2024S Software Engineering

Lecturer： 陶伊达

本整理不适合预习，并且仍可能存在考点遗漏，建议与ppt一同食用

陶老师喜欢考一些图（挖空），建议背一下

Lecture 1

Programming VS. Software

对比包含以下四点:

LifeSpan

Short-term -- Long-term

No-change -- Adapt-to-change

Resource

Human resources

Computing resources

Complexity of Decisions

Artifacts

• Building a software is programming integrated over **time, scale, and trade-offs**.

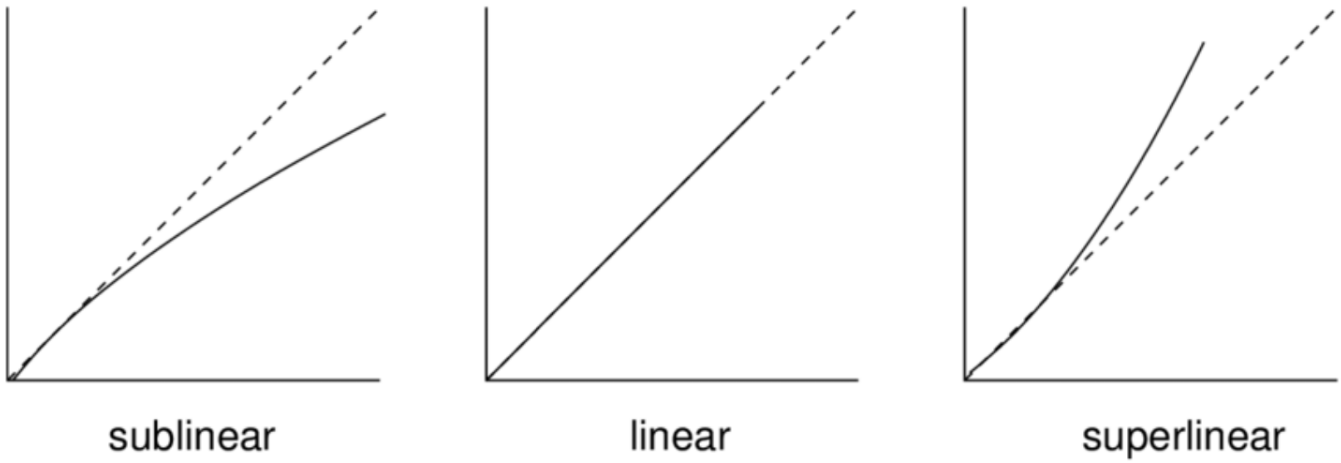
Hyrum's Law

Given enough time and enough users, even the most minor change WILL break something

Scale

X axis: the demand

Y axis: resources costs



superlinear is bad

Software engineering

Definition

Engineering is the application of an empirical, scientific approach to finding efficient solutions to practical problems.

Objective

quality, efficiency

Lecture 2

software process

7 PHASES IN SOFTWARE PROCESS

software process models

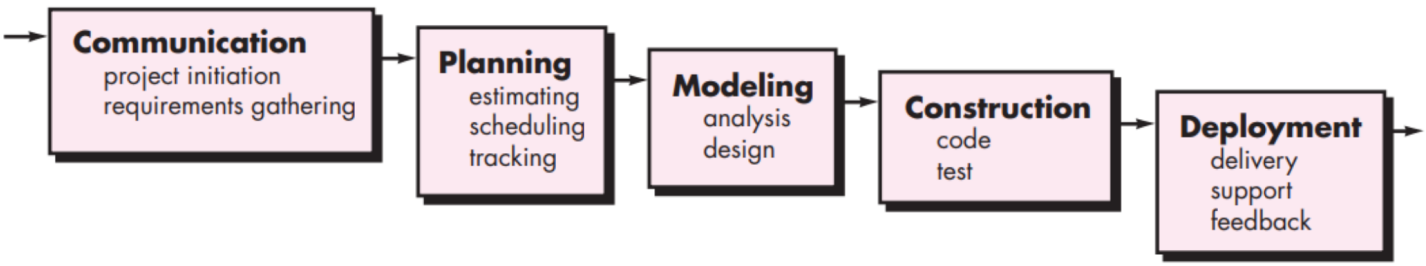
Broadly speaking, process models fall on a continuum

• **Plan-driven processes**: processes where all of the process activities are planned in advance and progress is measured against this plan.

• **Agile processes**: planning is incremental and it is easier to change the process to reflect changing customer requirements.

Plan-driven processes

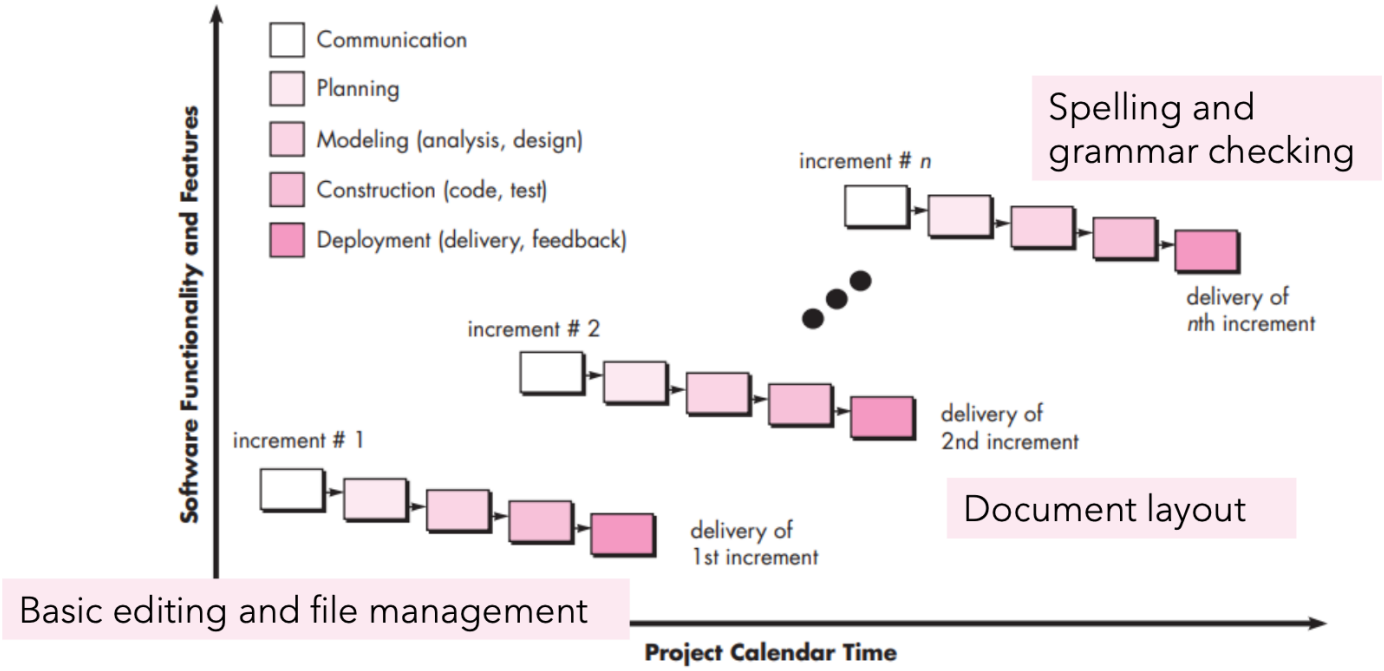
• **The Waterfall Model** -- sequential



适用场景： requirements are well defined and reasonably stable.

问题：sequential flow, need clear requirements, customer patience

• **Incremental Process Models**



优点：

- Reducing cost due to requirement changes
- Easier to get customer feedback and adapt
- Easier to test and debug during a smaller iteration
- More rapid delivery and deployment of useful software
- Particularly useful when staffing is unavailable for a complete implementation

缺点：

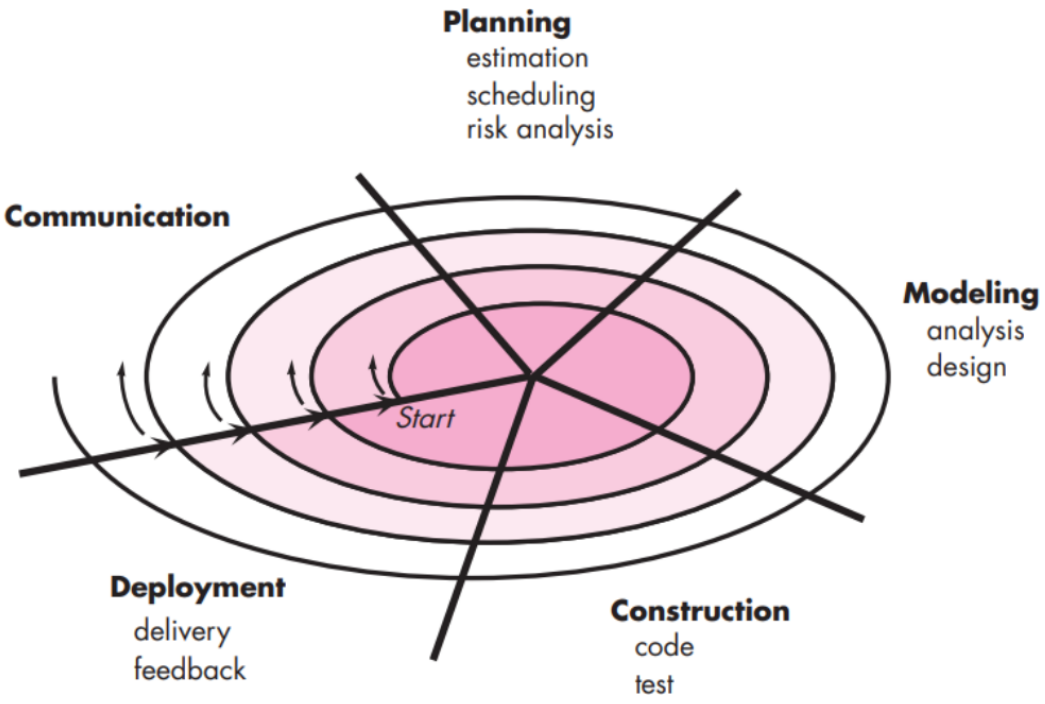
- Needs good planning and design of the whole system before breaking it down for incremental builds

- System structure tends to degrade as new increments are added. Unless effort is spent on refactoring to improve the software, regular change tends to corrupt its structure.
- Incorporating further software changes becomes increasingly difficult and costly.

• **Evolutionary Process Models**--the spiral model and prototyping

iterative (迭代)

• **The spiral model**



couples the iterative nature with the **waterfall mode**

early iterations: model or prototype.

later iterations: complete versions of the engineered system

effectively reducing potential risks

• **Prototyping**

Prototyping (原型开发) might be used when customers define a set of **general objectives** for software, but do not identify **detailed** requirements for functions and features.

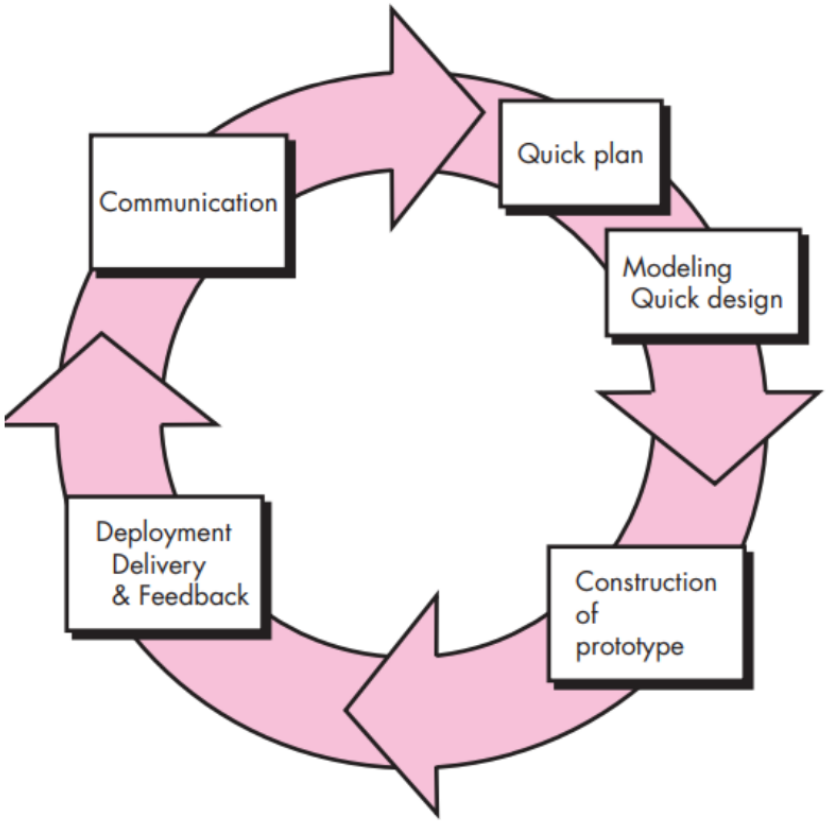
quick

make implementation compromises

The prototype is often **discarded** (at least in part), and the actual software is engineeredwith an eye toward **quality**.

✓ Prototyping should be used when the requirements are not clearly understood or are unstable

✓ Prototyping can also be used for developing UI or complex, high-tech systems in order to quickly demo the feasibility



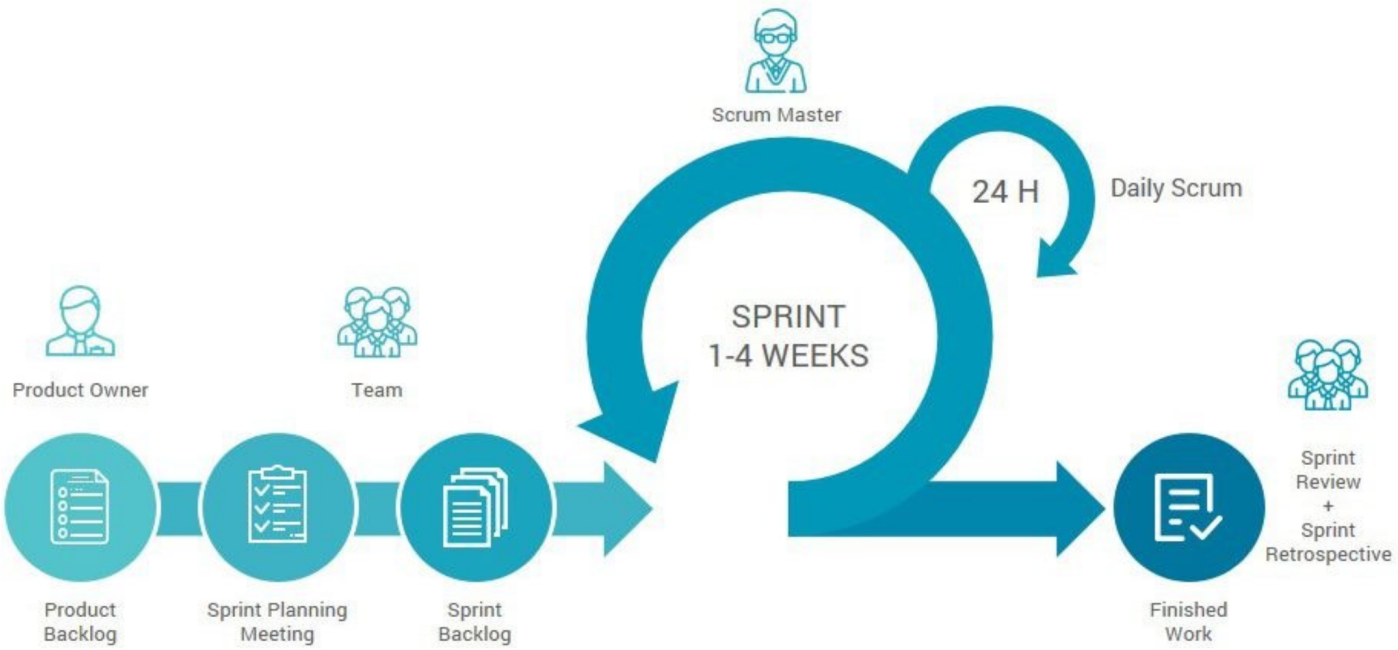
Agile

Agile (敏捷), in general, is the ability to create and respond to change.

Agile, in the context of software engineering, refers to the methods and best practices for organizing projects based on the values and principles documented in the **Agile Manifesto**.

iterative

Scrum--一种agile实现



Roles	Ceremonies	Artifacts
Product Owner	Sprint Planning	Product Backlog
Scrum Master	Daily Scrum	Sprint Backlog
Scrum Team	Sprint Review	Burndown Charts
	Sprint Retrospective	

Product Owner

Define and adjust product features; Accept or reject work results

Scrum Master

Coach the team and enact Scrum values and principles

Scrum Team

Cross-functional members like developers, testers, designers (Typically 5-9 size)

product owners created a **product backlog** (prioritized and dynamic)

product owner and the development team attend the **Sprint Planning Meeting**

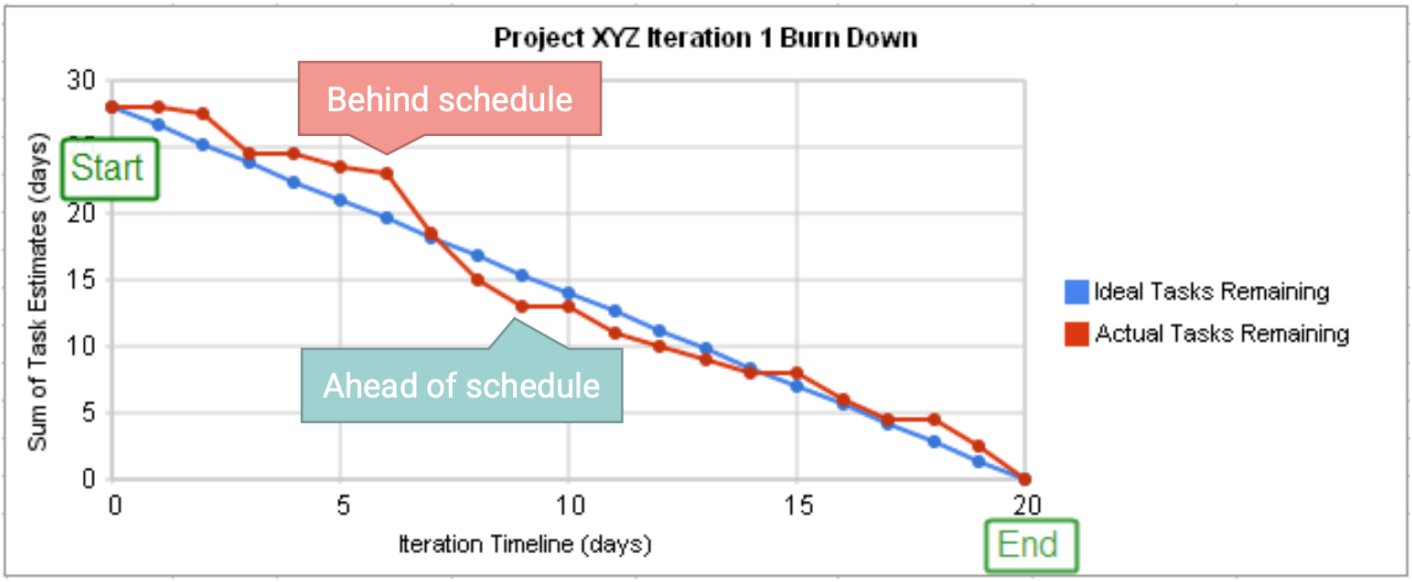
Sprint is a short, fixed period when a scrum team works to complete a set amount of work in the **Sprint Backlog**

A **sprint** duration is determined by the Scrum Master, typically lasts 30 days

A **Scrum Board** is a tool that helps Scrum Teams make **Sprint Backlog** items visible.

The board is traditionally divided up into three categories: To Do, Work in Progress and Done.

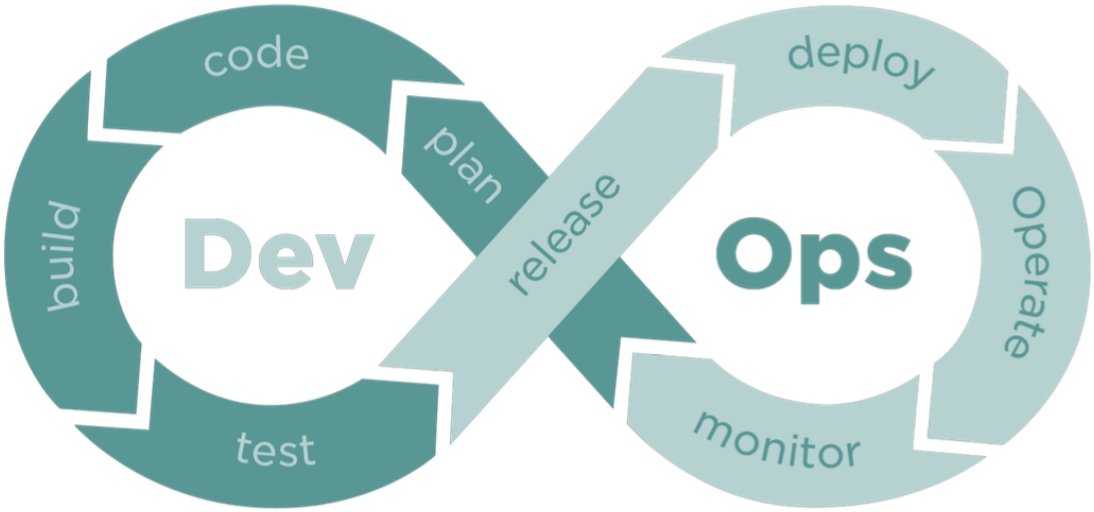
Burndown charts



Big Bang model: Developers jump right into coding without much planning, without any kind of clear roadmap

DevOps

下图必背



DevOps integrates developers and operations teams to improve collaboration and productivity by

• Automating infrastructure

• Automating workflows

• Continuously measuring application performance

• Continuous feedback

communicate frequently

Lecture 3

Requirements

Software requirements establish the system's functionality, constraints, and goals.

Requirements come from Application domain and **stakeholders**.

Primary Stakeholders

Primary stakeholders **have a direct impact** on your software project.

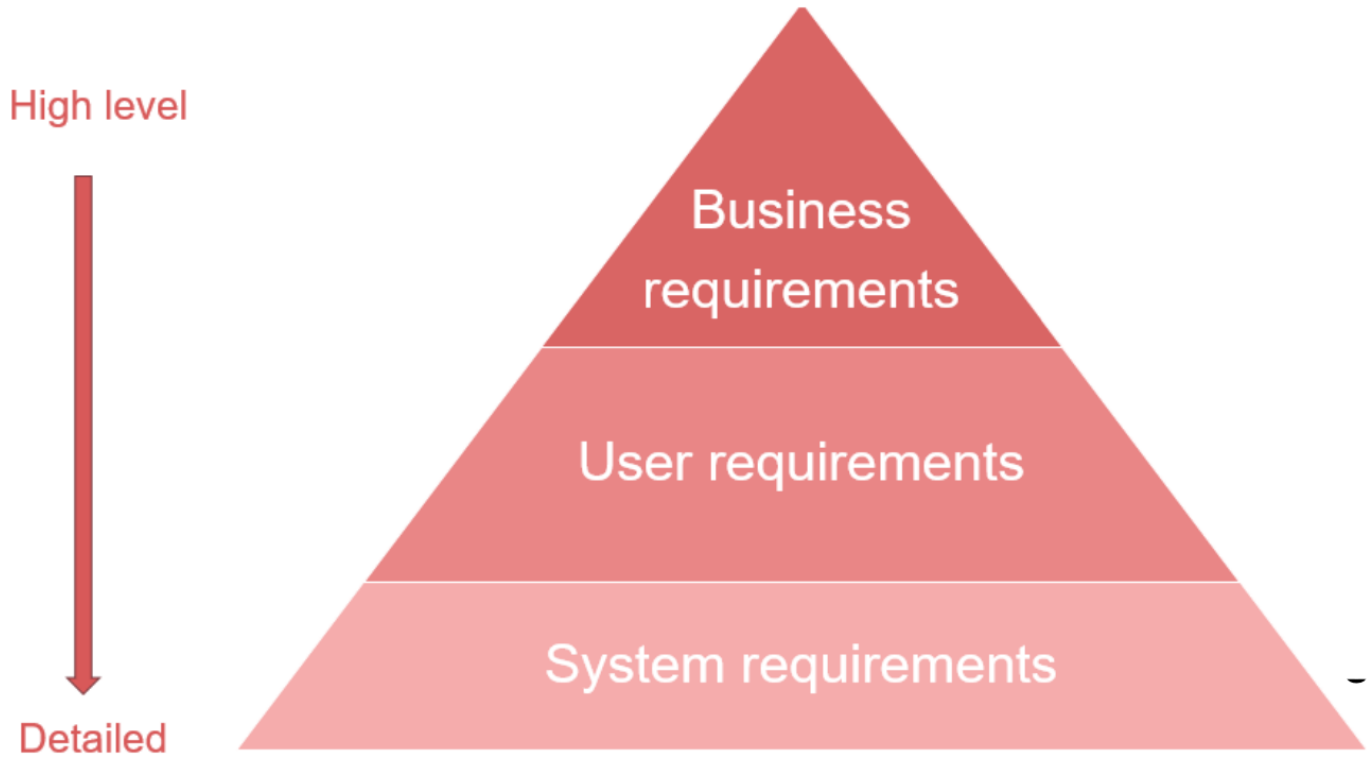
Customers, Project Managers, Business Analysts, Development Team, Quality assurance (QA), UI&UX designers

Secondary Stakeholders

Secondary stakeholders have an **indirect relationship** with a software development process.

End-users, Government, Competitors

Classification of software requirements



Business -- WHY

a general overview of a product

User -- WHO

use case, user scenarios, and user stories

NO technical details

System -- HOW

Functional/Non-Functional Requirements

for developers with more formal format

Functional Requirements

input to be given to the system, the operation performed and the output expected

Non-functional requirements

specify the software's quality attribute

affect the experience of the user

性能（performance）可靠性（reliability） 安全性（security） 易用性（usability）产品约束 (product constraints) 过程约束 (process constraints)

From *User story* to *Task*

The **product owner** is responsible for talking to all of the **stakeholders** and gathering requirements.

• Based on the gathered information, the **scrum team** work together to come up with **user stories**

As a <<role/ profile>> I want to <<action/ activity>> so that <<benefit/ reason>>

Stories come from the perspective of **users**

Stories create **business value** for the customers

Tasks

• Tasks are used to break down user stories even further.

• Tasks are the smallest unit used in scrum to track work.

• A task should be completed by one person on the team

• A task typically takes ~1 day

Acceptance Criteria

A product owner may be responsible for writing acceptance criteria for the stories in the product backlog

Scenario-based acceptance criteria

Also known as the Given/When/Then (GWT) acceptance criteria, with 5 components:

• Scenario: A description of the situation that the user will encounter.

• Given: The starting state of the scenario.

• When: The action that a user takes.

• Then: The result of the action in the previous “When”.

• And: used to continue any of three previous statements

Rule-based acceptance criteria

Importance

• Narrowing down the scope of the product

• Easier to test the product.

• Addressing invalid or negative outcomes.

Lecture 4

Version control

用途：**revert** files, **compare** changes, see **who** modified

to reduce risks caused by continuous changes and increase successful deployments

Local VCSs : RCS -- keep patch sets (the differences between files)

Centralized VCS: have a single server, clients **check out** files from that central place

Distributed VCS (such as Git, Mercurial,Bazaar or Darcs): fully mirror the repository

Git

Snapshots, Not Differences

不是delta-based version control

a series of snapshots of a miniature filesystem

Git Has Integrity

checksum：calculated based on the contents of a file or directory structure in Git

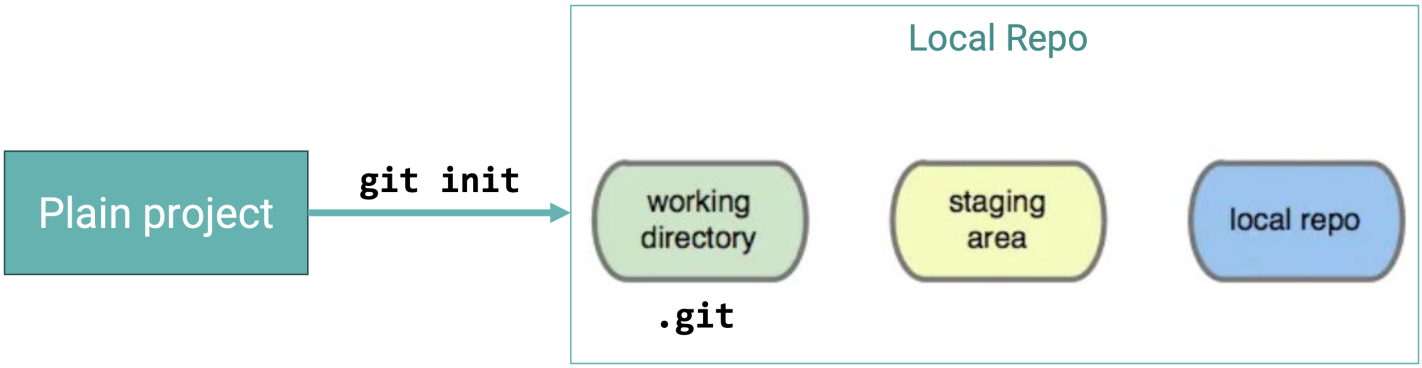
Git stores everything in its database not by file name but by the hash value of its contents.

Nearly Every Operation Is Local

Git Generally Only Adds Data

Git Architecture

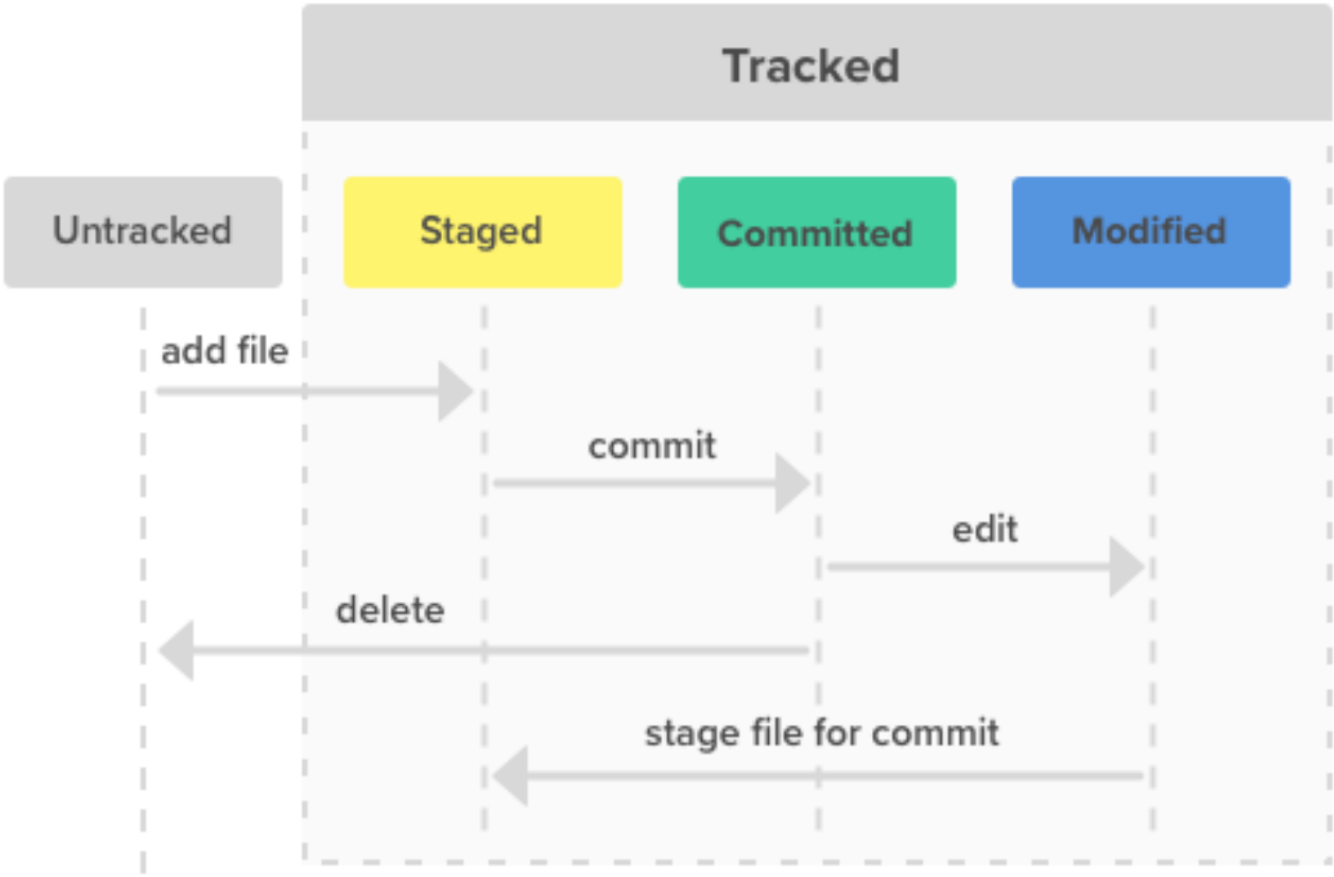
Local Repo -- Remote Repo

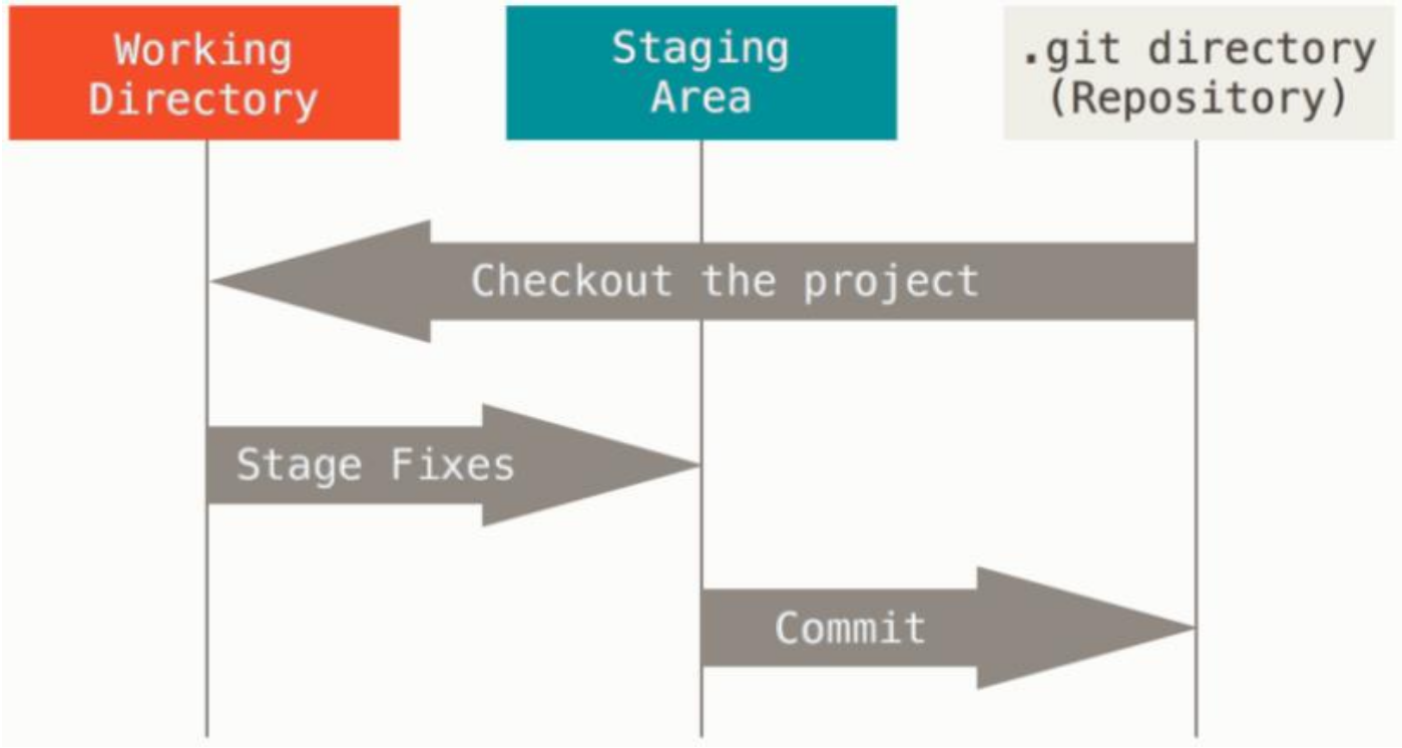


git add xxx: Untracked files ->stage files to be part of the next commit

Staging area (index): a file in your Git directory, that stores information about what will go into your next commit.

git commit

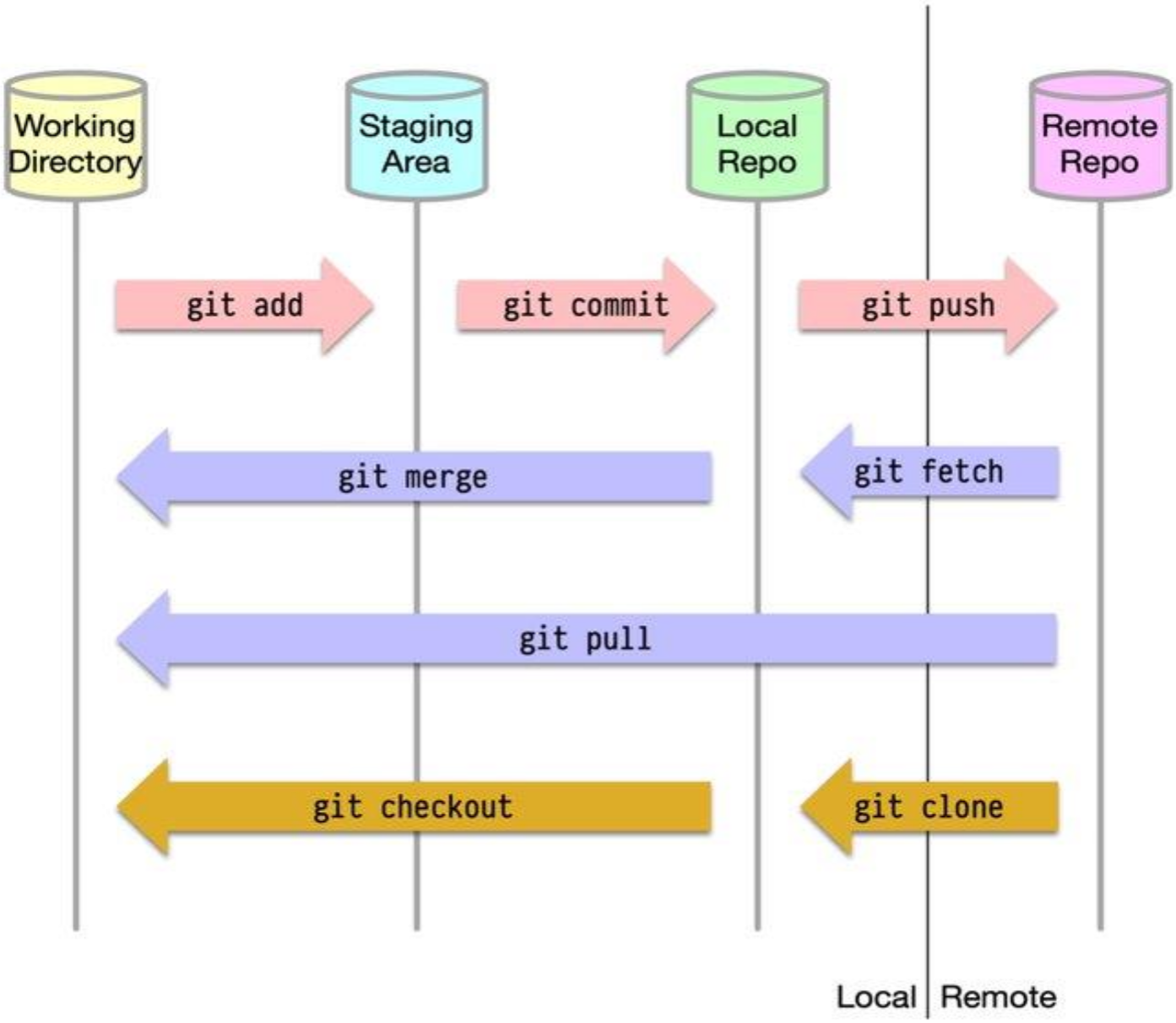




“link” a local repo and a remote repo

• Approach 1: Using git init and git remote add on local repo (origin is the default name)

• Approach 2: Using git clone to clone a remote repo, which also implicitly setups the remote for you



git clone: Cloning automatically creates a remote connection called "**origin**" pointing back to the original repository.

git push ： where **origin** is the default remote name and **main** is the default local branch name

git fetch： only downloads the data, don't merge

git pull： automatically fetches and then merges that remote branch into your current branch.

Git Internals

At the core of Git is a simple key-value data store.

• **Objects**: content of files, directories, commits, and tags, identified by SHA-1 hash, stored in `.git/objects/`

• **References**: A branch, remote branch or a tag, which is simply a pointer to an object, stored in plain text in `.git/refs/`

• **Index**: a staging area, stored as a binary file in `.git/index`.

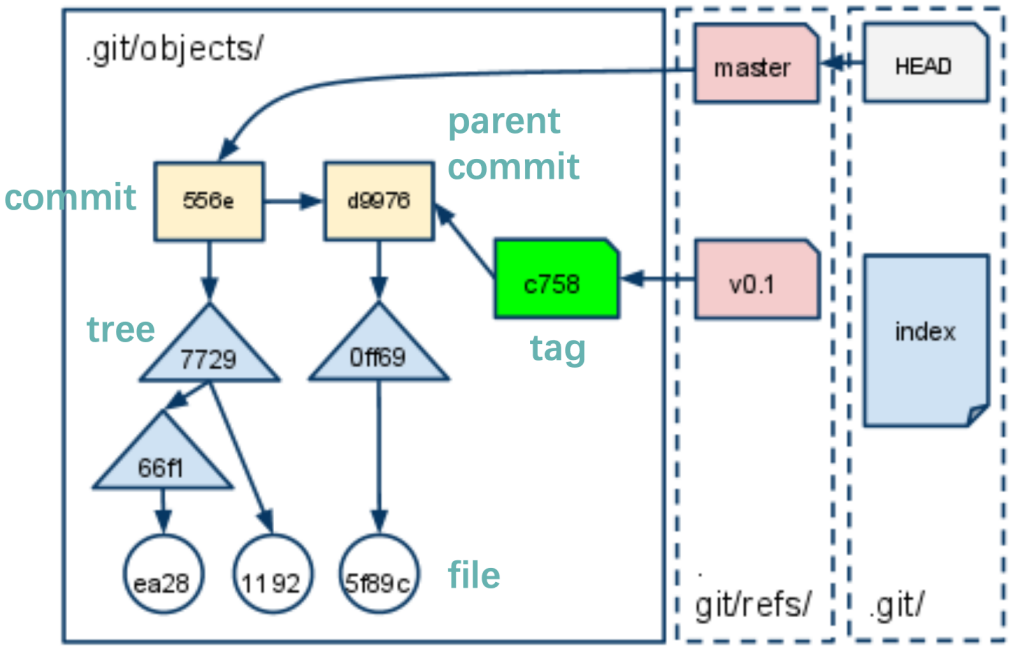
git objects

• **Blob**: file content, identified by a hash

• **Tree object**: list of pointers to blob (file), or tree (directory), identified by a hash

• **Commit object**: Reference to the top-level tree for the snapshot of the project at that point

• **Tag object**: name associated with a commit (+potential metadata)



Git Branching

"Killer feature" of Git

A branch in Git is simply a lightweight movable **pointer**

The default branch name in Git is **master/main**

git branch : create a branch（-d，删除）

git checkout ：switch branch (-b，创建并切换)

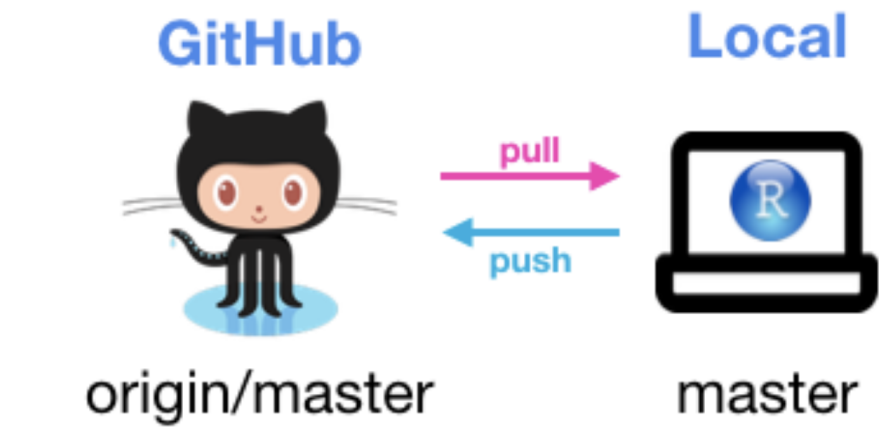
git push origin to push the new branch to remote

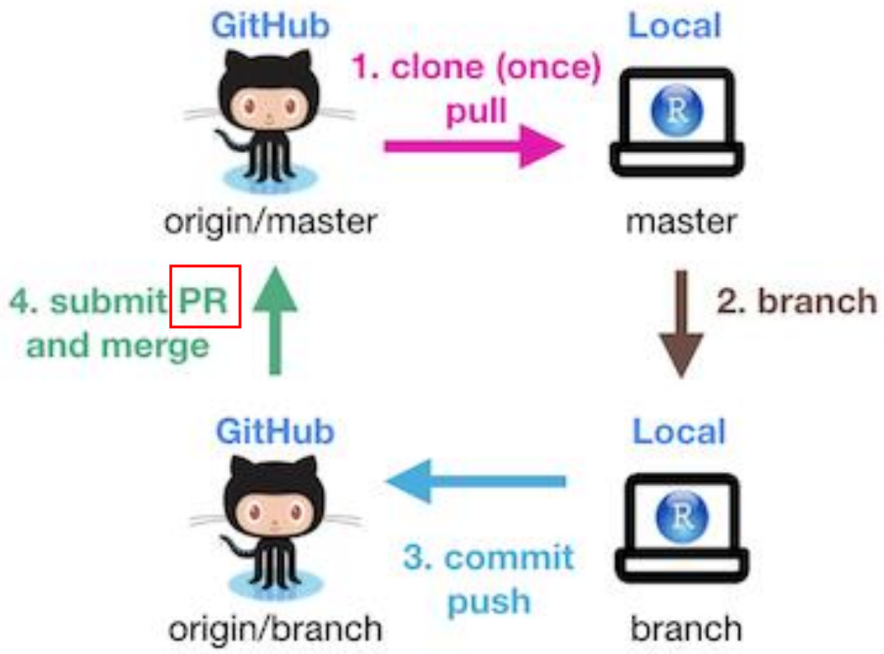
Fast-forward merge： Git simply moves the pointer forward. because there is no divergent work to merge together

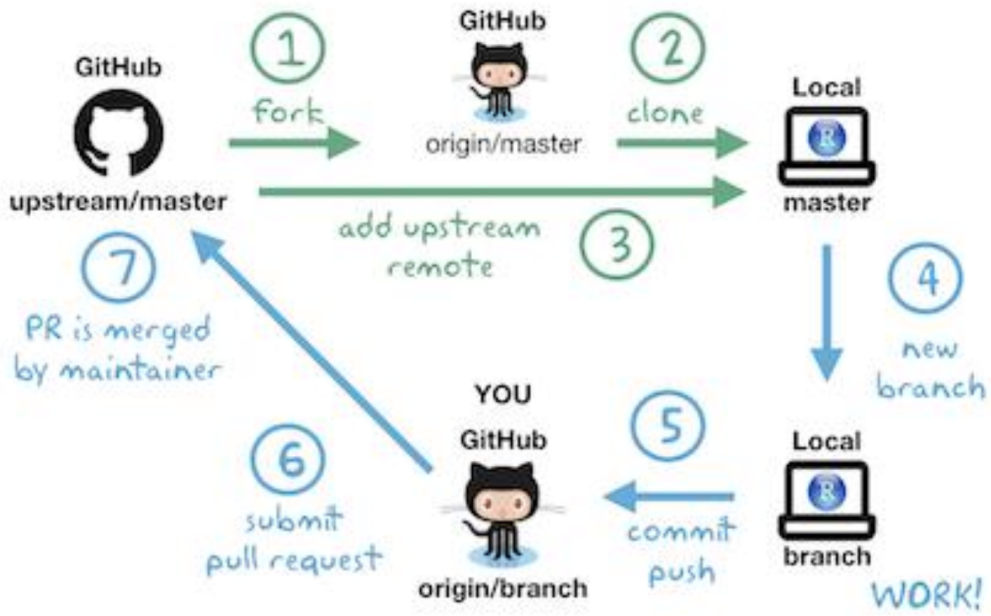
git merge：3-way merge: Git does a simple 3-way merge, using the two snapshots pointed to by the branch tips and the most recent common ancestor of the two.

git rebase: incorporating all the new commits in main, re-writes the project history

Github Workflow







git fork

• A Git fork operation will create a completely new copy of the target repository (**upstream** repo).

Key difference: how much control and independence you want over the codebase once you've copied it.

• Clone: a clone creates a linked copy that will continue to synchronize with the target repository.

• Fork: A fork creates a completely independent copy of Git repo, disconnecting the codebase from previous committers.

Best Practice

Don’t commit generated files or dependencies

Make clean, single-purpose commits

Commit early, commit often

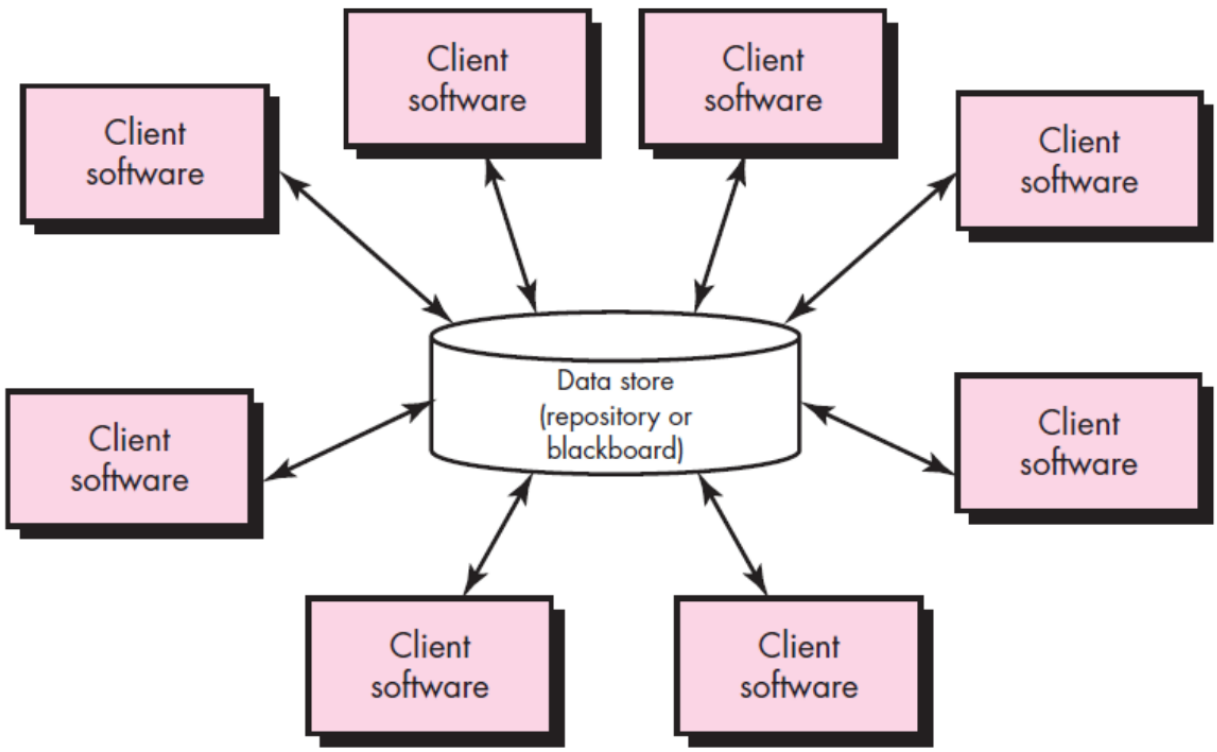
Write meaningful commit messages

Lecture 5

Architecture Style

Classic, Monolithic

• Data-centered architecture



以数据为中心的体系结构

• Data-flow architecture

pipe-and-filter

suitable for automated data analysis and transmission systems

Such systems contain a series of data analysis components, with almost no user interaction

• Call-and-return architecture

-- Main program/subprogram architecture

-- Remote procedure call architecture

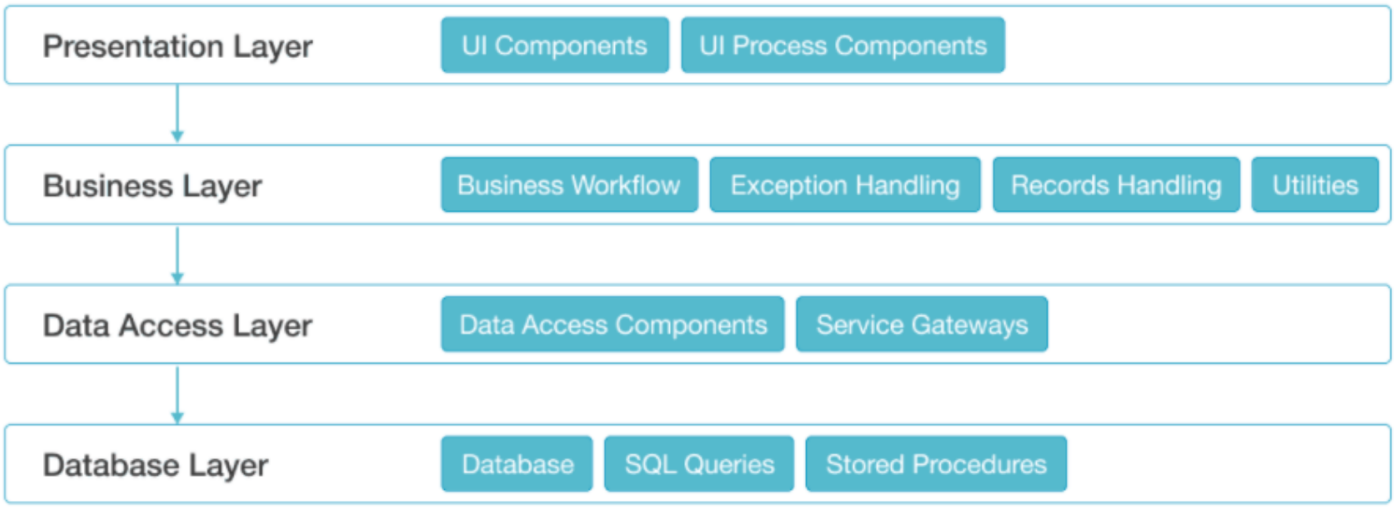
• Object-oriented architecture

Communication and coordination between components are accomplished via message passing

• Layered architecture

• At the outer layer, components service user interface operations.

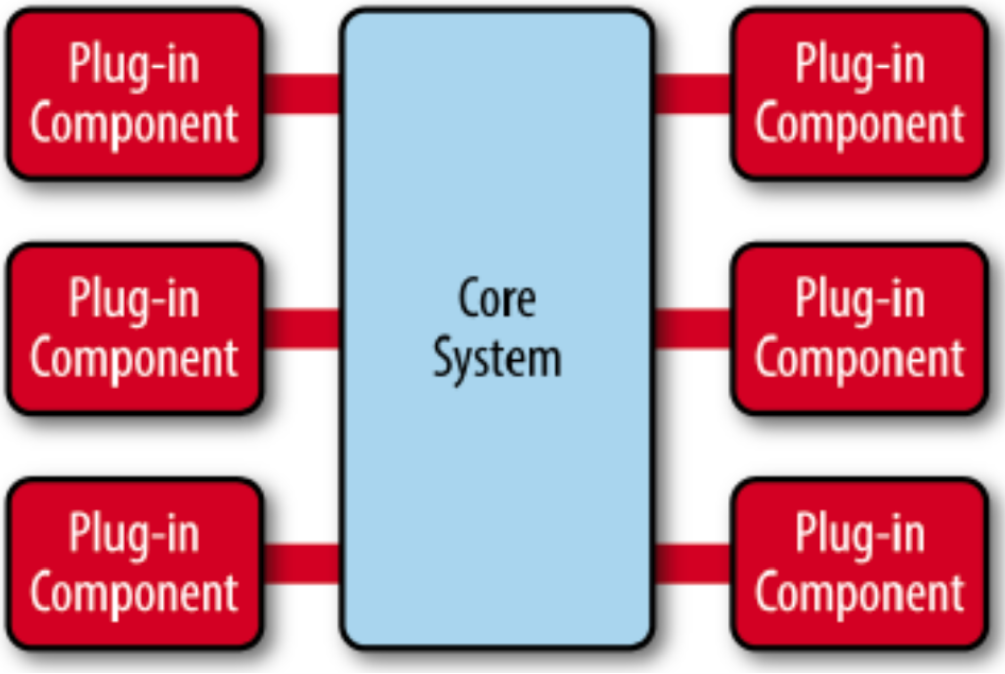
• At the inner layer, components perform operating system interfacing.



Service-based, Distributed, DevOps

• Microkernel architecture

• Also referred to as the plug-in architecture pattern



• Event-driven architecture

Core Components:

• Event Producer: Initiates and generates events.

• Event Consumer: Reacts to and processes events.

• Event channel/mediator/broker: orchestration

• Asynchronous and distributed

• Microservice architecture

a suite of small services, each running in its own process and communicating with lightweight mechanisms, often an HTTP resource API.

These services are built around business capabilities and independently deployable

微服务与单点服务的区别：

Service, Deployment, Decentralized Data Management, Scale, Team organization

Conway's Law "Any organization that designs a system will produce a design whose structure is a copy of the organization's communication structure."

Communication Mechaism

Synchronous

• RESTful API

Client-server Resources Stateless HTTP

• gRPC

binary message-based protocol

high-performance or data-heavy

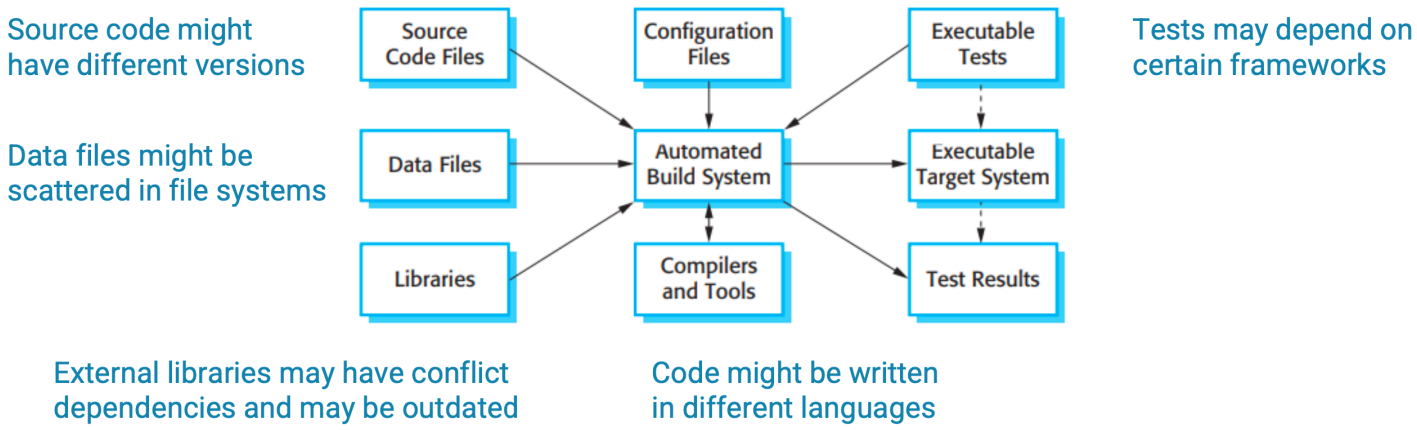
Asynchronous

• Messaging

Open source: RabbitMQ, Apache Kafka

Software UI Design

Lecture 6



Task-based Build Systems

In a task-based build system, the fundamental unit of work is the **task**

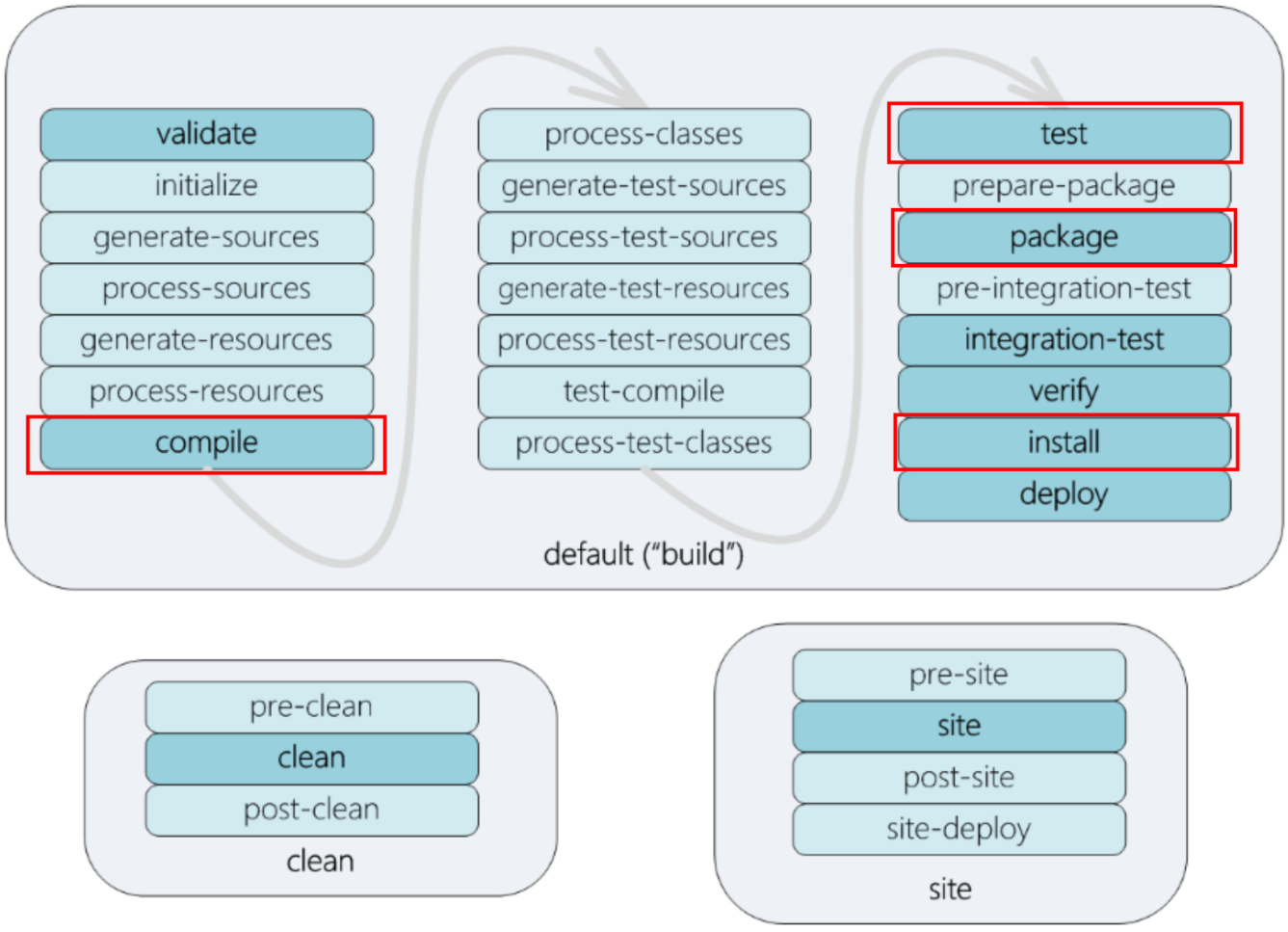
Most major build systems (e.g., Ant, Maven, Gradle, Grunt, and Rake), are task based.

buildfiles

ANT

task dependency

MAVEN



• Build **lifecycle**: an ordered list of build phases. Maven has 3 lifecycles.(default, clean, site)

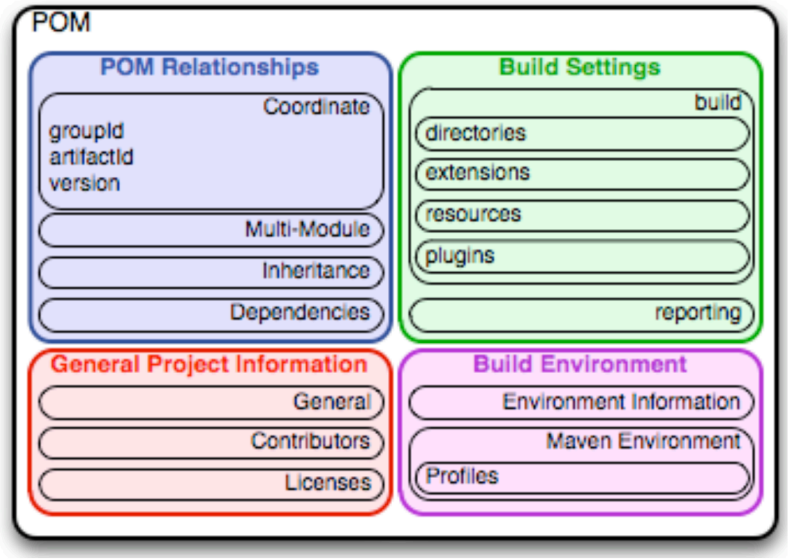
• Build **phase**: a set of build tasks

• **Plugin**: an artifact used for executing build tasks. A plugin provides one or more goals.

• **Goals**: used to execute build tasks. Smallest step of maven builds

• Goals of plugins are attached to one or more phases

POM



Drawback

Difficulty maintaining & debugging build scripts

Difficulty performing incremental builds & parallelism

Artifact-based Build Systems

the role of the build system is **producing artifacts**

• Engineers still need to tell the system **what** to build, but **how** to do the build would be left to the build system

Bazel

Buildfiles in artifact-based build systems like Bazel are **a declarative manifest** describing:

• A set of artifacts to build

• Their dependencies

• Limited set of configurations

Buildfiles define **targets**, every target has **name, srcs, deps**

Build Process

1. create a graph of dependencies among artifacts
2. determine the transitive dependencies
3. Build dependencies, in order

4. Build a final executable binary

Benefit

parallelism, incremental build

Task-based Build Systems-- Imperative approach

Artifact-based Build Systems-- Declarative approach

Build Artifacts

Semantic Versioning

版本号1.1.0

Format: {MAJOR}.{MINOR}.{PATCH}{Additional labels}

MAJOR(not backward compatible) MINOR(backward compatible)

package dependencies into the final artifacts?

Option 1: bundle everything together

Option 2: leave the dependencies out, upload to remote artifact(BINARY) repo

manage versions of artifacts

artifact(BINARY) repo

buildfile version controlled

Managing Dependencies

Internal vs External Dependency

Task vs Artifact Dependency

Dependency Scopes

Compile-time, Runtime, Test

Problems

A依赖B，B改了，A挂了

Solution:

Internal dependency: communicate with the owners of B

External dependency: A needs to be updated

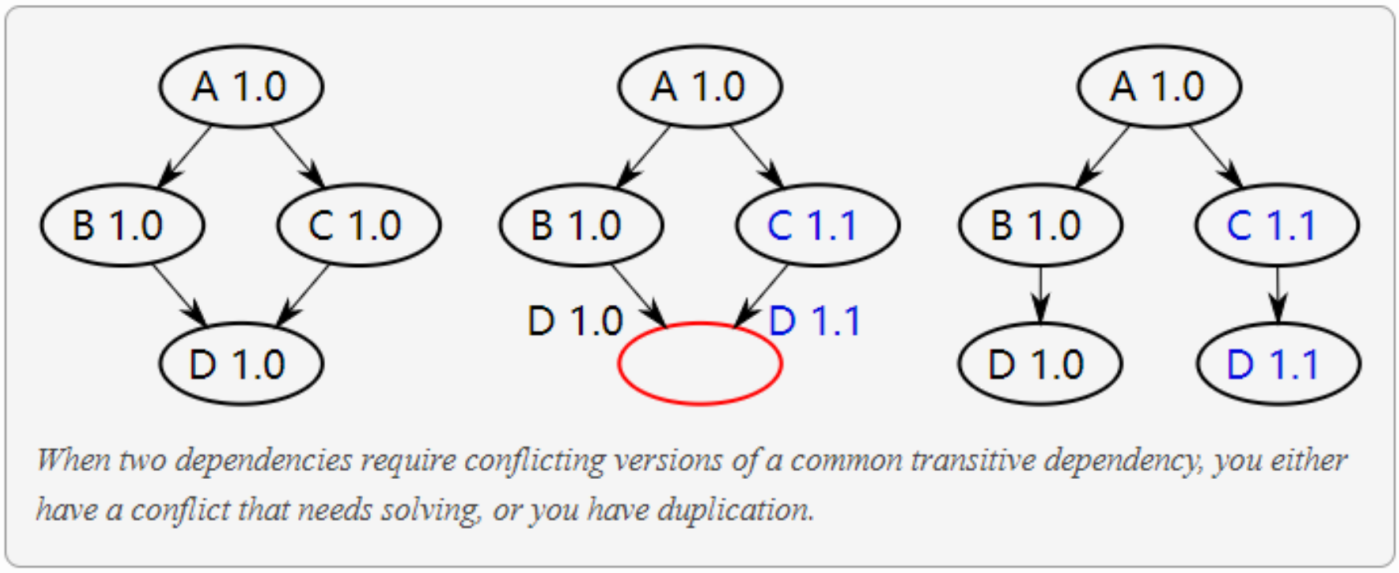
A依赖B，B依赖C（A其实需要C），C挂了，A也挂了

Solution: strict transitive dependencies

Blaze detects whether a target tries to reference a symbol without depending on it directly

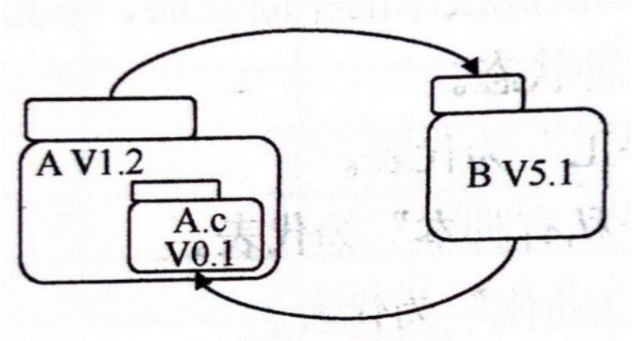
Diamond Dependency Issues

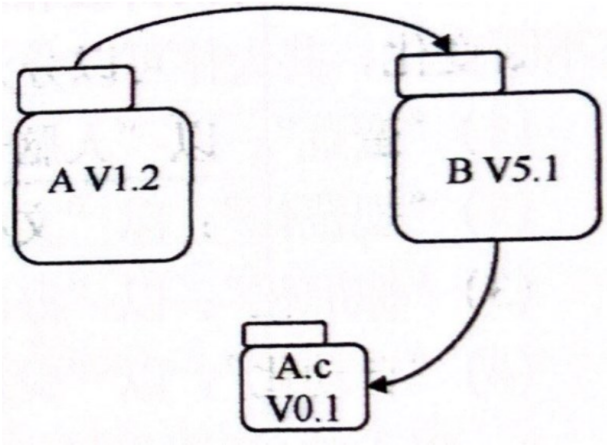
解决方法：升级、复制一个依赖



Circular Dependency

Solution: extract part of A to be another package





Lecture 7

Linters 代码检查

scans source code, especially useful for dynamically typed languages

Rule-based linters: CheckStyle, PMD

Metrics (for code complexity)

Lines of code

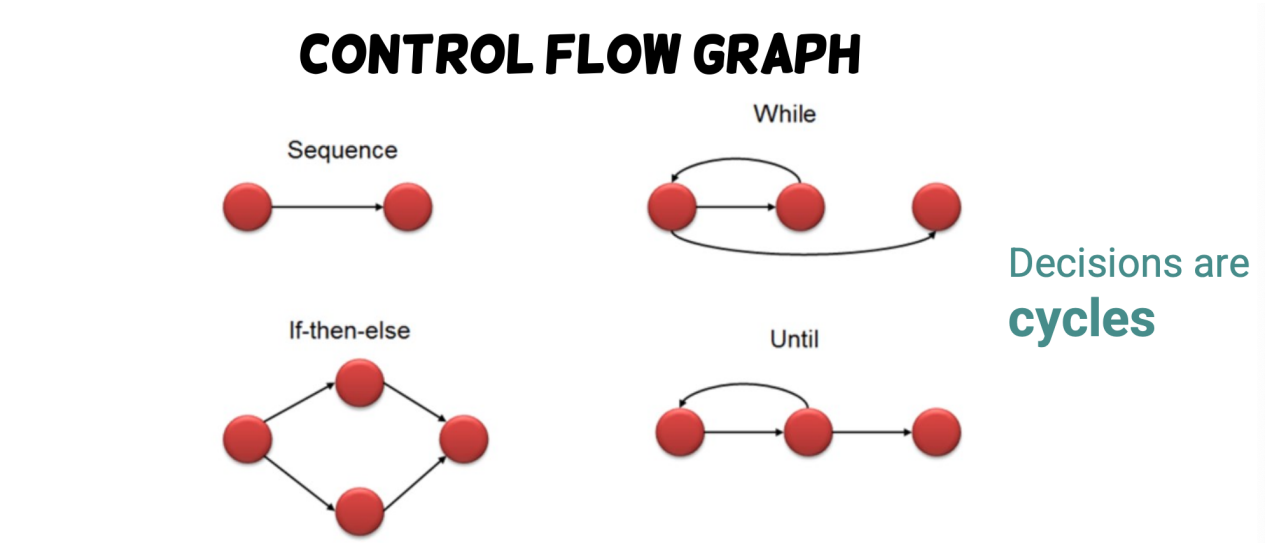
LoC needs to be **normalized** to be meaningful and comparable

Cyclomatic complexity

indicate the logic complexity

计算：using the control-flow graph (控制流图) of the program

范围：1-10比较好，大于10越来越差，但也有例外（switch语句；循环VS一堆if）



Approach 1: V (G) = P + 1

P: Number of branch nodes (e.g., if, for, while, case)

Approach 2: V(G) = E - N + 2

E: Number of edges

N: Number of nodes

OO Metrics

• **Weighted Methods per Class (WMC)** 衡量维护一个类的难度

• **Depth of Inheritance Tree (DIT)** 衡量一个类行为的可控性

• **Number of Children (NOC)** 孩子多，重要

• **Coupling between Object Classes (CBO)**

这里可能会引申到UML的考点。

doesn't care about the type or direction of a dependency

CBO = 0，没有依赖，应该去掉

CBD>4， 高耦合

• **Lack of Cohesion in Methods (LCOM)**

methods in a class that are not related

Considers all pairs of a class's methods:

• Q: In some pairs, both methods access at least one common field of the class

• P: In other pairs, the two methods do not share any common field accesses

LCOM = P - Q

A high LCOM could indicate that the design of the class is poor

• Response for a Class (RFC)

more methods can be invoked, the greater the complexity of the class

Code review

Type

• Modifications to existing code

• Entirely new code

• Automatically generated code

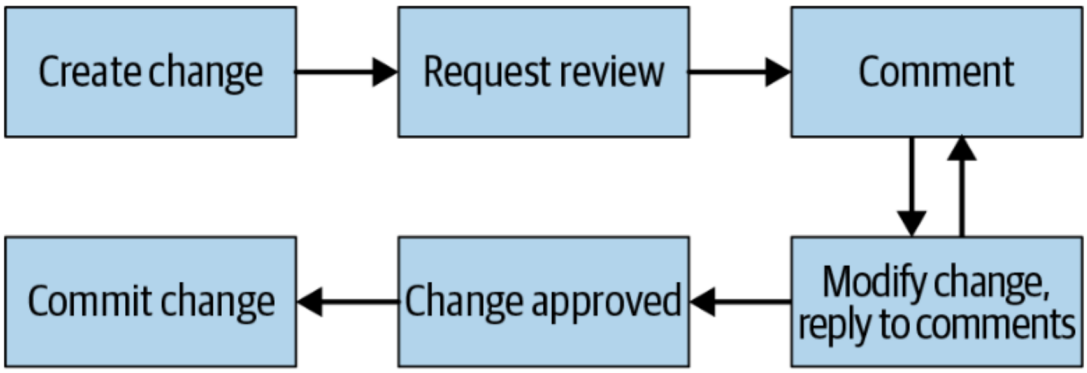
Benefits

Code correctness, readability, consistency, Knowledge sharing

Goals

Code Review Flow at Google

“looks good to me” (LGTM)



Lecture 8

好的测试方式：

Developer Driven Testing

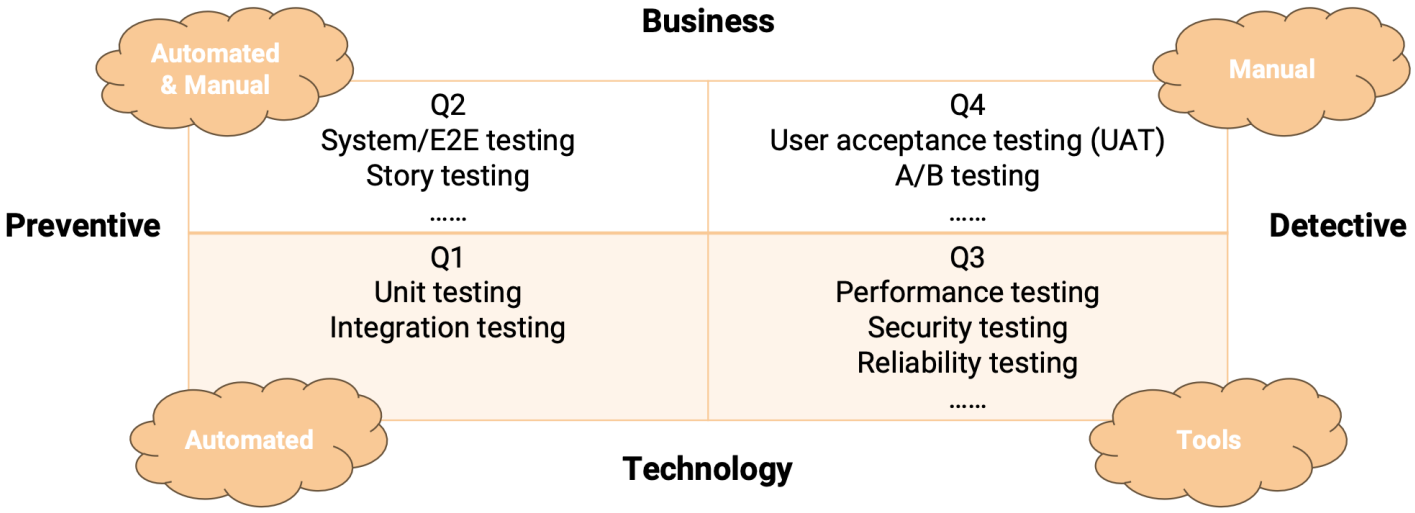
Automated Testing

Important concepts in software testing

Test Case (测试用例) VS. Test Suite(测试套件、测试集)

a test suite has multiple test cases

Test Input VS. Test Oracle (测试预言、测试判断准则,真值)



Unit Testing

测小的，methods, classes

work individually

Integration Testing

multiple components/modules working together(though interfaces)

System Testing

整体的测试

Test size

Small tests -- 单线程、进程，无网络

Medium tests -- 多进程、线程，localhost网络，多组件

Large tests -- 多台机器，整体

Performance Testing

load(期望负载)，stress（高负载），endurance（持续负载），spike/peak（突发负载），volume（数据容量负载）

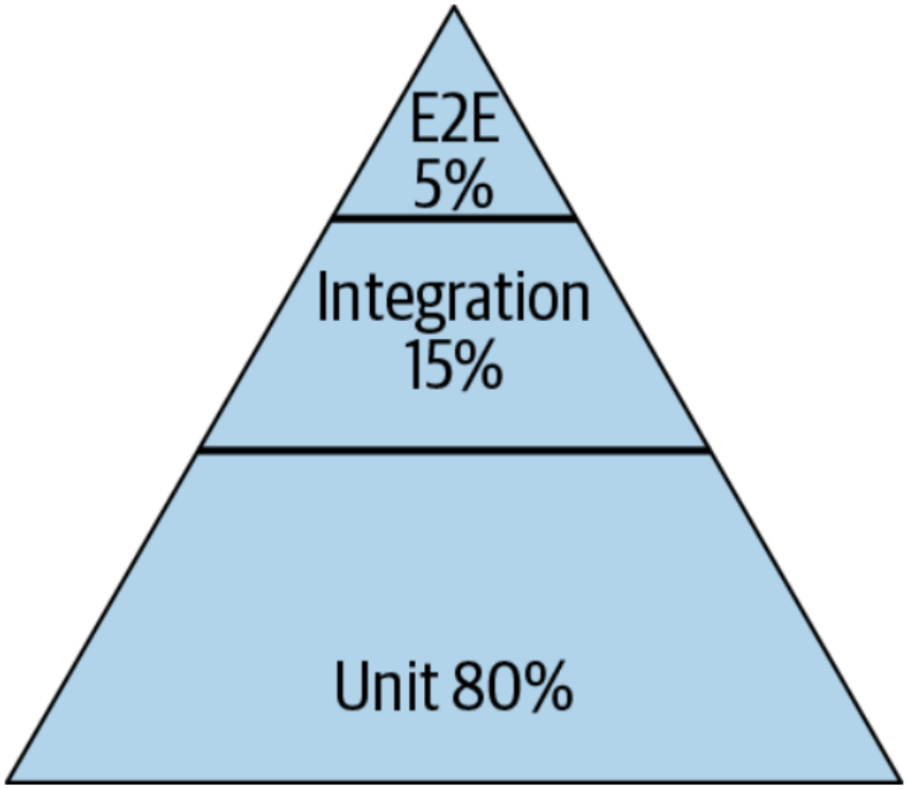
User Acceptance Testing (UAT)

UAT is performed by **business users** to verify that the application will meet the needs of the **end-user**, with scenarios and data representative of **actual usage** in the field

A/B Testing

一个变量，不同版本

Testing Pyramid



Google's test pyramid

Testing Techiques

Black box testing	White box testing
Based on software specification	Based on software code
Internal program structure is unknown	Internal program structure is known
Cover as much specified behaviors as possible	Cover as much coded behaviors as possible
Pros: simplicity, realistic results (simulate end users)	Pros: comprehensive testing, early defect detection
Cons: limited coverage, incomplete testing	Cons: requires technical expertise, expensive, limited real-world simulation.

White Box Testing - Code coverage

Statement coverage = Number of executed statements / Total number of statements *100

Branch coverage = (# of executed branches / # of total branches) x 100%

Condition coverage = (# of conditions that are both T and F / # total conditions) x 100%

Black Box Testing

Exhaustive Testing, 穷举，不现实

Random Testing，效果差

Partition Testing， divides data into equivalence data classes (等价类).

Equivalence Partition Hypothesis: If one condition/value in a partition passes all others will also pass. Likewise, if one condition in a partition fails, all other conditions in that partition will fail.

Boundary Values

Lecture 9

Maintainable Unit Tests

Ultimate Goal: Unchanging tests

Ideally, after a test is written, it never needs to change unless the requirements of the system under test change. Only changing the system's existing behavior would require the updates to existing tests,

Good Practice 1: Test Via Public APIs

Bad Practice:

Test the internal (private) implementation logic directly.

Test the internal states of the database.

Good Practice 2: Test Behaviors, Not Methods

A behavior-driven test: rather than writing a test for each method, write a test for each behavior.

Integration tests: Test Doubles

test suite可能很难写，很难跑，所以需要 Test Doubles模仿真实对象的行为或者被测系统(SUT)中的组件。

Fakes

behaves "naturally", but is not "real"

一些组件"虚假"的简单实现，比如内存里模拟的数据库

Stubs

behaves "unnaturally"

specific inputs with specific outputs 处理 specific state

Mocks

A mock is similar to a stub, but with verification added in.

A mocking framework is a software library that makes it easier to create test doubles within tests

Mockito, a mocking framework for Java.

UI Testing

test in the same way an end user would

很强力，因为把整个测试金字塔都测了

测 Consistency， Spelling， Typography， Behavior of interactive elements， Functional validation，Adaptability

测试工具：Selenium

Lecture 10

Software documentation

Types

External Software Documentation

End-user documentation

Enterprise user documentation(给部署的人看的)

Just-in-time documentation（如FAQ）

Internal Software Documentation

Administrative documentation

Developer documentation

Good Practices

Self-documenting (or self-describing) code

Code comments enhance readability

Should answer WHY, instead of WHAT，解释、补充代码原理，而不是复述代码

• Describe the design decisions to a class

• Describe the limitations of an implementation

• Describe usage assumptions of APIs

• Describe the purpose and intents of each file

Tools - JavaDoc

Comment Syntax

```
// 一行

/*
 * 多行
 */

/**
 * Javadoc
 */
```

用于 class, method, or field

JAVADOC TAGS

@author	A person who made significant contribution to the code. Applied only at the class, package, or overview level. Not included in Javadoc output. It's not recommended to include this tag since authorship changes often.
@param	A parameter that the method or constructor accepts. Write the description like this: @param count Sets the number of widgets you want included.
@deprecated	Lets users know the class or method is no longer used. This tag will be positioned in a prominent way in the Javadoc. Accompany it with a @see or {@link} tag as well.
@return	What the method returns.

@see	Creates a see also list. Use {@link} for the content to be linked.
{@link}	Used to create links to other classes or methods. Example: {@link Foo#bar} links to the method `bar` that belongs to the class `Foo`. To link to the method in the same class, just include #bar.
@since 2.0	The version since the feature was added.
@throws	The kind of exception the method throws. Note that your code must indicate an exception thrown in order for this tag to validate. Otherwise Javadoc will produce an error. @exception is an alternative tag.
@Override	performs a check to see if the method is an override. used with interfaces and abstract classes.

一些别的文档工具

PyDoc for Python

JSDoc for JS

Sphinx, Swagger, Gitbook

一些擦边Doc

tests，git commit msg

Documentation as Code

Documentation should be integrated into the codebase and workflow

例子：g3doc

Design1: docs lives with code

Design2: use markdown

好处：docs有版本控制，渲染精美，与代码链接紧密

Lecture 11

Continuous Integration

The fundamental goal of CI is to automatically catch problematic changes as early as possible.

CI process - preparation

• Version control

• Automated build

• Automated quality control

• Automated doc generation

• Team consensus

CI process - commit phase

1. 拉代码
2. 改代码，确保过本地测试，这个过程中别人可能也push代码远程库里
3. 拉人家的代码
4. 整合，确保过测试
5. push
6. central system说ok

CI process - CI servers

Typical **tasks** executed in a CI server

• Compilation

• Linters & code analysis

• Unit testing and integration testing

• Test coverage analysis

• Artifact generation

• Security scans

• Notification and reporting

• Deployment to testing/production environment (later)

Jenkins是一个不错的开源CI server

CI foundation：Buildfile As code

CI Best Practice:

1. 每天push
2. 每次commit自动构建和测试
3. 如果寄了赶紧修

Continuous Delivery and Deployment

Environment

• Development Environment (开发环境)

• Staging Environment (类生产环境)

• Production Environment (生产环境)

Continuous delivery 是CI的拓展，让更新自动加入类生产/生产环境，使得可以高频发布

Continuous deployment 是 Continuous delivery的拓展，直接发布给用户，没有人为干涉

CI/CD Pipeline

Common components of a deployment pipeline

• Version control

• Artifact repositories

• Build automation tools

• Automated testing tools

• Continuous integration (CI) servers

• Deployment tools

Required for CI/CD pipelines

• Version control

    Source code

    Buildfile (as code)

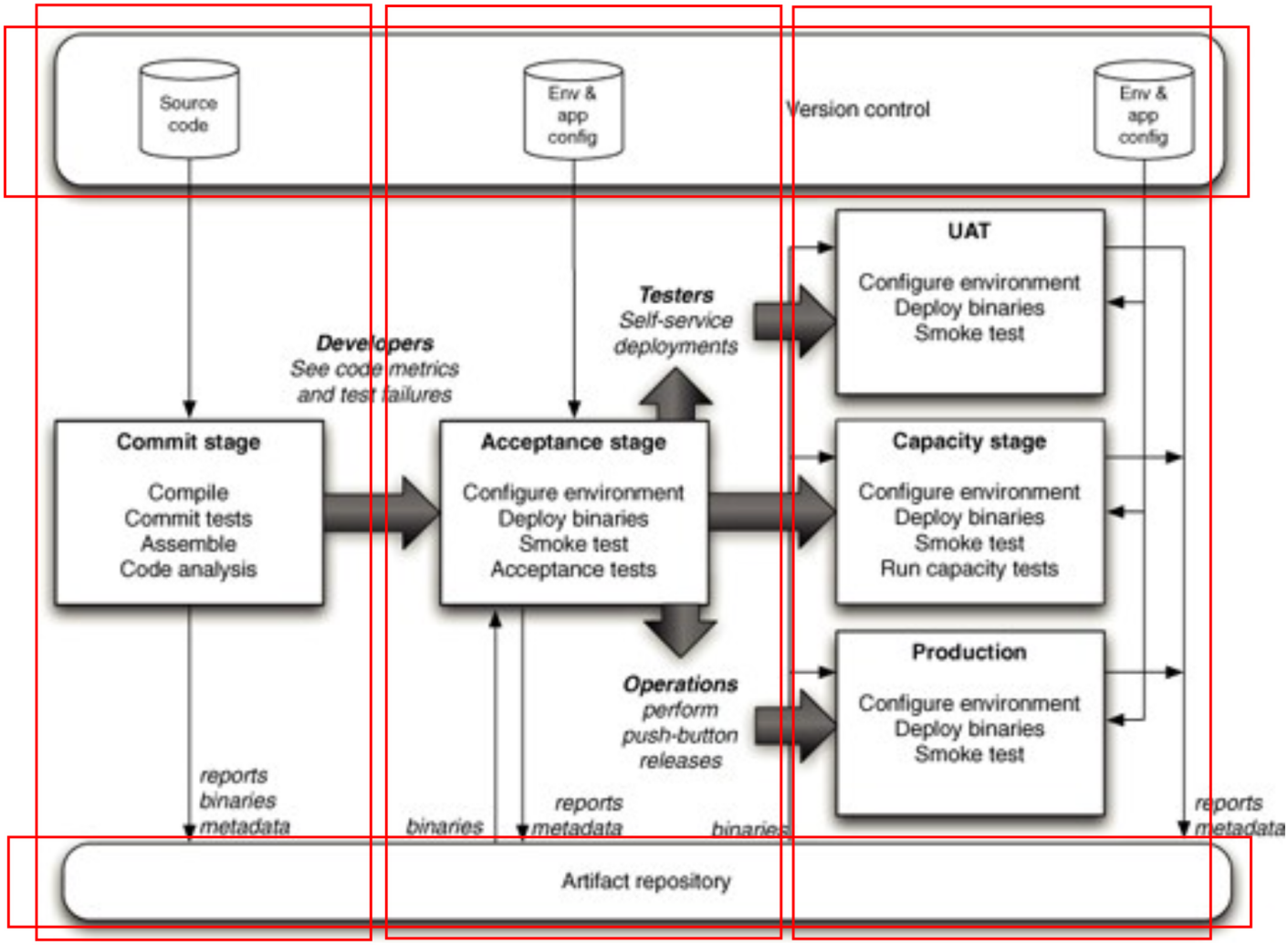
    Documentation (as code)

    Infrastructure (as code)

• Artifact repositories

    Libraries, plugins, etc.

Pipeline Breakdown



Stage 1

1. Commit to version control
2. Trigger the build
3. Generated binaries stored in the artifact repository

Stage 2

Run automated acceptance tests

Can be executed on the staging environment

Configurations for the environment are version controlled just like source code (infrastructure as code)

Smoke test: It consists of a minimal set of tests run on each build to test major functionalities.

Stage 3

deploys the verified binary version to the production environment

可选：deployed to a staging environment for tool-aided non-functional tests or manual UAT by testers.

CI/CD Vs. Rapid Release

部署是部署进生产、类生产环境，发布是让end user可用

有时候两者概念类似

Deployment Strategy

Low-risk release (低风险发布)

• Blue-Green Deployment (蓝绿部署)

两个环境，蓝的是老的，绿的是新的，通过一个路由来定向请求，新的如果有问题可以及时定向回老的。

• Benefits

Minimal downtime

Rollback is easy and fast

• Drawbacks

Expensive: a redundant infrastructure

Data consistency and integrity problems

• Rolling Deployment (滚动部署)

用于 server cluster

逐渐更新所有的系统，部分下线，滚动更新

• Benefits:

Less infrastructure cost and maintenance efforts

Early detection of problems

• Drawbacks:

A significant latency between the moment you start deploying the new version and the moment it is all live.

If problems slip past initial checks, they'll propagate as more servers are updated, making rollback difficult

Can't control which users get the new version

• Canary/Greyscale Release (金丝雀发布/灰度发布)

简单来说，内测

a limited selection of users with different location or device type

• Benefits

Test the new version with real users

Identify bugs or performance issues before releasing to a wider audience. In case of failure, only a small number of users is affected

Rollback is simple and quick

• Drawbacks

Complex implementation & infrastructure mechanism (e.g., API compatibility, DB integrity, user verification, etc.)

• Feature toggle (功能开关)

Branching Strategy

• Trunk-based dev & release (主干开发，主干发布)

directly used for deployment as release

• Benefits

Less management efforts for branches

• Drawbacks

Unfinished work affects release

Hard to merge branches back in case of high-frequency deployment & rapid release

• Trunk-based dev & branch-based release (主干开发，分支发布)

some very limited work happens in the release branch in order to fully make it release-ready

all dev work still happens on the trunk

如果发布的有问题，在主干上修了push到发布分支上去

Benefits: devs stay on trunk and are less affected by release; release won't be affected by unfinished work on trunk

Drawbacks: "release branch hell" 过量了

• Branch-based dev & trunk-based release (分支开发，主干发布)

Feature branch开发新功能，Develop branch 开发，main branch发布

Benefits

• Independent development on branches

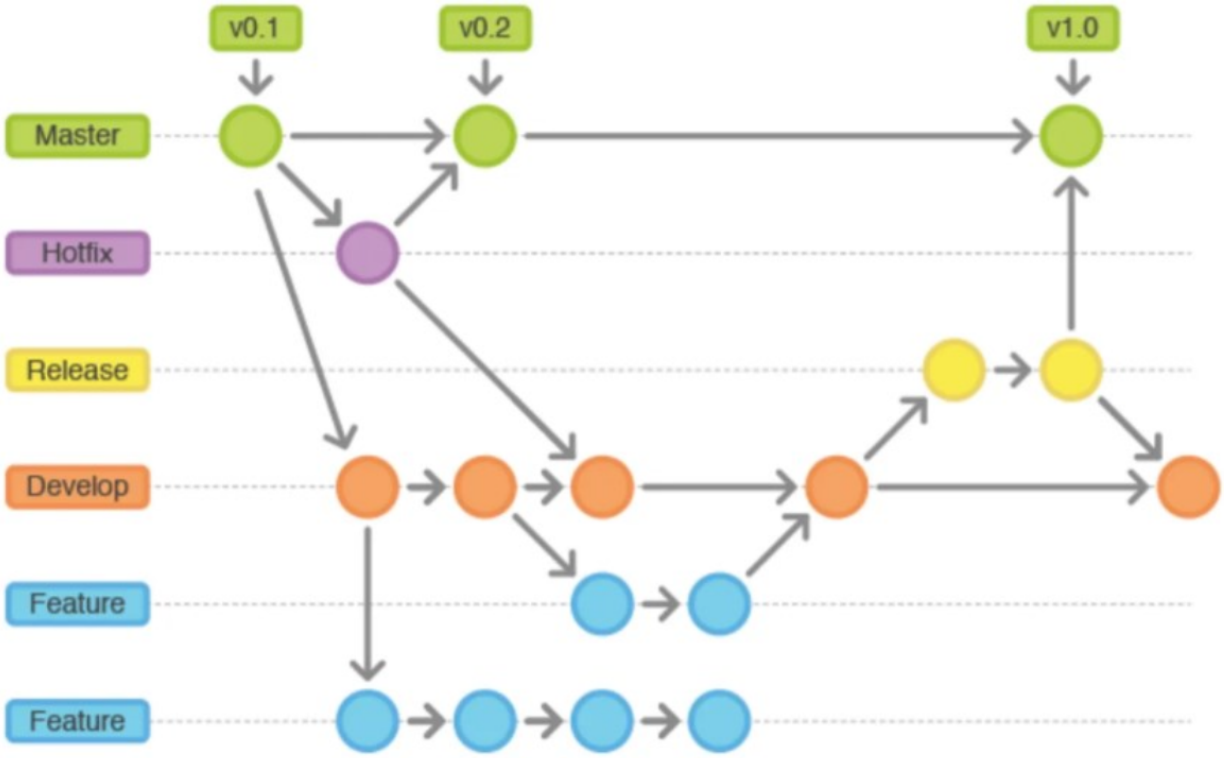
• Teams can choose which features to be released.

• If bugs occur, fixes can be done directly on the main branch or on a new hotfix branch, without affecting other feature branches.

Drawbacks

• Merge efforts on many branches

Gitflow branching strategy



Master: branch for official releases

Develop: branch for feature integration

Feature branch: pull from develop branch; merge back to develop branch once the feature is done.

Release: branch for pre-release

• only bug fixes, documentation, etc.

• Not used for new feature development.

• Merged back to master and develop branch once passed QA(and optionally, Alpha & Beta releases)

Hotfix: if v0.1 has a bug, create a hotfix branch for quick fixing, then release as patch v0.2

should also be merged to develop branch

Lecture 12

Deploy

Solution 1: automate the deployment through a shell script, which should be reusable, robust, easy-to-maintain

Solution 2: automate the monitoring of server status, exporting key healthy metrics, detecting anomalies.

Solution 3: (autohealing) automate the anomaly handling, e.g., kill and reboot the process

Solution 4: Isolation, a guarantee that a process can safely proceed without being disturbed by other processes.

Solution 5: (automated scheduling) a central service that knows the complete list of machines available to it and can—on demand—pick unoccupied machines and automatically deploy the binary to those machines.

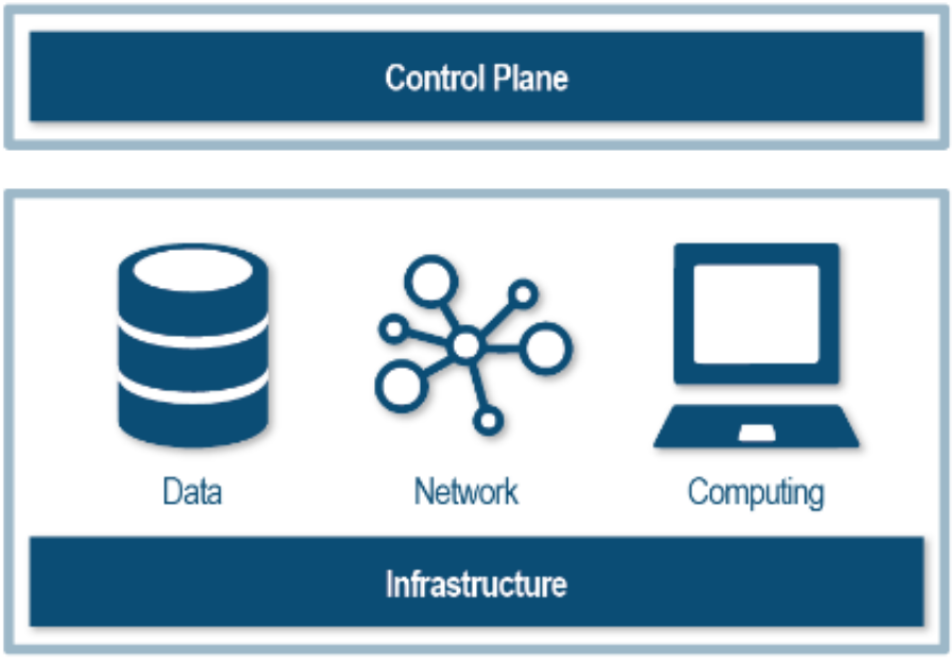
Solution 5: (autoscaling) automate the settings of configuration parameters.

Scale up (Vertical scaling) 变大

Scale out (Horizontal scaling) 变多

Load Balancing，Configuration，Monitoring/Troubleshooting，Maintenance

Infrastructure



computing 包含 hardware, software

Monitoring and control plane: 3 forms of infrastructure monitoring: hardware, network, and application monitoring.

On-premise (本地部署)

Cloud-native Applications

特性

• **Processes:** DevOps & CI/CD

• **Architecture:** Microservices

• **Deployment:** Containers

• **Practice:** Infrastructure as code

Deploy Pattern

Language-specific package pattern

问题：

Lack of encapsulation of the tech stack

Lack of isolation

Hard to constrain the resources consumed by a service instance

Manually determine the resources assigned to service instances

Virtual machine

问题：

Less-efficient resource utilization

Slow deployment

Containers

isolate expose services

• At build-time, the deployment pipeline uses a container image-building tool, which reads the service's code and a description of the image, to create the container image and stores it in a registry

• At runtime, the container image is pulled from the registry. The service consists of multiple containers instantiated from that image.

• Containers typically run on virtual machines. A single VM will usually run multiple containers.

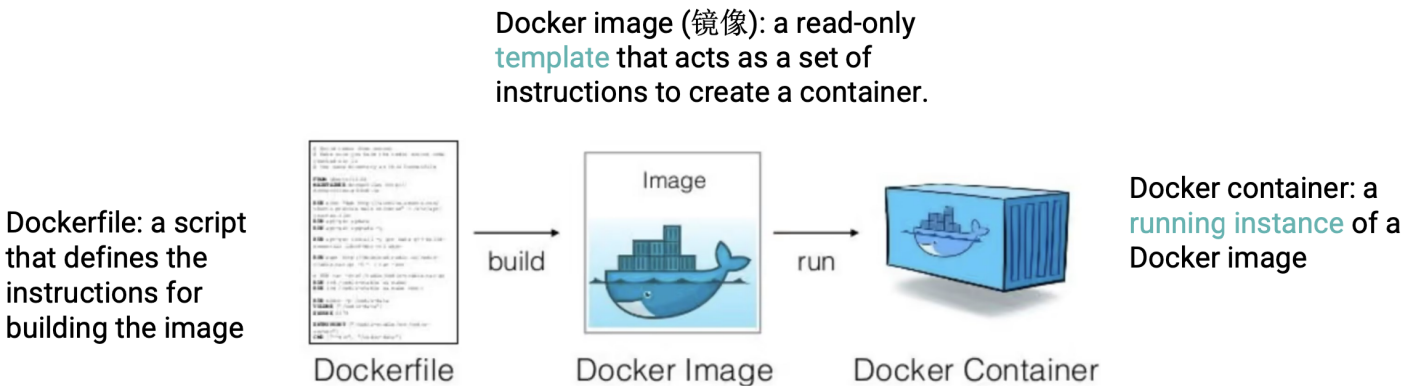
问题：

Managing container images

Managing the container infrastructures

解决方法：需要编排

Docker



Images (read-only) can be layered on top of one another, specified by Dockerfile

The top layer（the container layer）has read-write permissions, and all the remaining layers have read-only permissions

push the Docker image to a registry

docker run command pulls the image from the registry if necessary. It then creates and starts the container

container orchestration (编排)

automate and manage tasks such as:

• Deployment

• Configuration

• Scheduling

• Resource allocation

• Container availability

• Load balancing and traffic routing

• Scaling or removing containers based on balancing workloads across your infrastructure

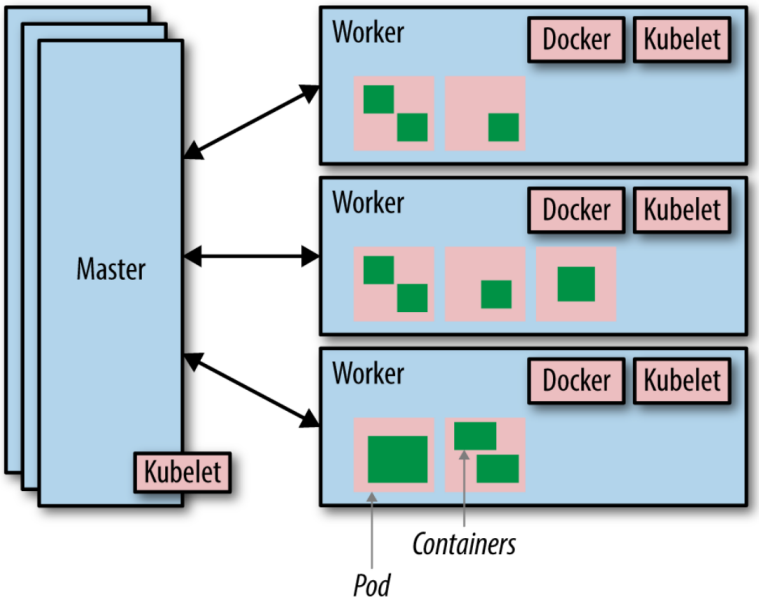
• Monitoring container health

• Keeping interactions between containers secure

Tools: Docker Swarm and Kubernetes (K8s)

Kubernetes Architecture

Cluster: A **control plane** (master) and one or more compute machines, or **worker nodes**.



Worker node

• **Pod:** smallest deployable unit in k8s.

A pod hosts one or more containers

A pod provides shared storage and networking for its containers

• **Kubelet:** a system service (daemon) that runs on a worker node

Managing and monitoring the state of the pods and containers on that node.

Communicate with the master to receive instructions and report status updates.

• **Container runtime (Docker):** runs the containers on worker nodes

Pulling container images from registry

Starting and stopping containers

Managing container resources

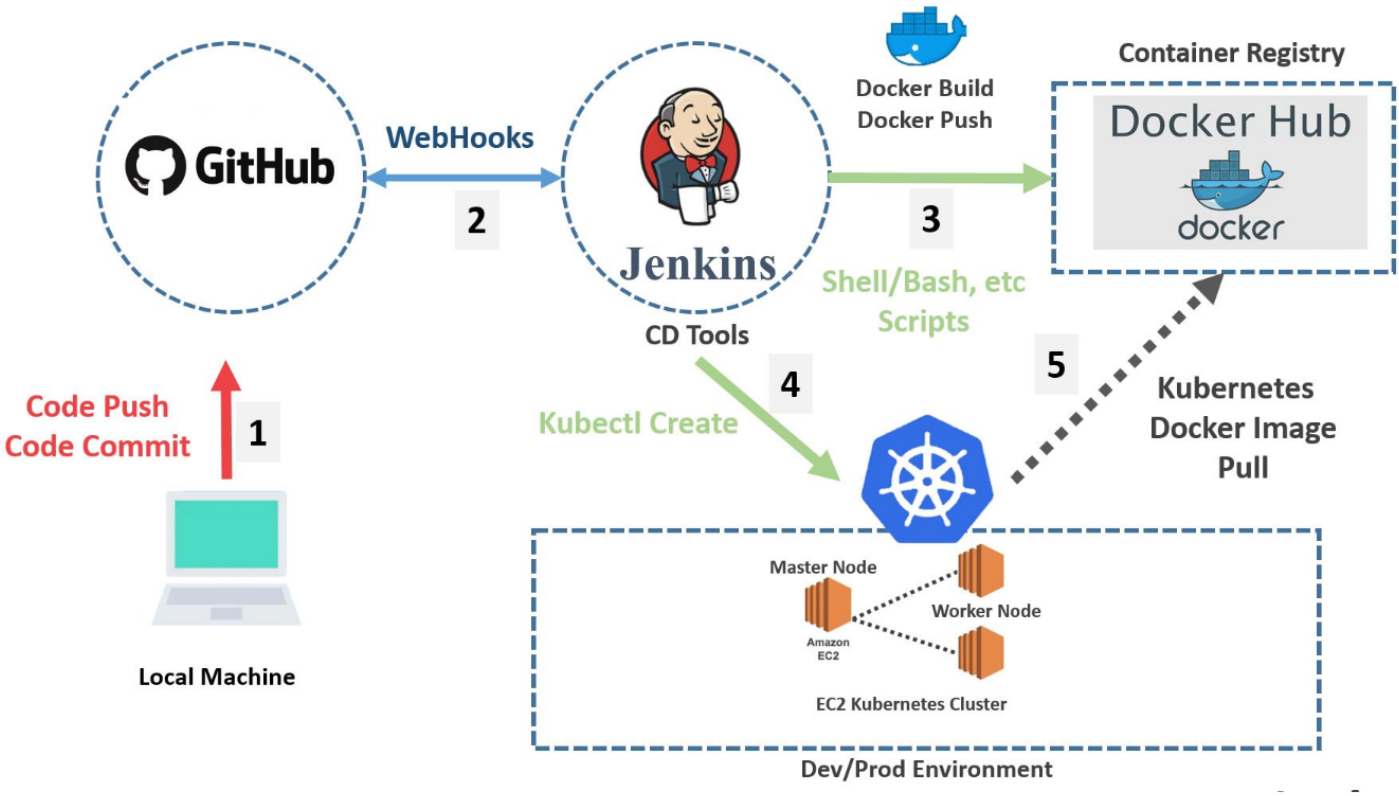
• **Kube-proxy:** network proxy on workers

Routing traffic to the correct pods

Load balancing

A k8s deployment YAML file is a configuration file written in YAML that defines the desired state of a Kubernetes Deployment.

A typical Jenkins Pipeline



K8s Deployment Strategy

Deployment Strategy	Available in K8s out of the box?	Pros
Recreate	Yes	Fast and consistent.
Rolling	Yes	Minimizes downtime, provides security guarantees.
Blue/Green	No	No downtime and low risk, easy to switch traffic back to the current working version in case of issues.
Canary	No	Seamless to users, makes it possible to evaluate a new version and get user inputs with low risk.

Infrastructure as code (IaC)

Declarative approach: focuses on **what** the eventual target configuration should be.

Imperative approach: focuses on **how** the infrastructure is to be changed.

Configurations

• Environment config: IP, port, OS, version, etc.

• Application config: versions and configs for dependent software or services, caches, token, etc.

• Business config: discount, feature toggles, etc.

Benefits

• Scale and agility: automation

• Self-documenting

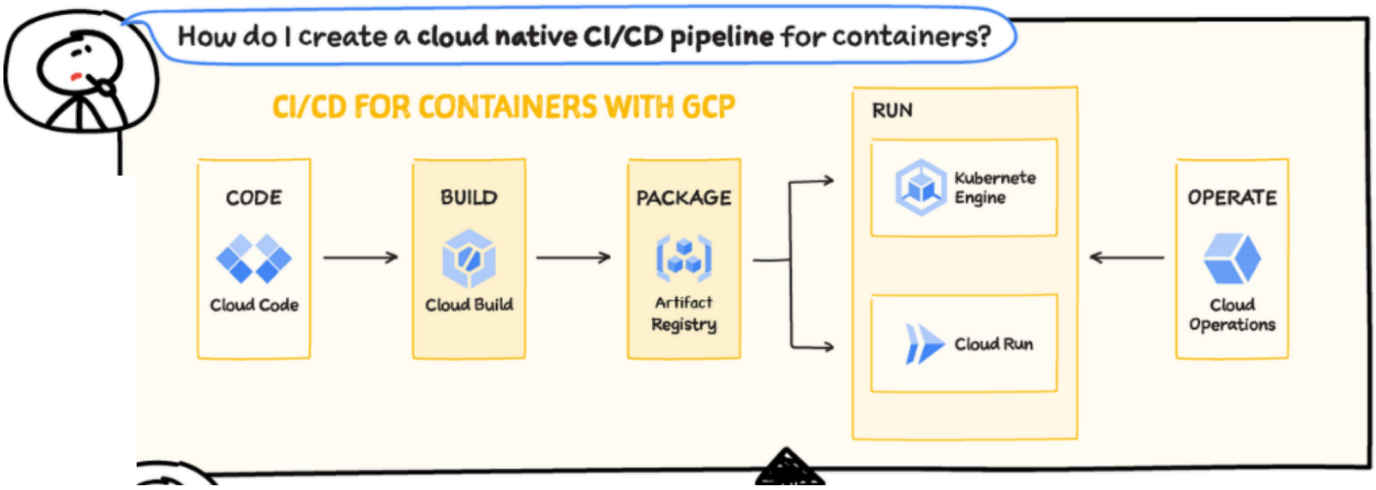
• Auditability: everything is trackable

Immutable Infrastructure

要升级就要摧毁老的，用新的

Cloud-Native

CLOUD-NATIVE CI/CD PIPELINE ON GOOGLE CLOUD



Lecture 13

为什么软件进化开销巨大?

needs to migrate to new platforms

complexity grows

get newcomers familiar

Legacy systems

还在用的老系统，不太好动的那种

Why not simply replace the legacy code?

一言以蔽之：expensive & risky

Lack of specification or documentation

Important business rules may be implicitly embedded

maintenance degrades/decays the system structure

using obsolete programming languages, old techniques, and adapted to older, slower hardware

Data out of date, inaccurate, incomplete

Decisions for Legacy System

1. **Abandon** the system completely -- **Low quality, low business value**; **High quality, low business value**
2. Leave the system unchanged and continue with **regular maintenance** -- **High quality, high business value**; **High quality, low business value**
3. **Reengineer** the system to improve its maintainability -- **Low quality, high business value**

4. **Replace** part or all of the system with a new system -- **Low quality, high business value**

Deprecation

demonstrably **obsolete** and a **replacement** exists

How to deprecate (ELEGANTLY)

Dependency Discovery 看看谁在用

Warning flags 告诉他们要废弃了

Sunset period (sunsetting) 提供一个平滑过渡

Migration & Testing

Maintainability

分类

Fault repairs to fix bugs and vulnerabilities

难度由易到难：coding errors, design errors, requirements errors

Environmental adaptation to adapt the software to new platforms and environments.

Functionality addition to add new features and to support new requirements.

指标

都是越高越难修

Number of requests for corrective maintenance

Average time required for impact analysis

Average time taken to implement a change request

TECHNICAL DEBT (技术债) 优先考虑速度而不是设计

Reengineering

3 stages

• Reverse engineering

Code -> design -> requirements

-- Goal is to understand how it was built

• System transformation

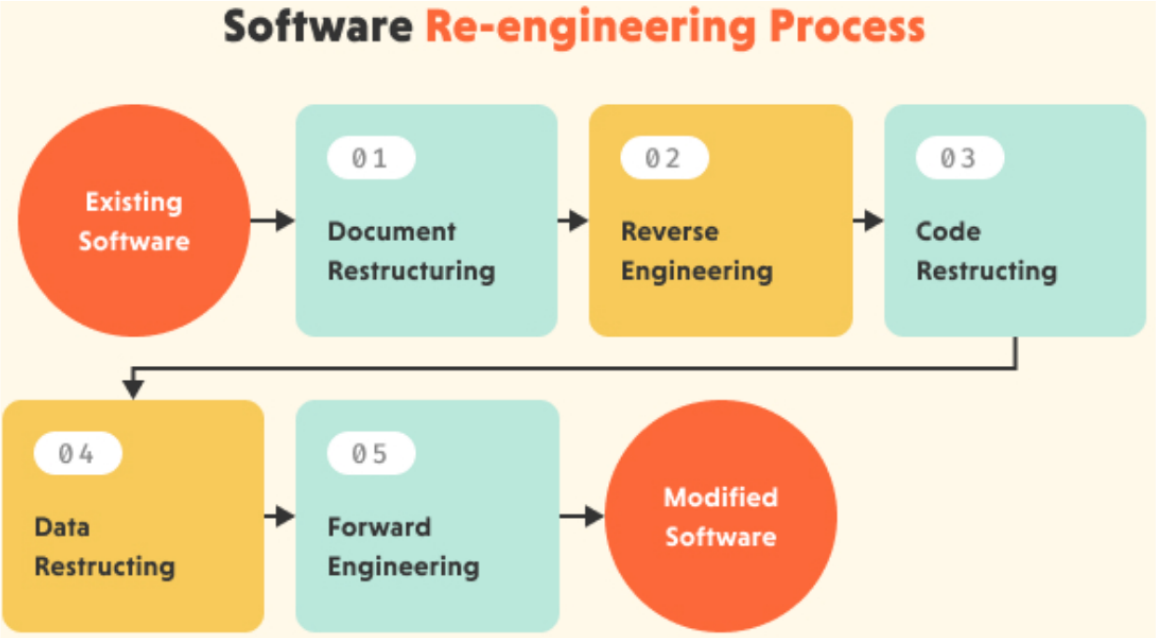
-- Refactoring - methods and classes层次

-- Rearchitecting - modules and components层次

-- Rewriting - 最高层次

• Forward engineering

Requirements -> design -> code



Refactoring

什么时候重构?

事不过三；增加功能；修复bug；进行代码审查时

重构什么？ - Code Smells

Bloaters

代码臃肿

Long method 解决方法 extract method

Long parameter list 解决方法 Replace parameter with method call, etc

OO Abusers

refused bequest 拒绝的遗产 解决方法 Replace inheritance with delegation；pushdown methods/fields

子类继承了父类一些不需要的行为

Liskov Substitution Principle

> • Objects of subclasses should behave in the same way as the objects of superclass.
>
> • Objects of a superclass should be replaceable with objects of its subclasses without breaking the application.

Change Preventers

难改

Shotgun Surgery 指的是对系统中某一处的修改需要在多个不同的地方进行相应修改，解决方法 extract method

Single Responsibility Principle

> • Every class, module, or function in a program should have one responsibility/purpose in a program.

> • Every class, module, or function should have only one reason to change

Couplers

Feature envy 喜欢用别人的数据

Inappropriate intimacy 喜欢用别人的内部变量

Message Chains 连续调用

Middle Man 做了很多额外的委托工作，将请求转发给其他类，而自身并没有多少实际的逻辑处理

Dispensables

垃圾（重复、废弃代码）

另外

code smells 不是bug，不总是有问题的

重构MONOLITHIC to MICROSERVICES

一体化架构越来越垃圾，且无药可救时，该重构

Refactoring strategies:

1. Implement new features as services
2. Separate the presentation tier and backend
3. Break up the monolithic by extracting functionality into services