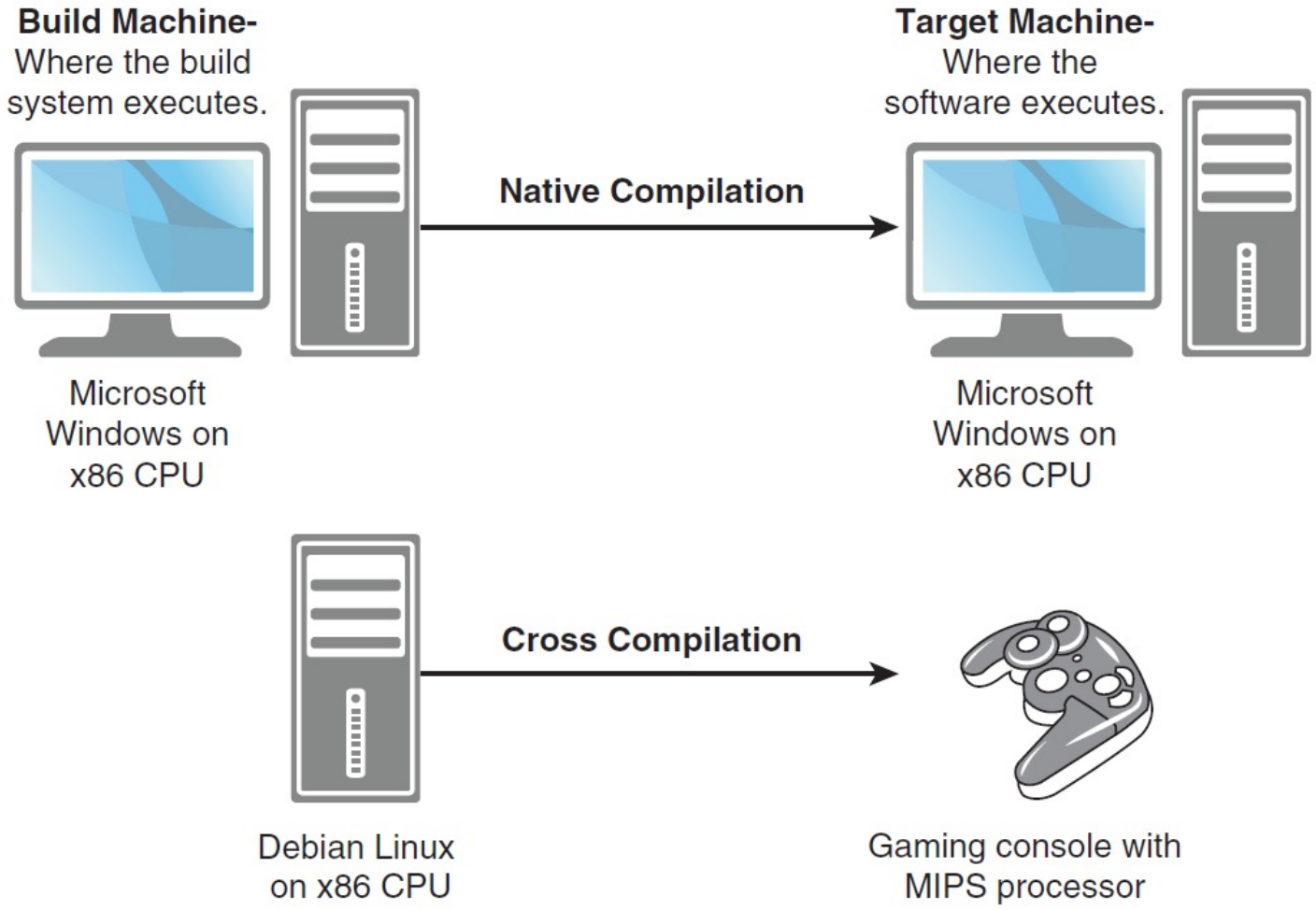




Components of a Build System

- **Version-Control Tools**
- **Source Tree:** a program's source code is stored as a number of disk files. This arrangement of the files into different is known as the **source tree**. The structure of the source tree often reflects the architecture of the software.
- **Object Trees:** a separate tree hierarchy that stores any object files or executable programs constructed by the build process.
- **Compilation Tools:** a program that translate the human-readable source files into the machine-readable executable program files.
  - Compiler: source files ⇒ object files
  - Linker: multiple related object files ⇒ executable program image
  - UML-based code generator: models ⇒ source code files
  - Documentation generator: scripts ⇒ documents

Native compilation vs. cross-compilation

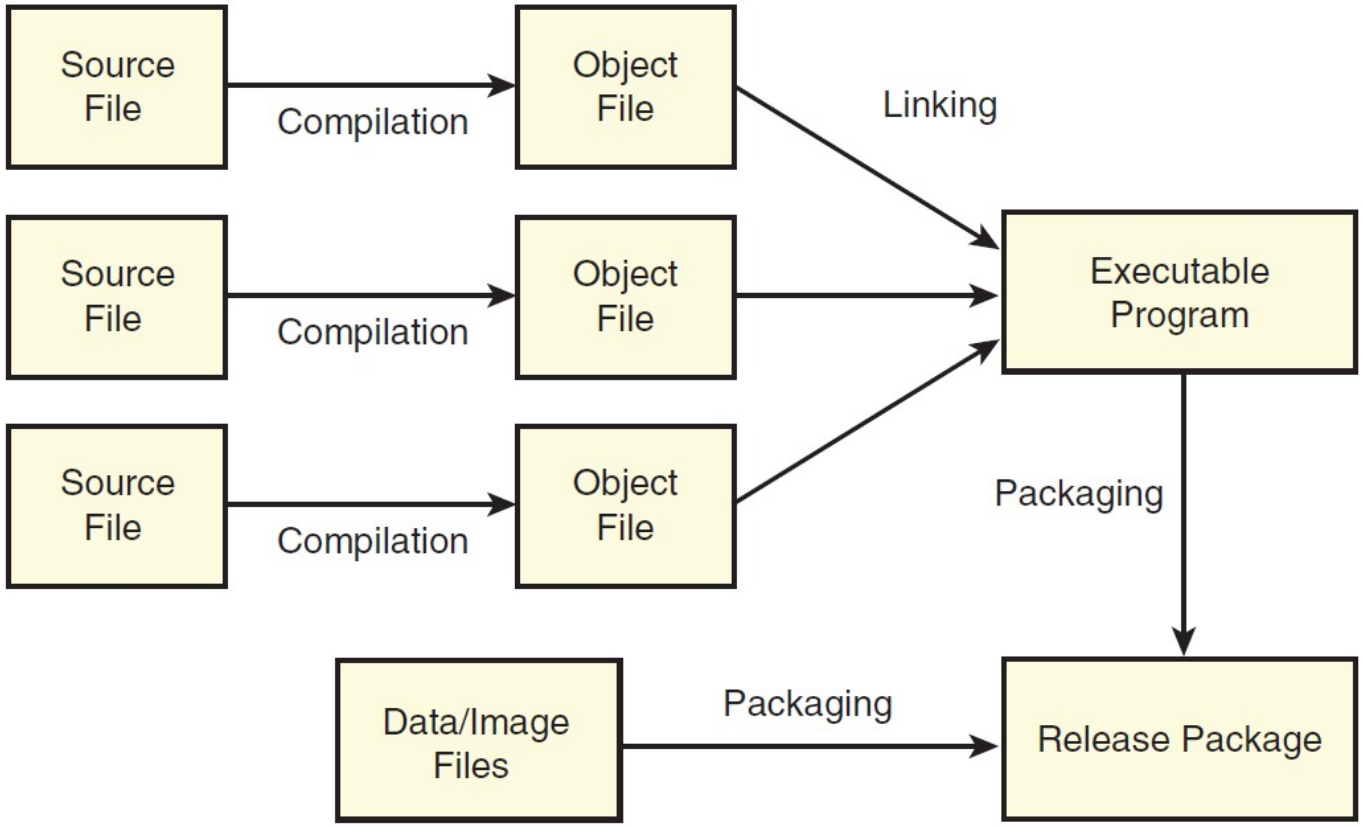


Components of a Build System

- **Release Packaging and Target Machines:** produces something that you can actually install on a user's machine.
  - To extract the relevant files from the source and object trees and store them in a release package.
  - The release package should be a single disk file and should be compressed to reduce the amount of time it takes to download.
  - Any nonessential debug information should be removed so that it doesn't clutter the software's installation.
- **Types of packaging:**
  - Archive files: zip and unzip
  - Package-management tools: UNIX-style such as .rpm and .deb
  - Custom-built GUI installation tools: Windows-style

Build process

- **Build process:** the build tool invokes each of the compilation tools to get the job done, which is an end-to-end sequence of events.



Build language (build description)

- A build tool needs the **build description** to be written in a text-based format.
  - It follows syntax rules of a specific build language.
- For example, when using `Make`, the interfile dependency information is specified in the form of **rules**, which are stored in a file named `Makefile`.
- You can write the description manually or generate it by IDE.

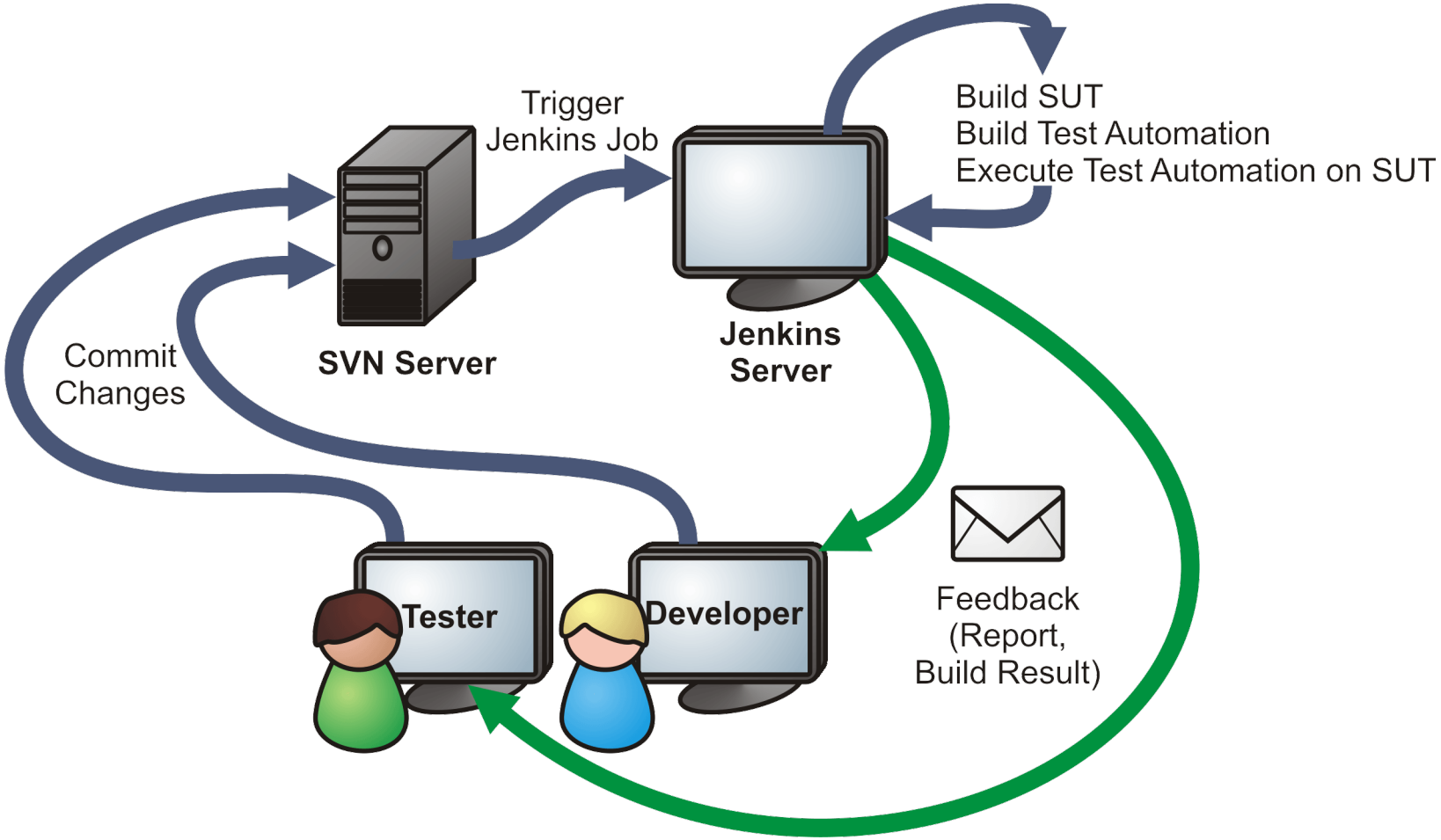
Makefile for a java program

```
JFLAGS = -g
JC = javac
.SUFFIXES: .java .class
.java.class:
        $(JC) $(JFLAGS) $*.java

CLASSES = \
        Foo.java \
        Blah.java \
        Library.java \
        Main.java

default: classes

classes: $(CLASSES:.java=.class)

clean:
        $(RM) *.class
```

How a build system is used

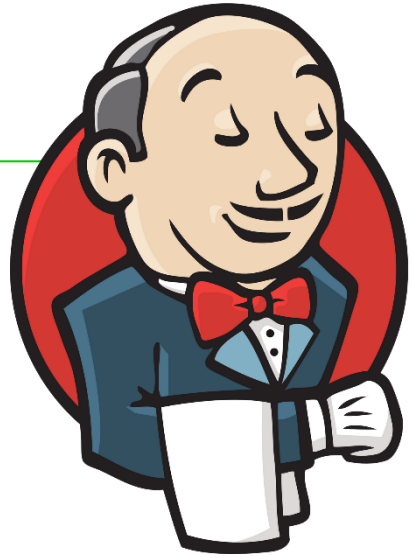
- **Developer (or private) build:** The developer has checked out the source code from VCS and is building the software in a private workspace.
- **Release build:** to provide a complete software package for the test group to validate. When the testers are convinced that the software is of high enough quality, that same package is made available to customers.
- **Sanity build:** The build process determines whether the current source code is free of errors and passes a basic set of sanity tests. This type of build can occur many times per day and tends to be fully automated.
  - Daily build / nightly build 每日构建
  - Continuous Integration (CI) 持续集成

Continuous Integration (CI) 持续集成

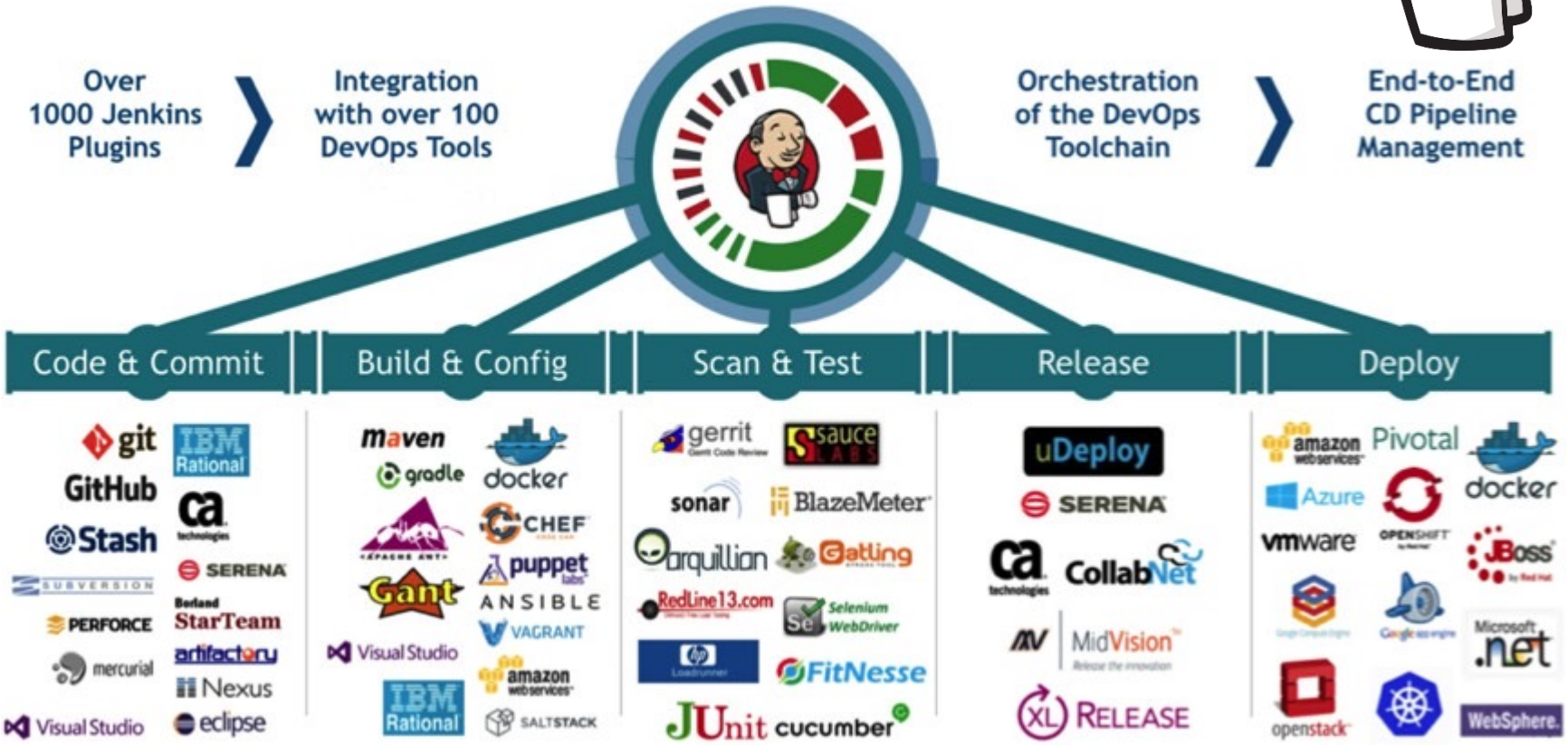


https://en.wikipedia.org/wiki/Continuous_integration

Jenkins



- https://jenkins.io



Build tools

- **For Java:**
  - Make
  - Ant
  - Maven
  - Gradle
  - Eclipse IDE

- **Build工具 + 相应的build script (类似于编程语言，告诉工具具体如何一步一步的build)**

- **建议最好学会Ant、Maven、Gradle中的一种，提高你的编程效率**

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**

Summary of this lecture

- **General process of software construction:** Design ⇒ Programming /refactoring ⇒ Debugging ⇒ Testing ⇒ Build ⇒ Release
  - Programming / refactoring
  - Review and static code analysis
  - Debugging (dumping and logging) and Testing
  - Dynamic code analysis /profiling
- **Narrow-sense process of software construction (Build):** Validate ⇒ Compile ⇒ Link ⇒ Test ⇒ Package ⇒ Install ⇒ Deploy
  - Build system: components and process
  - Build variants and build language
  - Build tools: Make, Ant, Maven, Gradle, Eclipse

The end

*