《开源软件通识基础》

开源的 DevOps 流程

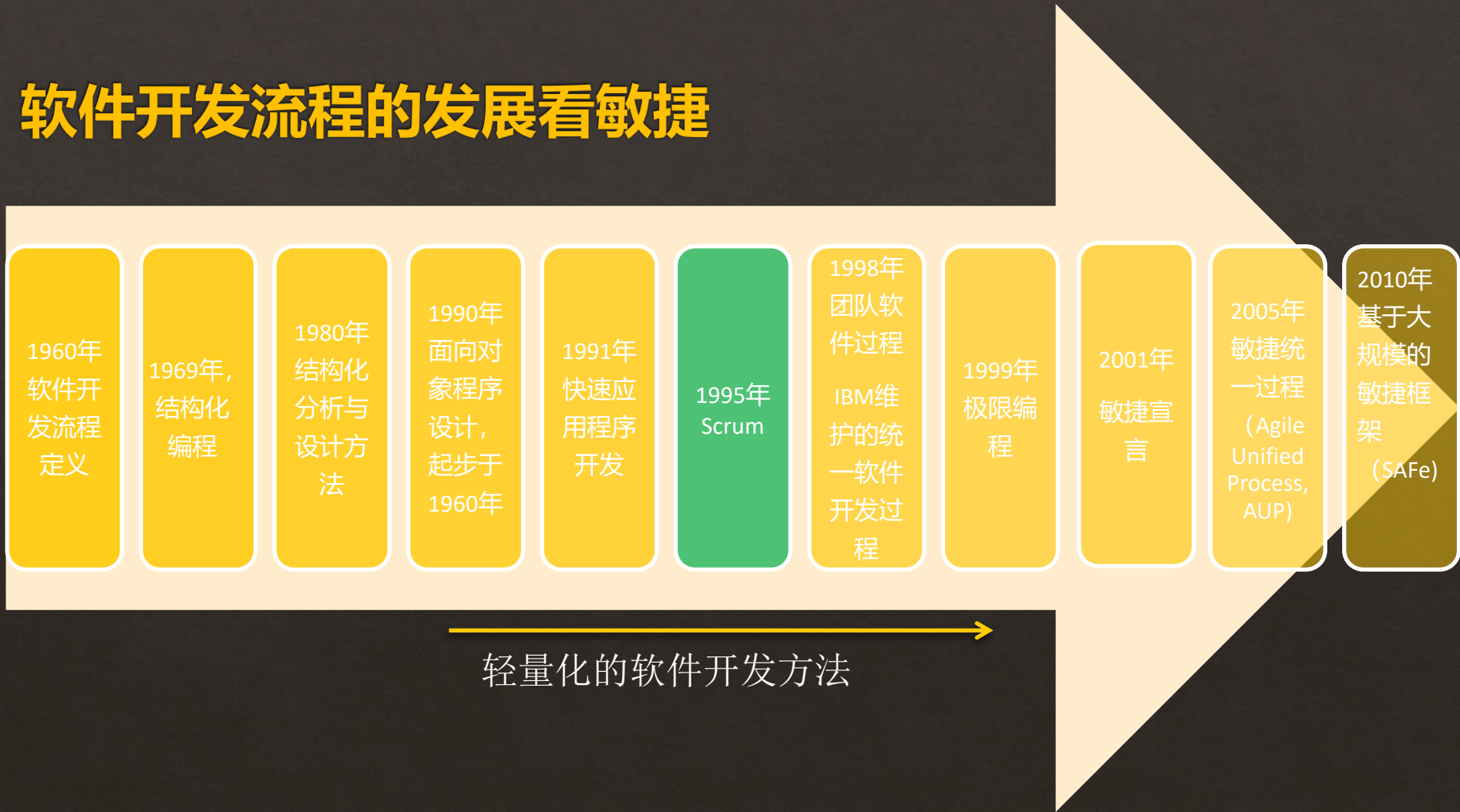
华东师范大学　张琰彬

DevOps

从敏捷到DevOps

DevOps 和开源的碰撞

软件开发流程的发展看敏捷



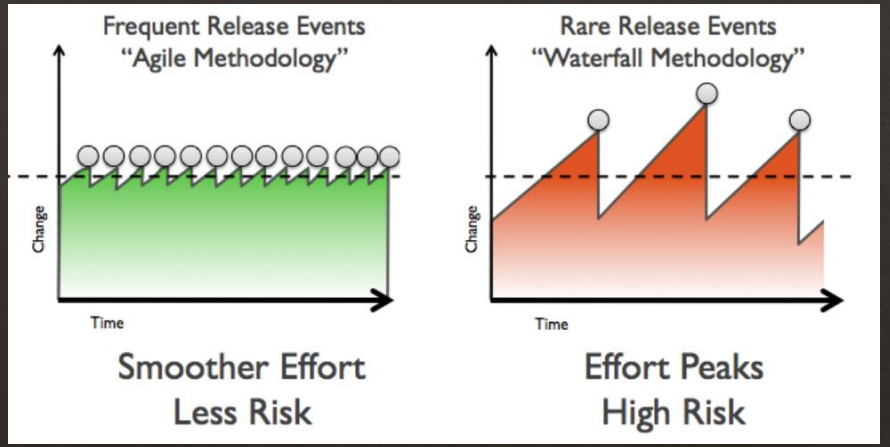
敏捷开发

价值

个体和互动：高于流程和工具。
工作的软件：高于详尽的文档。
客户合作：高于合同谈判。
响应变化：高于遵循计划。

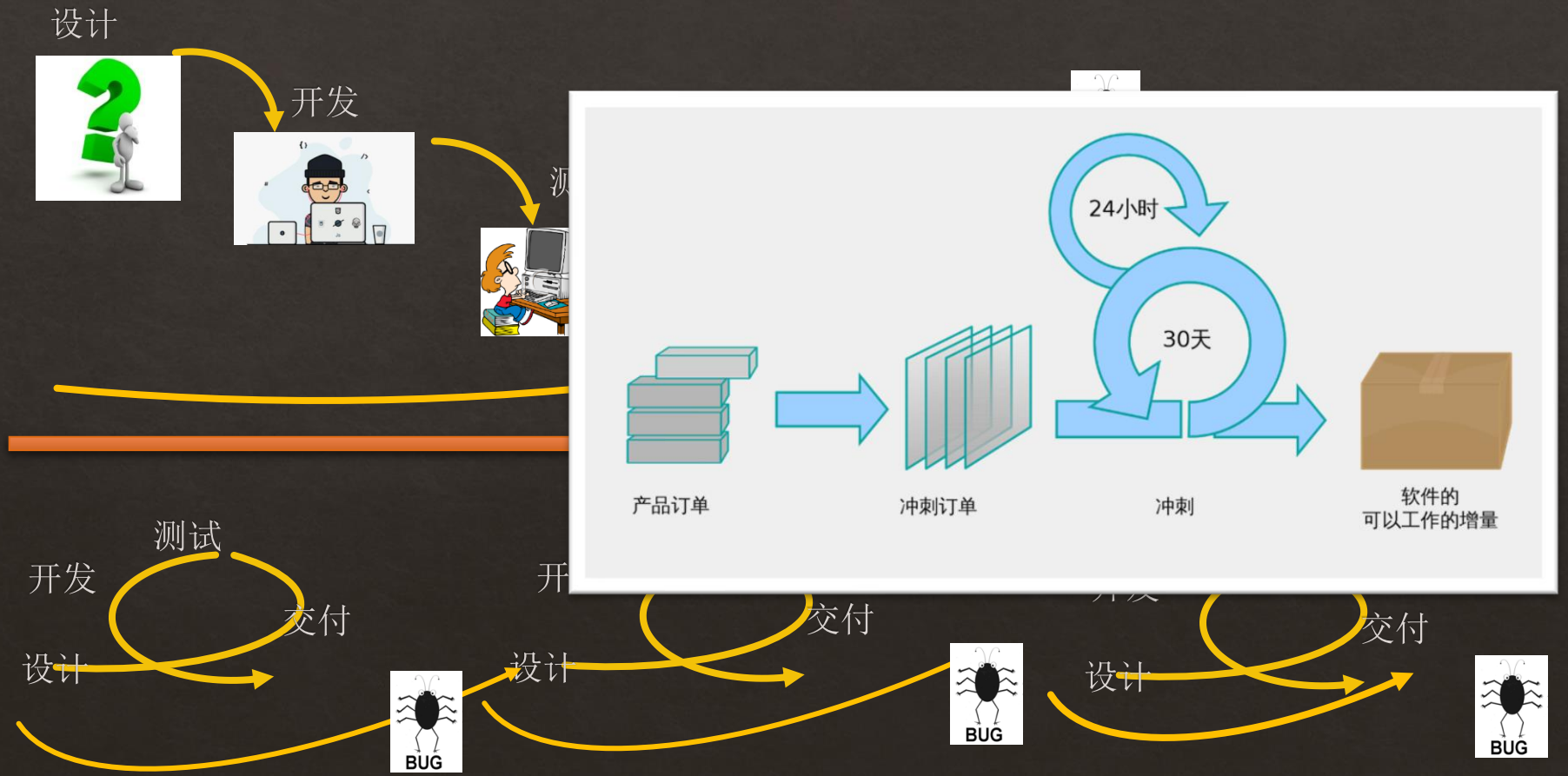
敏捷开发框架

极限编程，精益软件开发（实践）
Scrum Scrum Kanban (管理工作流程)
FDD 功能驱动开发（需求规范和开发）



https://commons.wikimedia.org/wiki/File:Agile-vs-iterative-flow.jpg

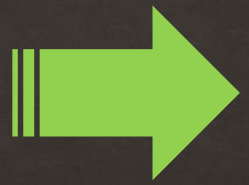
瀑布WaterFall模型 VS 敏捷开发



敏捷开发

是解决需求方，产品方和开发团队合作的文化和方法

是一种应对快速变化的需求的软件开发能力

敏捷开发  DevOps

敏捷开发 ➡ DevOps


1960年 软件开发流程定义
1990年 面向对象程序设计 1960年(起步)
1991年 快速应用程序开发
1995年 Scrum
1998年 团队软件过程 IBM维护的统一软件开发过程
1999年 极限编程
2001年 敏捷宣言
2005年 敏捷统一过程(AUP)
2009年 Velocity大会 10+ Deploys Per Day: Dev and Ops Cooperation at Flickr
2009年 比利时根特，DevOpsDays
2010年 The Agile Admin博客发表“What is DevOps”
2010年 第二届DevOps Day 《持续交付》
2012年 微服务
2015年 Cloud native
2020年 Cloud native+ DevOps +开源
轻量化与细微化开发管理，项目管理

敏捷开发 DevOps

DevOps

用户+ 需求方
+产品方

敏捷
开发

开发团队：
开发+测试

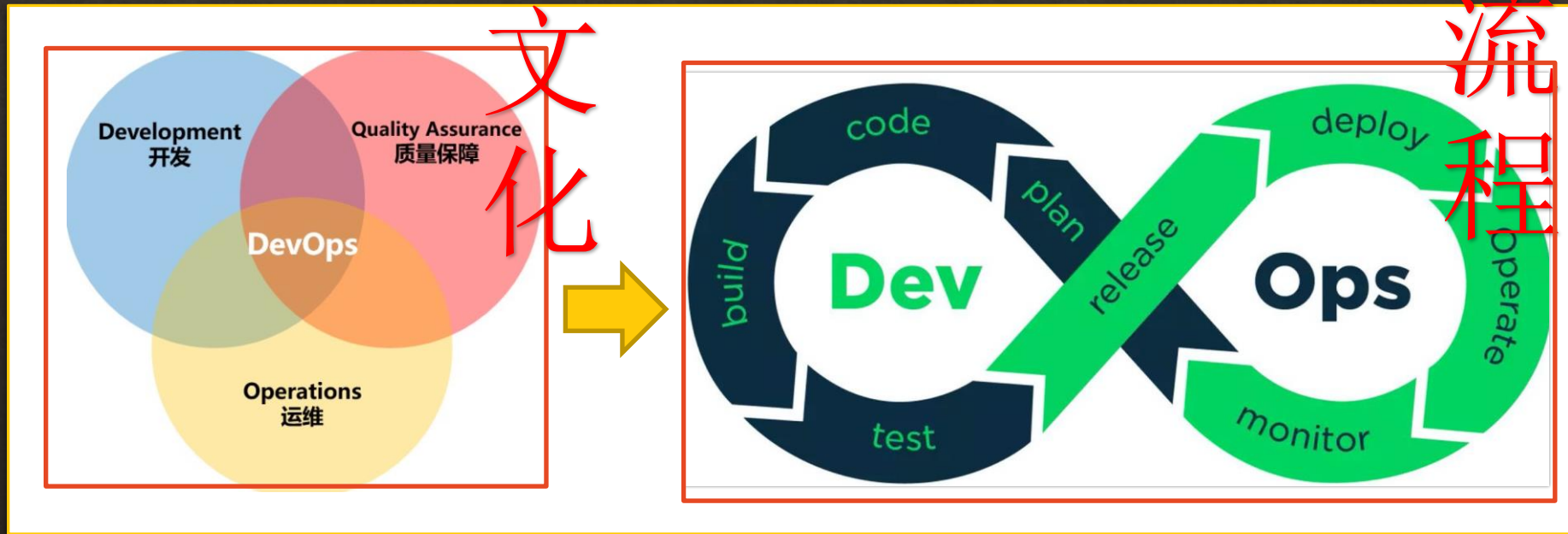
DevOps

运维团队:
运维+IT

DevOps

Development +Operations 重视“软件开发人员（Dev）”和“IT运维技术人员（Ops）”之间沟通合作的文化、运动或惯例。透过自动化“软件交付”和“架构变更”的流程，来使得构建、测试、发布软件能够更加地快捷、频繁和可靠。

https://zh.wikipedia.org/wiki/DevOps

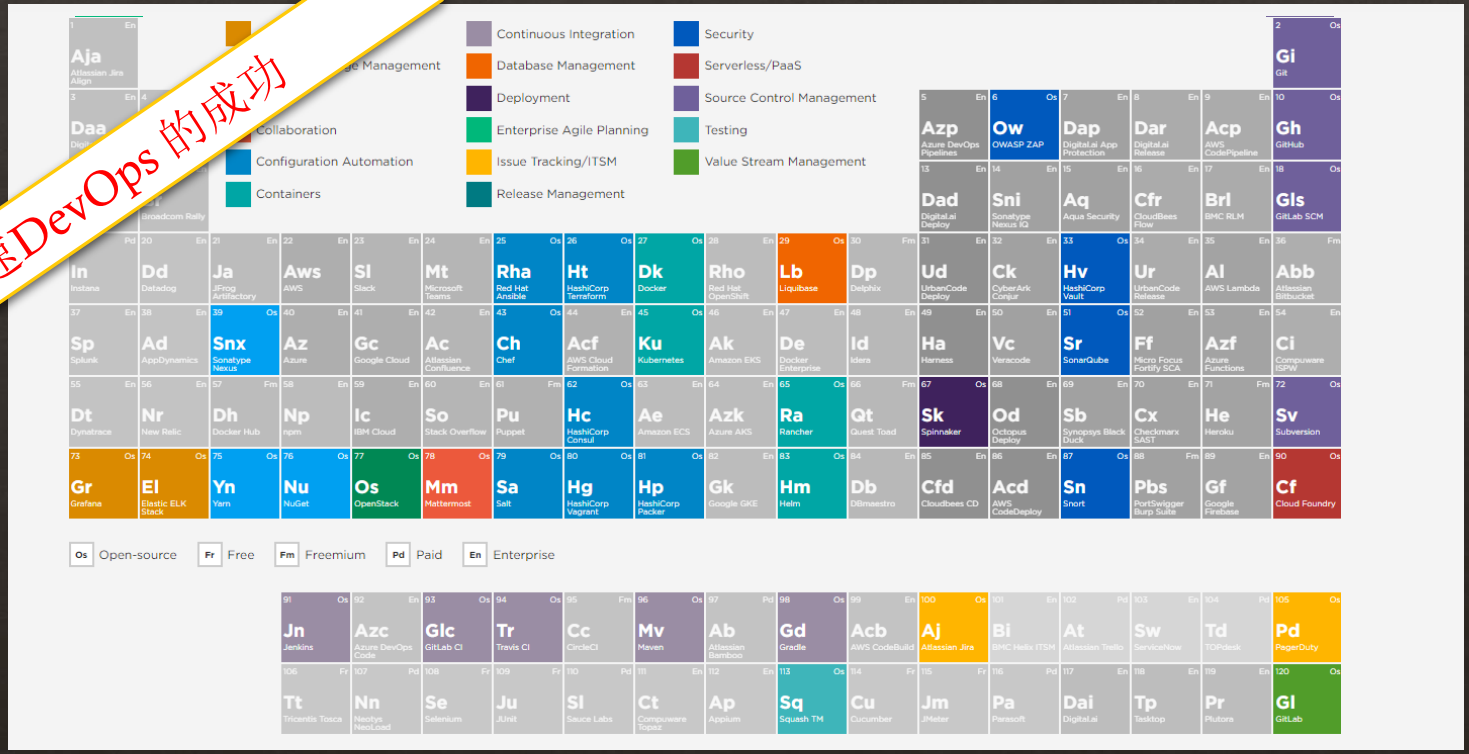


DevOps

从敏捷到DevOps

DevOps 和开源的碰撞

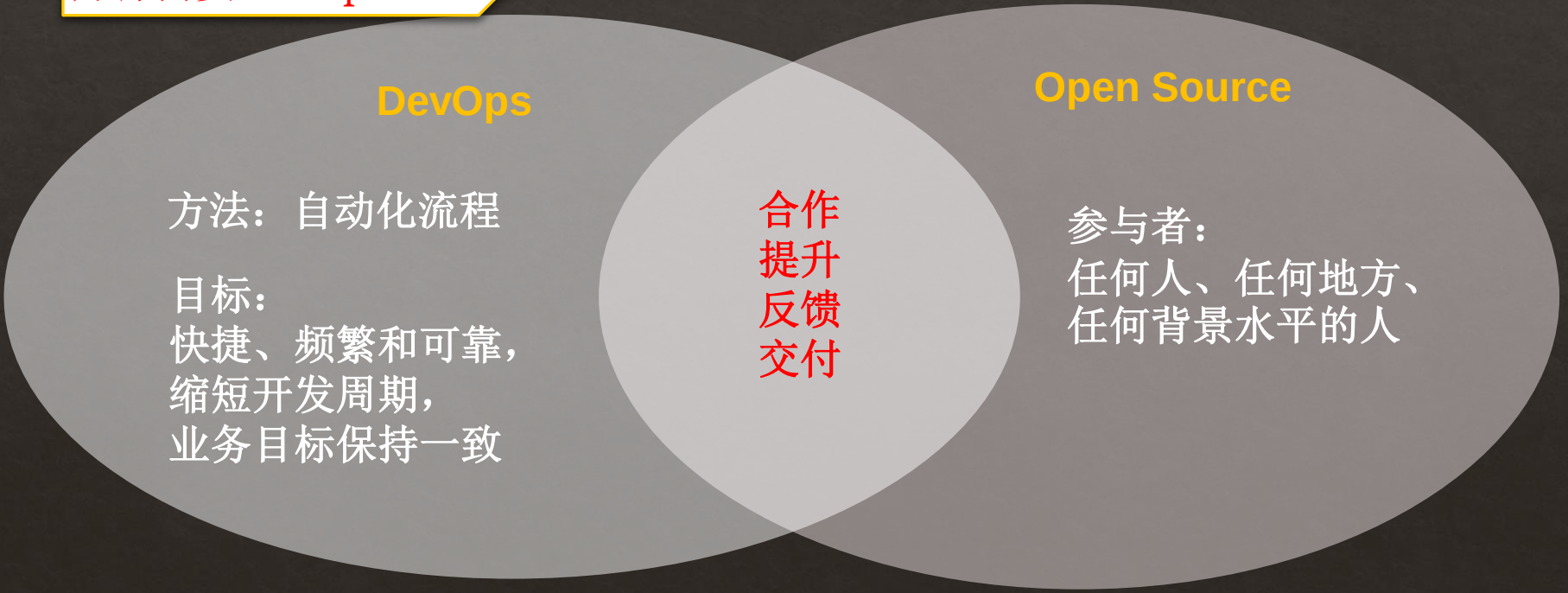
DevOps 与 开源（Open Source）的碰撞



https://digital.ai/periodic-table-of-devops-tools

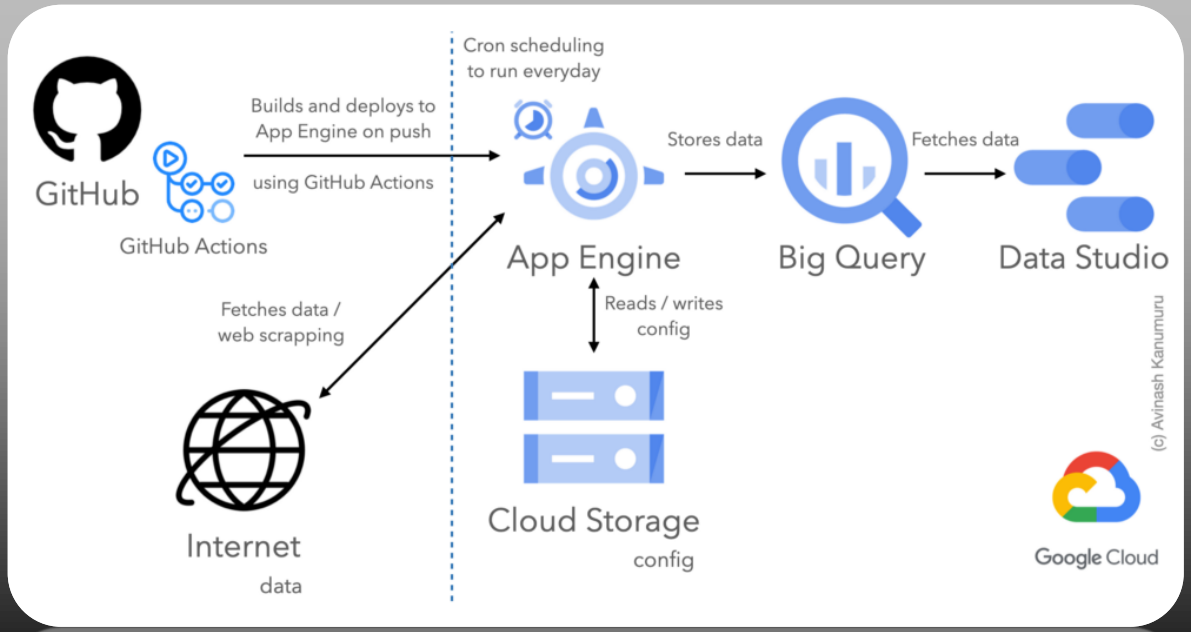
DevOps 与 开源（Open Source）的碰撞

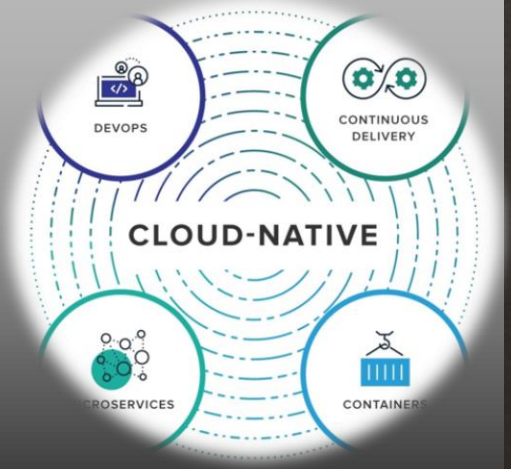
开源需要DevOps



DevOps 与 开源（Open Source）的碰撞

开源促使DevOps新形态





Deploy to Google App Engine using GitHub Actions (CI/CD)

Q&A

《开源软件通识基础》

开源项目代码管理

华东师范大学　张琰彬

开源项目代码管理

开源项目面临的质量问题

代码的持续集成测试管理体系

传统软件测试体系的变革和发展

基于开源的代码质量管理体系

开源项目代码管理

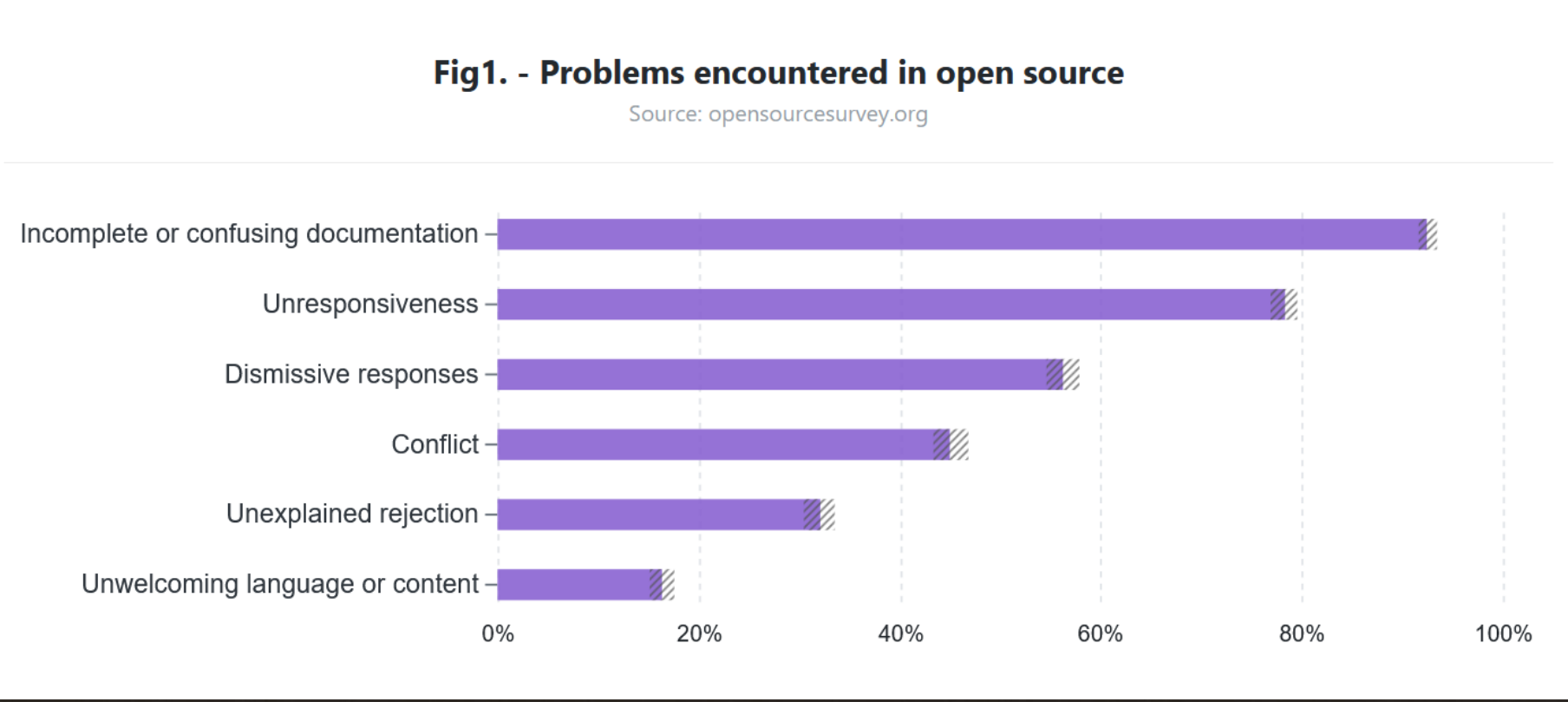
开源项目面临的质量问题

代码的持续集成测试管理体系

传统软件测试体系的变革和发展

基于开源的代码质量管理体系

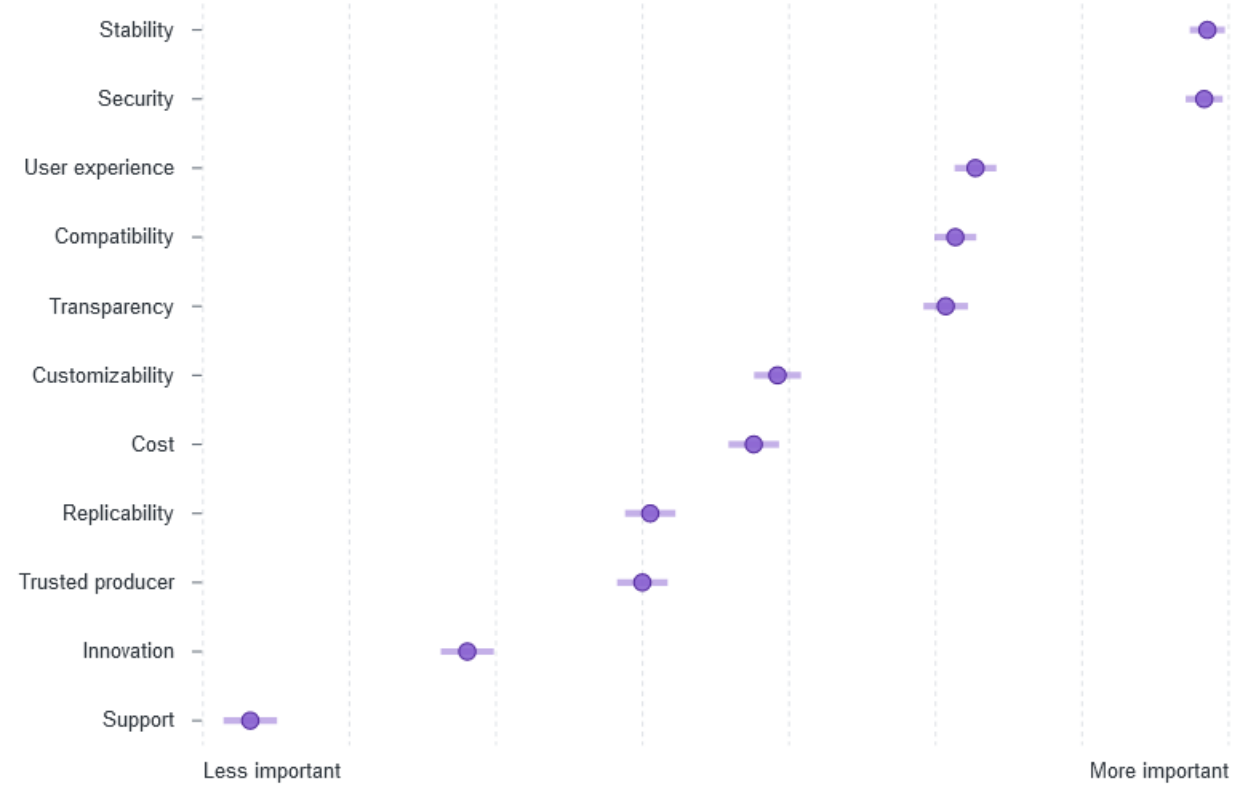
什么是一个好的开源项目



https://opensourcesurvey.org/2017/#overview

Fig5. - What open source users value in software

Source: opensourcesurvey.org



什么是一个好的开源项目

什么是一个好的开源项目

文档管理

介绍文档，用户安装文档， 用户使用/开发文档

言简意赅，清晰易懂，及时更新

代码管理

高质量的代码管理

清晰的代码迭代管理

社区管理

积极的响应

活跃的社区建设

社区文化建设，License 选择

Where the world builds software

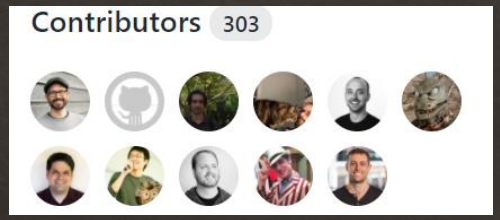
Millions of developers and companies build, ship, and maintain their software on GitHub—the largest and most advanced development platform in the world.

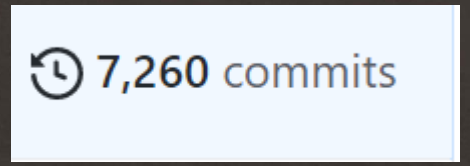
开源项目代码平台

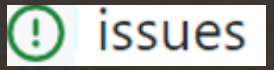
56+ million	3+ million	100+ million	72%
Developers	Organizations	Repositories	Fortune 50

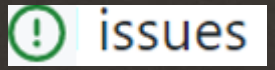
开源项目面临的问题

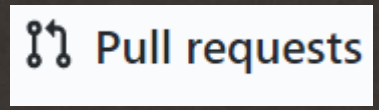
开源项目面临的问题

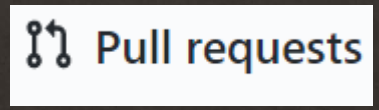


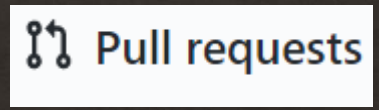


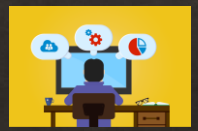


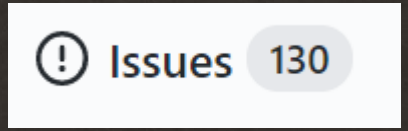


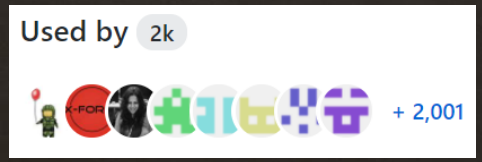


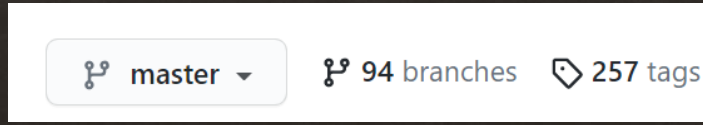






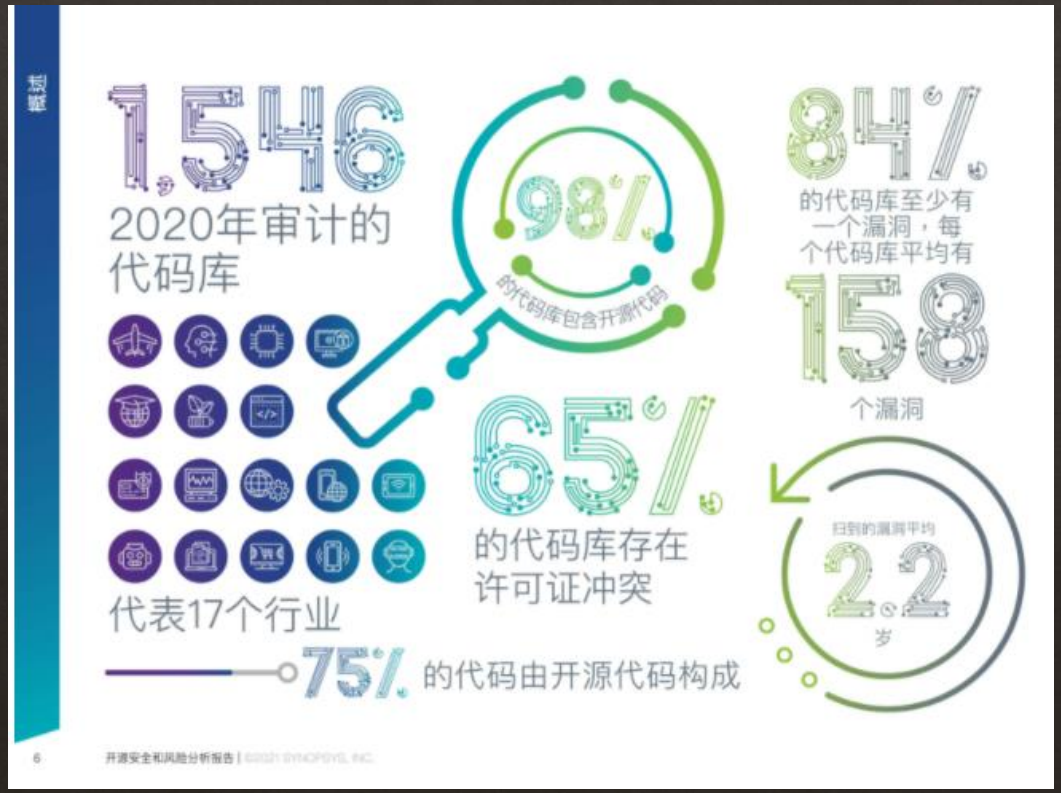






如何解决这些问题和代码（**pull request**）的管理?

开源项目面临的问题



新思科技发布《2021年开源安全和风险分析》报告

- 98%的医疗保健行业代码库包含开源，其中有67%的代码库存在漏洞。
- 97%的金融服务/金融科技行业代码库包含开源，其中超过60%的代码库存在漏洞。
- 92%的零售和电子商务行业代码库包含开源，其中71%的代码库存在漏洞。

如何解决这些问题?

代码的持续集成测试管理体系

开源项目代码管理

开源项目面临的质量问题

代码的持续集成测试管理体系

传统软件测试体系的变革和发展

基于开源的代码质量管理体系

什么是持续集成(Continuous Integration CI)

DevOps,CI,CD

DevOps

Continuous Deployment

Continuous Integration

- 第一：持续顺畅高质量地交付有效价值
  - 缩短交付时间
  - 快节奏交付
  - 从质量保障到质量协助
- 第二：极致弹性和韧性的系统运行
  - 业务负载自动、实时的扩缩容
  - 系统安全、合规和稳定的运行

持续集成(Continuous Integration CI) 操作流程

持续集成(Continuous Integration CI) 操作流程

Github: Continuous integration (CI) is a software practice that requires frequently committing code to a shared repository

“a development practice that requires developers to integrate code into a shared repository several times a day. Each check-in is then verified by an automated build, allowing teams to detect problems early” (Fowler, 2006) Martin Fowler
https://martinfowler.com/

频繁的多次的将本地的代码修改合并到公共代码仓库对应的分支

开发提交合并请求-PR（Pull Request)

主代码仓库repo

Diff → Pull Request

Forked repo

Git clone

本地代码

修改代码

Git add
Git commit

提交代码变动

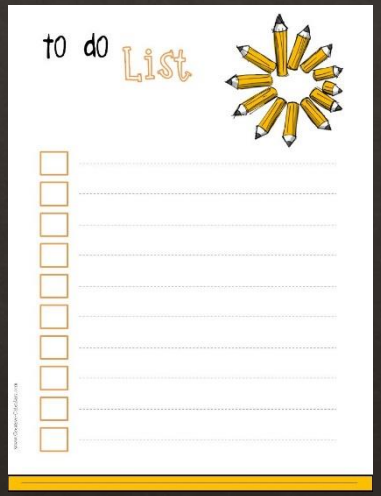
Git push

提交PR 到代码最终合并，中间发生了什么?

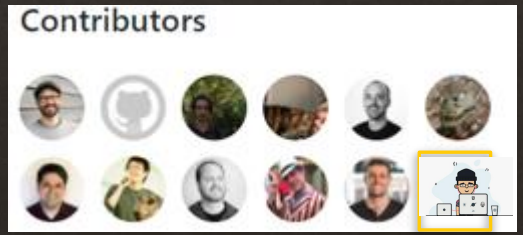
- 提交pull request

Pull requests

- 新的PR 产生



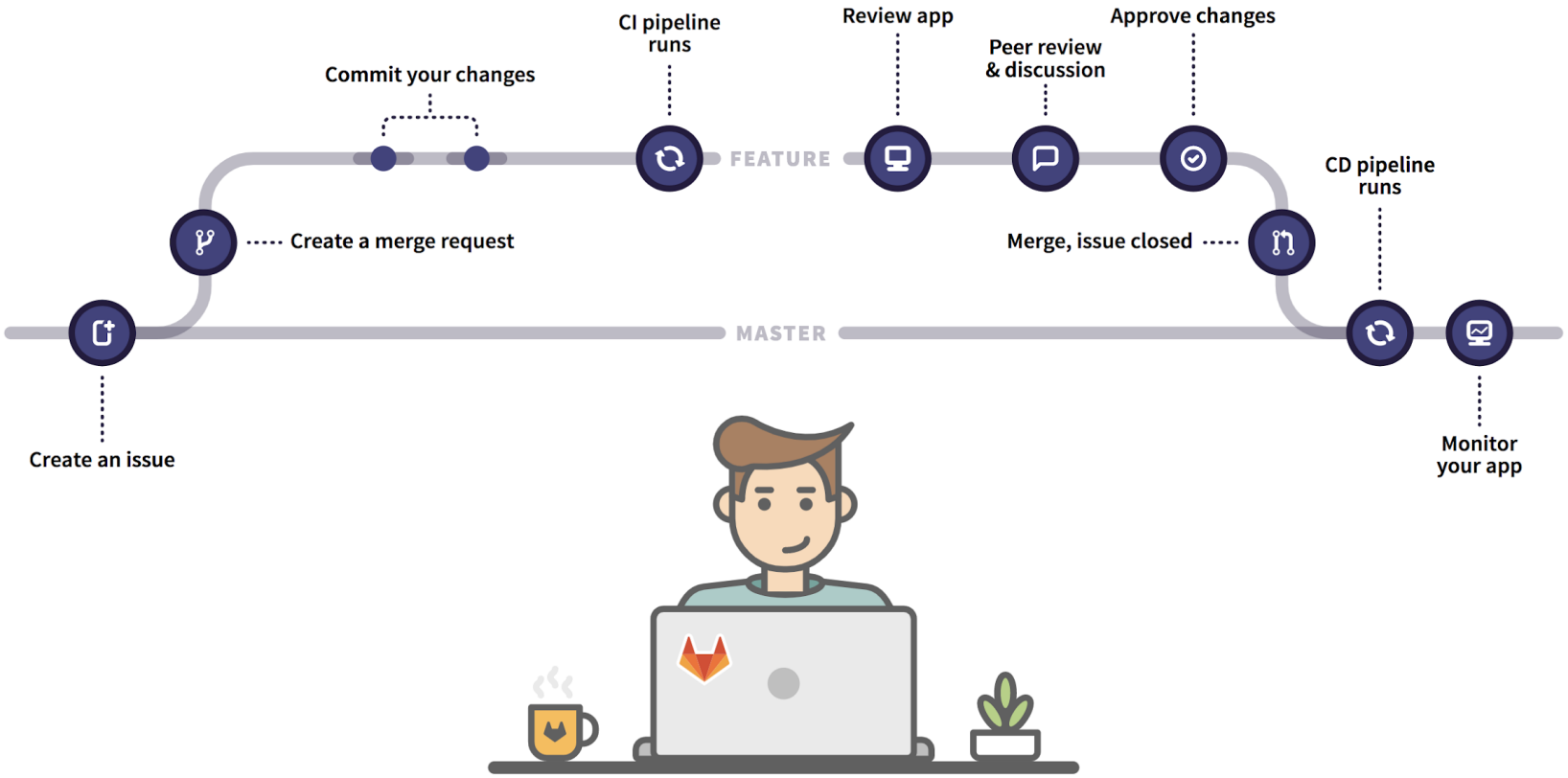
- ✓ 触发集成测试Continuous Integration Test(CI test)
- ✓ Comments
- ✓ Rebase/Update
- ✓ 再次提交
- ✓ CI test …
- ✓ Merge 合并代码



- 成为Contributors

- ✓ 触发集成测试 (CI test)
- ✓ CD(持续集成部署）

Gitlab推荐工作流



持续集成带来的优点

Continuous Integration doesn't get rid of bugs, but it does make them dramatically easier to find and remove.
Martin Fowler, Chief Scientist, ThoughtWorks

快速发现错误： 每完成一点更新，就集成到对应的分支，可以快速发现错误，定位错误也比较容易。

防止分支大幅偏离主干： 如果不是经常集成，分支或者主干又在不断更新，会导致以后集成的难度变大，甚至难以集成。

持续集成发布流程-CI/CD

极狐 GITLAB

GitHub

gitee

CODING

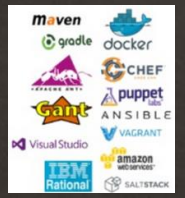
1. 检测代码变动
2. 自动构建编译
3. 自动测试
4. 自动 打包

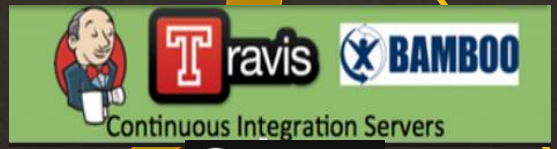
持续集成服务器

编译服务器

测试服务器

打包发布





GitHub

极狐 GITLAB

通知

持续集成测试流程管理工具平台

Actions -- Github

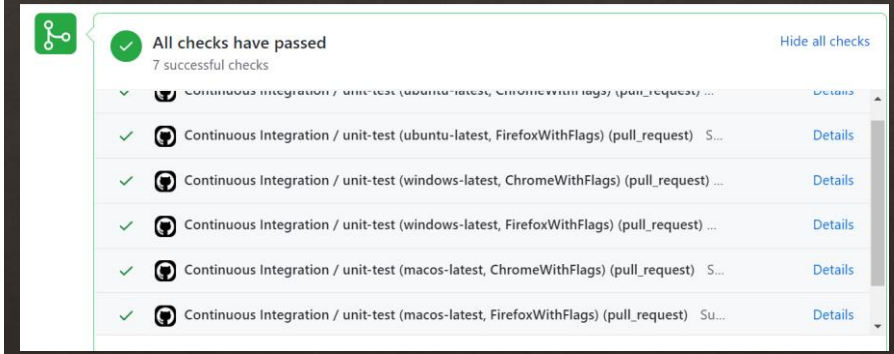
Travis CI – 3rd party public server

Jenkins -- Self host CI server

GitHub Actions

GitHub Actions

Automate, customize, and execute your software development workflows right in your repository with GitHub Actions. You can discover, create, and share actions to perform any job you'd like, including CI/CD, and combine actions in a completely customized workflow.



https://lab.github.com/githubtraining/github-actions:-hello-world?overlay=register-box-overlay

GitHub Actions

通过*.yml 文件描述整个CI/CD 的workflow, 在GitHub 提供的服务器上或者自己维护的服务器上进行CI/CD 的流程自动构建

```yaml
name: Python package

on:
  push:
    branches: [ master ]
  pull_request:
    branches: [ master ]

jobs:
  build:

    runs-on: ubuntu-latest
    strategy:
      matrix:
        python-version: [2.7]

    steps:
    - uses: actions/checkout@v2
    - name: Set up Python ${{ matrix.python-version }}
      uses: actions/setup-python@v2
      with:
        python-version: ${{ matrix.python-version }}
    - name: Install dependencies
      run: |
        python -m pip install --upgrade pip
        python -m pip install flake8  pytest
        if [ -f requirements.txt ]; then pip install -r requirements.txt; fi
    - name: Lint with flake8
      run: |
        # stop the build if there are Python syntax errors or undefined names
        flake8 . --count --select=E9,F63,F7,F82 --show-source --statistics
```

名字

监听事件

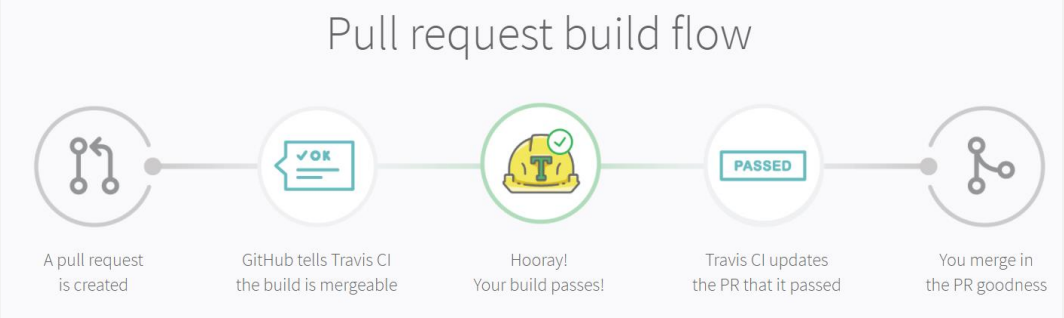
编译环境设置

编译

运行

- name
- on
- on.<event_name>.types
- on.<push|pull_request>.<branches|tags>
- on.<push|pull_request>.paths
- on.schedule
- permissions
- env
- defaults
- defaults.run
- concurrency
- jobs
- jobs.<job_id>
- jobs.<job_id>.name
- jobs.<job_id>.needs
- jobs.<job_id>.runs-on
- jobs.<job_id>.permissions
- jobs.<job_id>.environment
- jobs.<job_id>.concurrency
- jobs.<job_id>.outputs
- jobs.<job_id>.env

Travis CI –free for open source project



Travis CI is a hosted[2] continuous integration service used to build and test software projects hosted on GitHub[3] and Bitbucket.[4]

Travis CI是在软件开发领域中的一个在线的，分布式的[1]持续集成服务，用来构建及测试在GitHub[2]托管的代码。这个软件的代码同时也是开源的，可以在GitHub上下载到[3]，尽管开发者当前并不推荐在闭源项目中单独使用它。[4]

Travis CI

通过*.travis.yml 文件描述整个CI/CD 的workflow,在Travis 提供的服务器上进行CI/CD 的流程自动构建

5 lines (4 sloc) | 45 Bytes

```
1  language: node_js
2
3  node_js:
4  - node
5  - 'lts/*'
```

语言

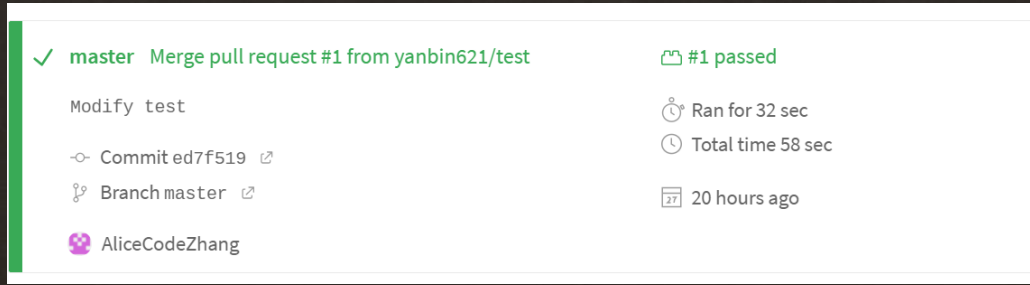
版本和环境参数

Default Build Script

The default build script for projects using nodejs is:

```
npm test
```

Bash



持续集成（CI）管理工具

Availability: Free
Platform: Cross-platform

https://jenkins.io/

Jenkins

Build great things at any scale

The leading open source automation server, Jenkins provides hundreds of plugins to support building, deploying and automating any project.

Jenkins 的优势

- 25万+的用户,600+ contributors
- 强大的开源社区文化和运营，持续交付基金会（CDF)成员
- 超丰富的插件支持（>1800+ Plugins）
- 支持几乎所有的source control management 和 Version control management tools
- 可定制化部署

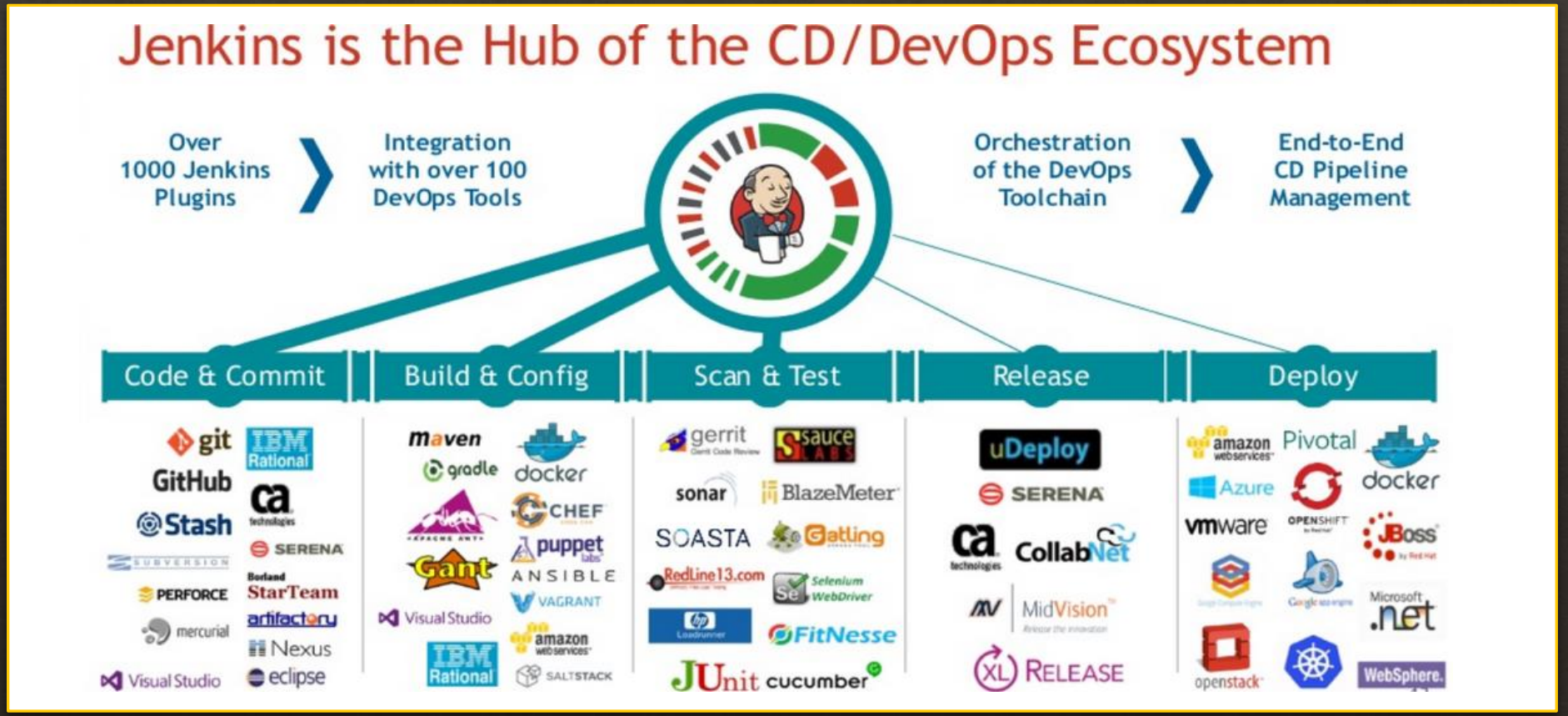
Jenkins 的优势

Source Control Management or the Version Control Management

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

Name	AccuRev	BitKeeper	CA Harvest	ClearCase	CVS	Darcs	Git	GNU Bazaar	Integrity	Mercurial	Perforce	Plastic	PVCS	StarTeam	Subversion	Surround	Synergy	Team Concert	Team Foundation Server
Vexor	No	No	No	No	No	No	Yes	No	No	No	No	No	No	No	No	No	No	No	No
TeamCity	Yes	No	No	Yes	Yes	No	Yes	Yes[24]	No	Yes	Yes	No	No	Yes	Yes	No	No	No	Yes
Team Foundation Server	No	No	No	No	No	No	Yes	No	No	No	No	No	No	No	Yes	No	No	No	Yes
OpenMake Software Meister	Yes	No	Yes	Yes	Yes	No	Yes	No	Yes	No	Yes	No	Yes	Yes	Yes	No	Yes	Yes	Yes
Jenkins-Hudson	Yes	Yes	Yes	Yes	Yes	Yes[17]	Yes	Yes	Yes[18]	Yes	Yes	Yes[19]	Yes	Yes	Yes	Yes[20]	Yes[21]	Yes[22]	Yes
Distelli	No	No	No	No	No	No	Yes	No	No	Yes	No	No	No	No	No	No	No	No	No
CruiseControl.NET	Yes	Yes	No	Yes	Yes	No	Yes	No	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes	No	Yes
CruiseControl	No	No	No	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	No	No	Yes	Yes	Yes	No	No	Yes
CABIE	No	No	No	No	Yes	No	No	No	No	No	Yes	No	No	No	Yes	No	No	No	No
BuildMaster	Yes	No	No	Yes	Yes	No	Yes	No	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes	No	No	Yes
BuildBot	No	No	No	No	Yes	Yes	Yes	Yes	No	Yes	Yes	No	No	No	Yes	No	No	No	No
Buddy	No	No	No	No	No	No	Yes	No	No	No	No	No	No	No	No	No	No	No	No
Bamboo	Yes[15]	No	No	Yes	Yes	No	Yes	No	No	Yes	Yes	No	No	No	Yes	No	No	No	Yes[16]
AppVeyor	No	No	No	No	No	No	Yes	No	No	Yes	No	No	No	No	Yes[14]	No	No	No	No
Apache Gump	No	No	No	No	Yes	No	No	No	No	No	No	No	No	No	Yes	No	No	No	No
Apache Continuum	No	No	No	Yes[2]	Yes[2]	No	Yes[2]	Yes[2]	No	Yes[2]	Yes[2]	No	No	Yes[2]	Yes[2]	No	Yes[2]	No	No
AnthillPro	Yes	No	Yes	Yes	Yes	No	Yes	No	Yes	Yes	Yes	No	Yes	Yes	Yes	No	Yes	Yes	Yes

Jenkins 与CD 和DevOps 关系



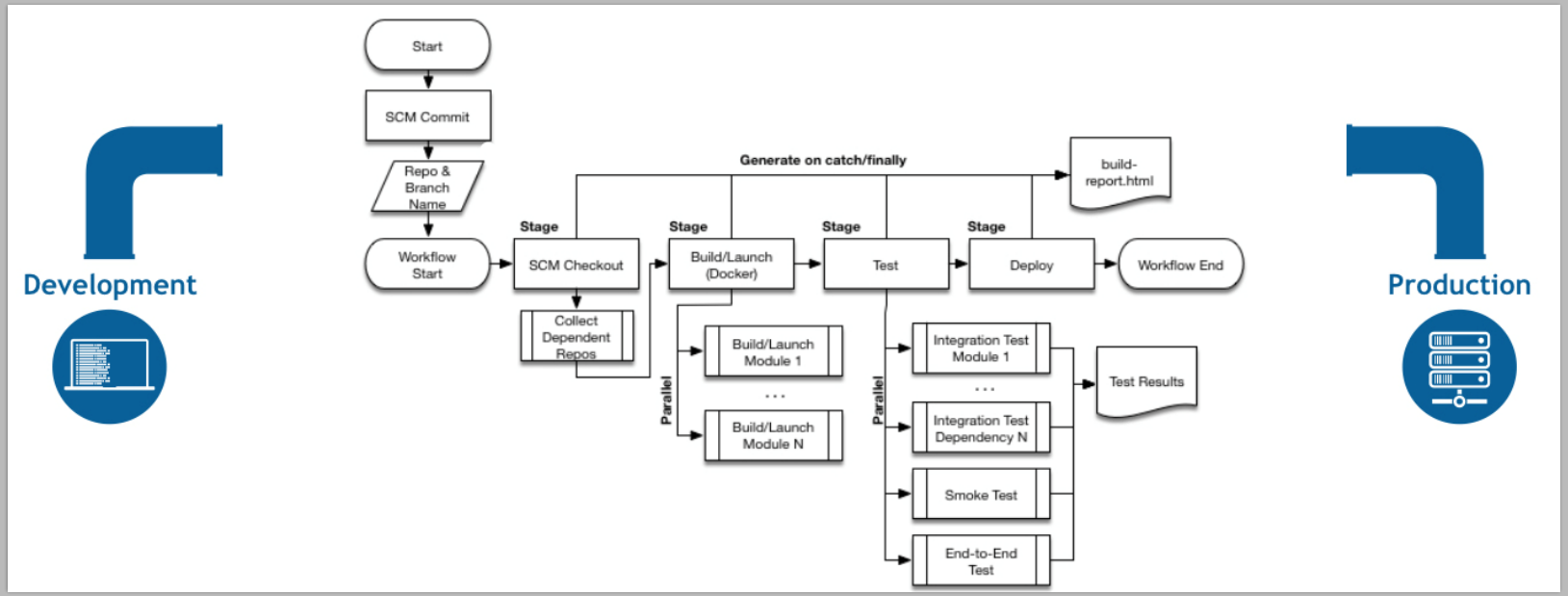
Jenkins Pipeline 机制

用解释性代码Jenkinsfile来描述，通过Jenkins服务器和一系列Jenkins 插件将整个CD(持续交付流程

Jenkins Pipeline 机制

Pipeline as code

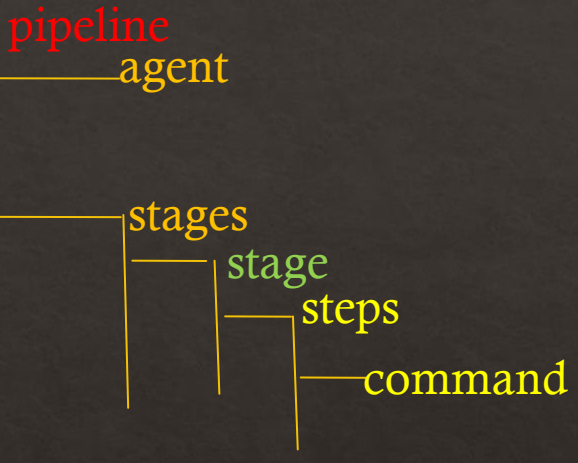
Design a whole pipeline with Jenkinsfile



http://Jenkins.io

Declarative Pipeline scripts 语法

```
pipeline {
    agent{
        label 'mac'
    }
    environment{
        mvnHome = tool 'maven'
    }
    stages{
      stage('Preparation') { // for display purposes
          // Get some code from a GitHub repository
          steps{
          git 'https://github.com/AliceCodeZhang/androidSample.git'
          }
       }
      stage('Build') {
         // Run the maven build
         steps{
           sh "pwd ; cd ./SeleniumTest; '${mvnHome}/bin/mvn' -Dmaven.test.failure.ignore clean package"
         }
       }
      stage('Results') {
         steps{
           junit '**/target/surefire-reports/*.xml'
           archive 'target/*.jar'
         }
      }
    }
}
```



Declarative Pipeline scripts 语法

DECLARATIVE PIPELINE SYNTAX - REQUIRED

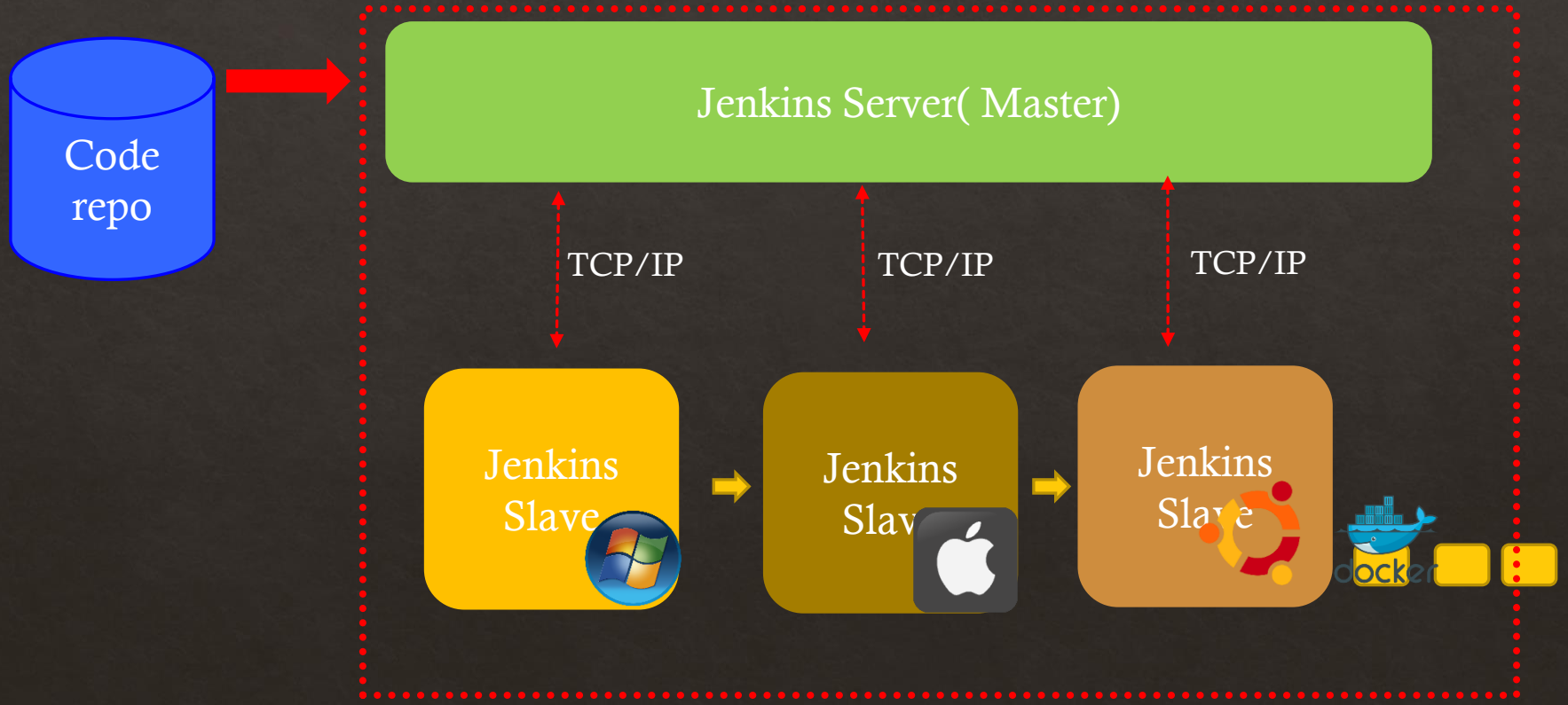
- **pipeline -** Contains the entire Jenkins Pipeline definition
- **agent -** Defines the agent used for the entire Pipeline or a stage
  - **label -** Existing Jenkins node label
  - **docker -** Requires Docker-enabled node
    - **image -** Run inside specified Docker image
    - **label -** Existing Jenkins node label
    - **args -** Arguments for Docker container
  - **dockerfile -** Use a local Dockerfile
    - **filename -** Name of local Dockerfile
    - **label -** Existing Jenkins node label
    - **args -** Arguments for Docker container
- **stages -** Contains Pipeline stages and steps
- **stage -** A specific named "stage" of the Pipeline
  - **steps -** One or more build steps that define the actions in the stage. Contains one or more of the following:
    - Any build step or build wrapper defined in Pipeline e.g. **sh, bat, timeout, echo, archive, junit,** etc.
    - **parallel (optional) -** Execute steps in parallel. May not be used with other steps
    - **script (optional) -** Execute Scripted Pipeline block
  - **when (optional) -** Runs stage conditionally
    - **branch -** Stage runs when branch name matches
    - **expression -** Boolean expression
  - **agent, environment, tools** and **post** may also optionally be defined in **stage**

+

DECLARATIVE PIPELINE SYNTAX - OPTIONAL

- **environment -** A sequence of "key = value" pairs to define environment variables
  - **credentials('<id>') -** Bind credentials to variable
- **options -** Options for the entire Pipeline. For example:
  - **skipDefaultCheckout -** Disable auto checkout SCM
  - **timeout -** Sets timeout for entire Pipeline
  - **buildDiscarder -** Discard old builds
  - **disableConcurrentBuilds -** Disable concurrent Pipeline runs
- **tools -** Installs pre-defined tools to be available on PATH
- **triggers -** Triggers for Pipeline based on schedule, polling, etc.
- **parameters -** Pipeline parameters that are prompted for at run time
- **post -** Defines actions to be taken when pipeline or stage completes based on outcome. Conditions execute in order:
  - **always -** Run regardless of Pipeline status
  - **changed -** Run if the result of Pipeline has changed from last run
  - **success -** Run if Pipeline is successful
  - **unstable -** Run if Pipeline result is unstable
  - **failure -** Run if the Pipeline has failed

Jenkins 的Master 和 Slaves 架构



https://wiki.jenkins.io/display/JENKINS/Distributed+builds

各个构建工具对比

- GitHub Actions
  配置简单
  无需额外配置服务器
  支持自维护服务器挂载

- Jenkins
  插件灵活
  自定义，自维护，分布式部署

- Travis CI
  第三方CI/CD 平台，构建相对简单，对免费支持的限制，可能会取消免费支持

开源项目代码管理

开源项目面临的质量问题

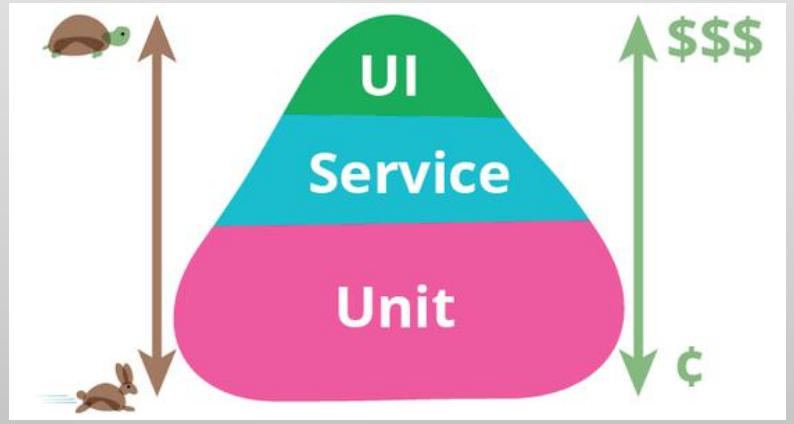
代码的持续集成测试管理体系

传统软件测试体系的变革和发展

基于开源的代码质量管理体系

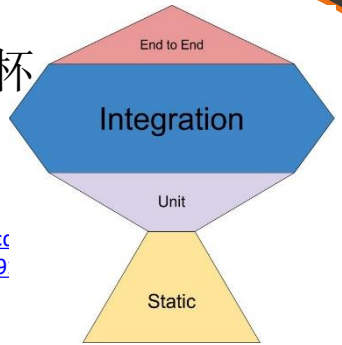
传统的测试体系

尽可能少的UI 测试
API，component , Integration test 多覆盖
Unit test 全覆盖



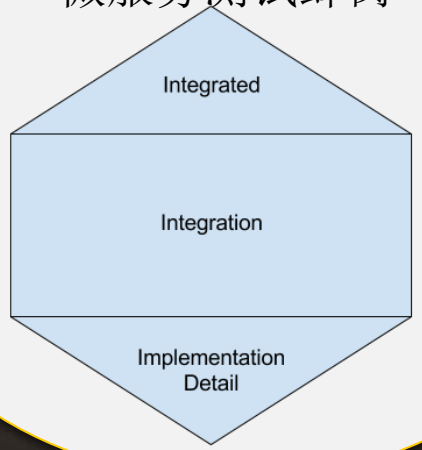
Testing Pyramid Rationale - Source: [Martin Fowler]

JS Web 的奖杯模型



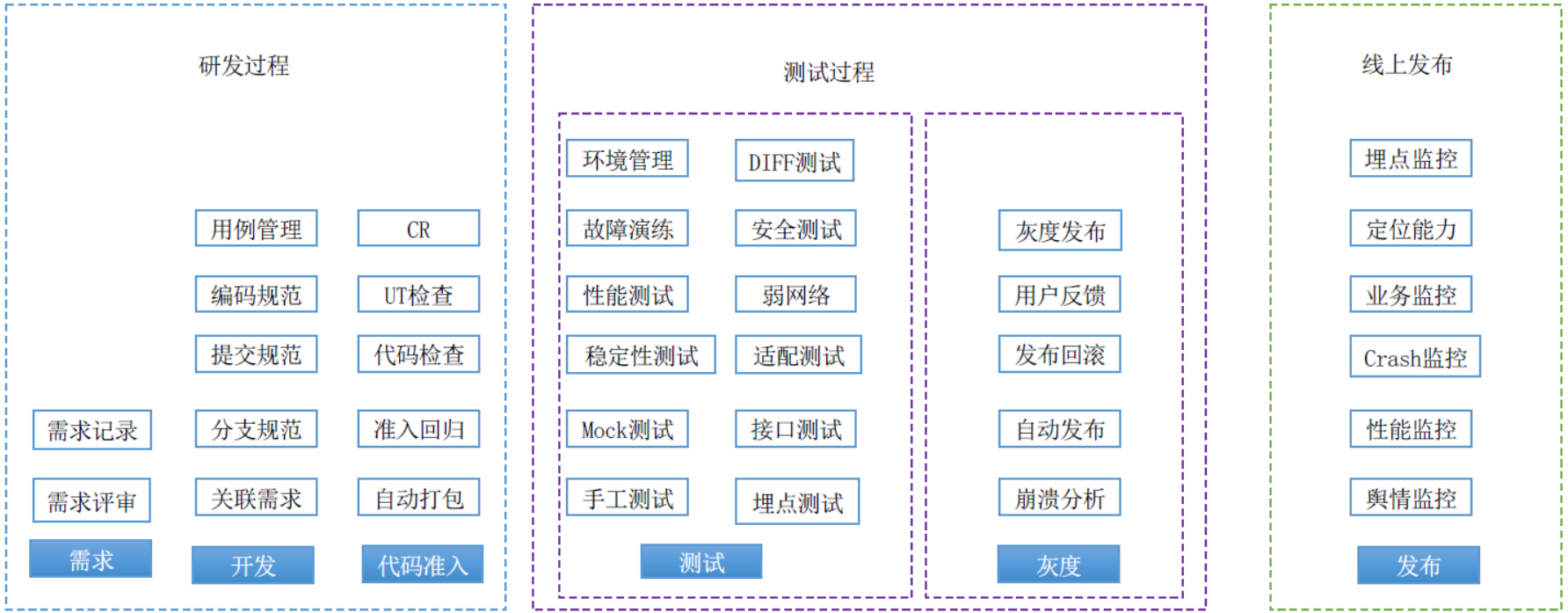
https://twitter.com/kentcdodds/status/960723172599992832/photo/1

微服务测试蜂窝



https://engineering.atspotify.com/2018/01/11/testing-of-microservices/

完善的质量管理体系



开源项目代码管理

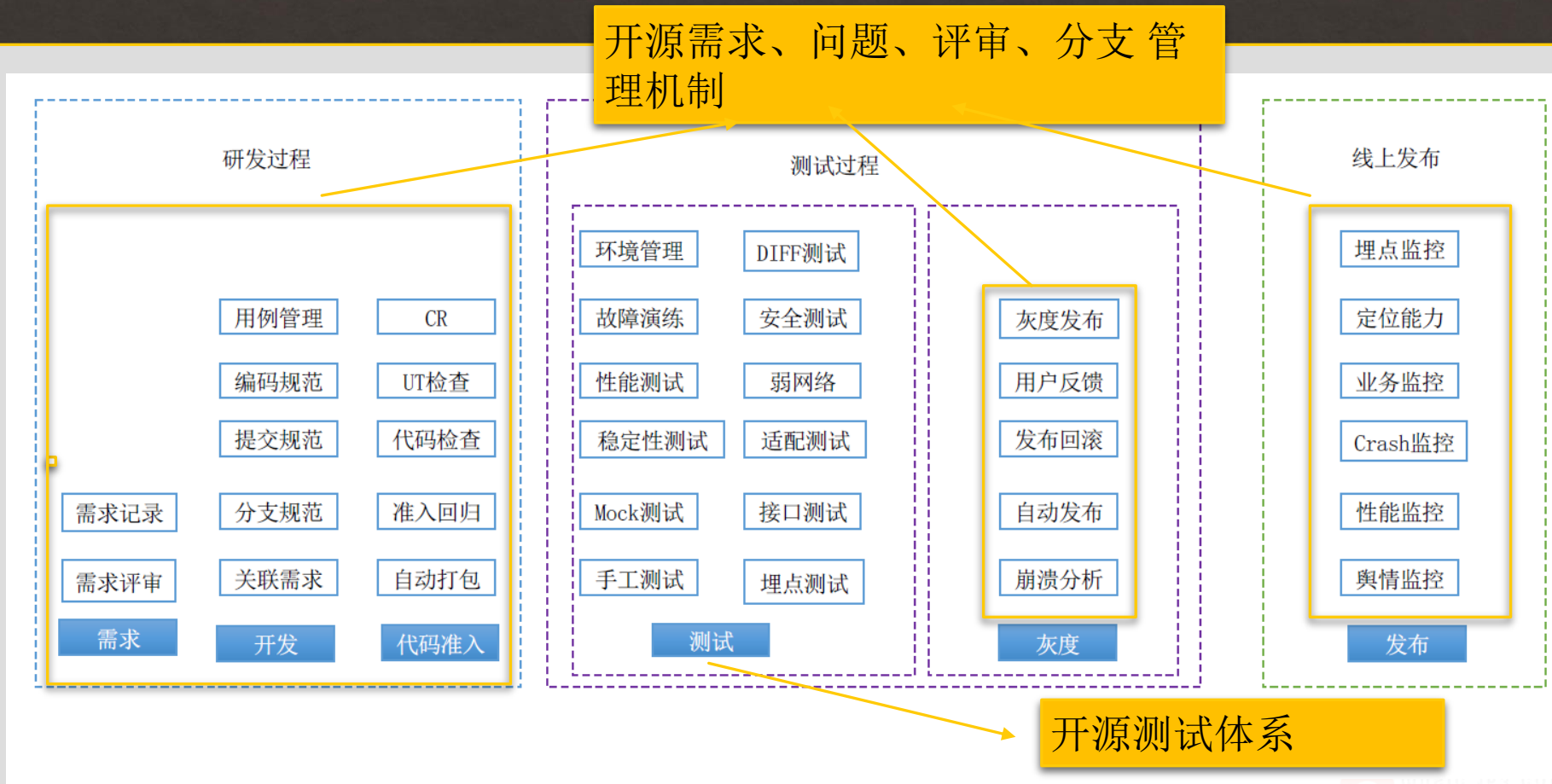
开源项目面临的质量问题

代码的持续集成测试管理体系

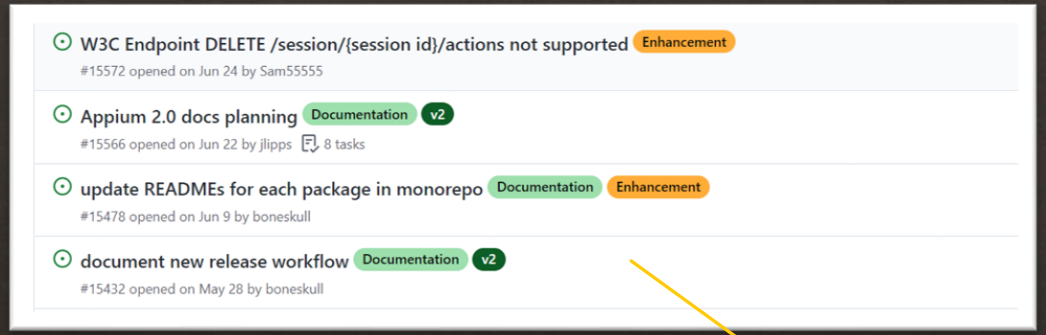
传统软件测试体系的变革和发展

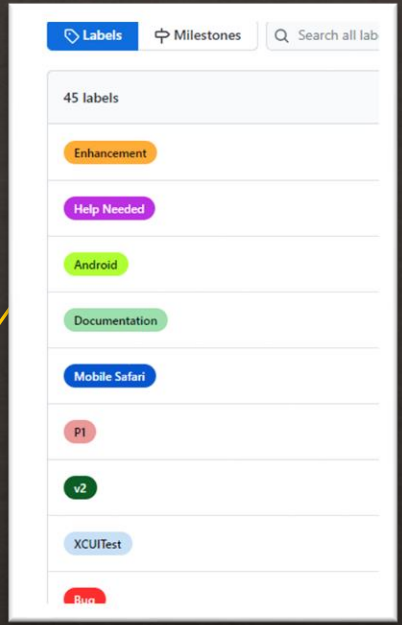
基于开源的代码质量管理体系

完善的质量管理体系

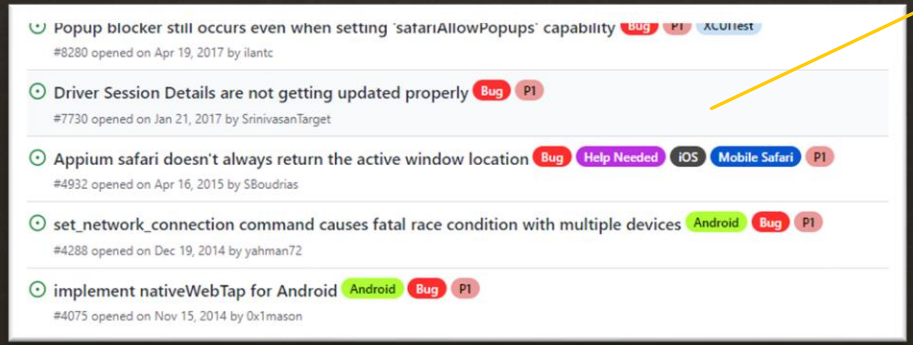

开源需求、问题、评审、分支 管理机制
研发过程
测试过程
线上发布
用例管理
CR
编码规范
UT检查
提交规范
代码检查
需求记录
分支规范
准入回归
需求评审
关联需求
自动打包
需求
开发
代码准入
环境管理
DIFF测试
故障演练
安全测试
性能测试
弱网络
稳定性测试
适配测试
Mock测试
接口测试
手工测试
埋点测试
测试
灰度发布
用户反馈
发布回滚
自动发布
崩溃分析
灰度
埋点监控
定位能力
业务监控
Crash监控
性能监控
舆情监控
发布
开源测试体系

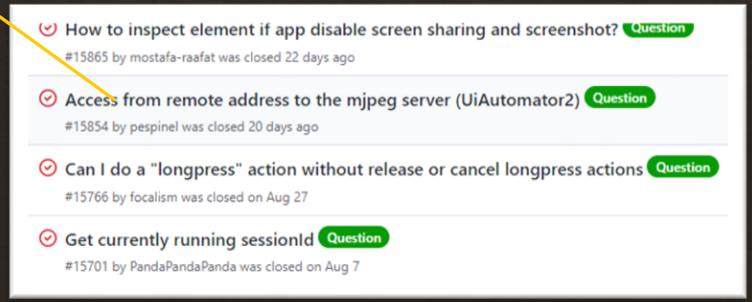
开源需求、问题、管理机制



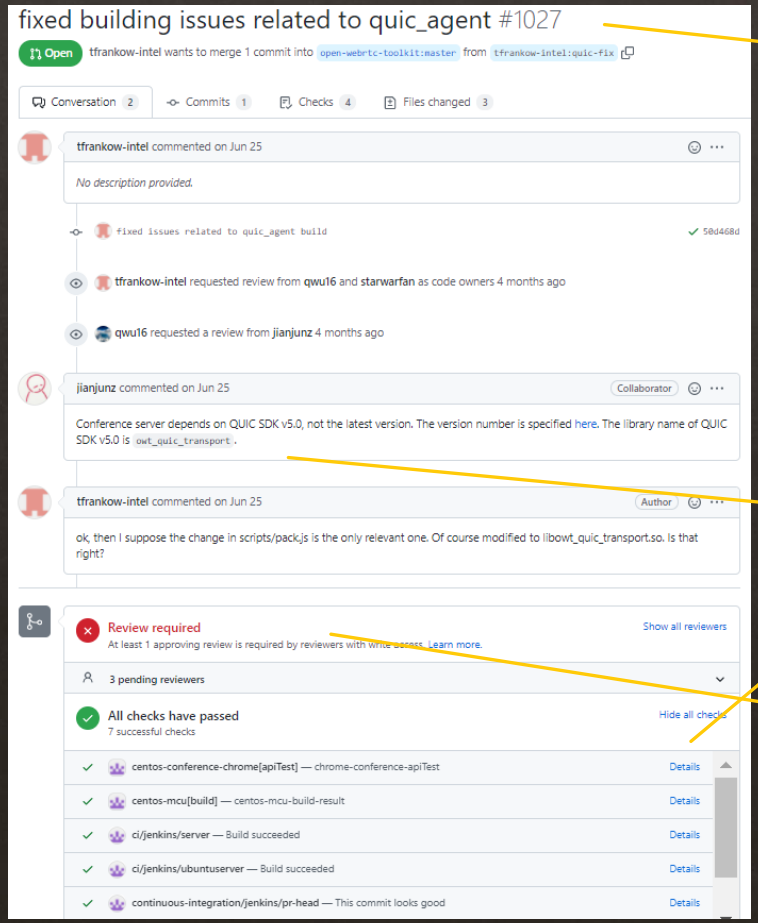


Issue





开源评审管理机制



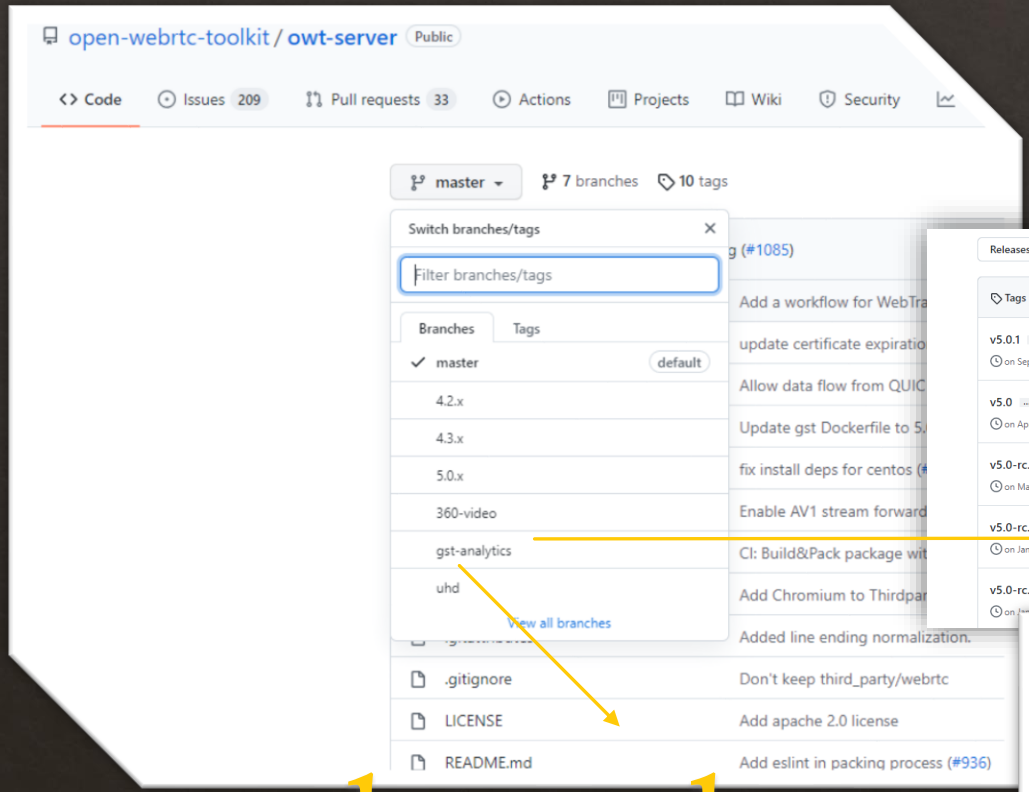
Open Pull request

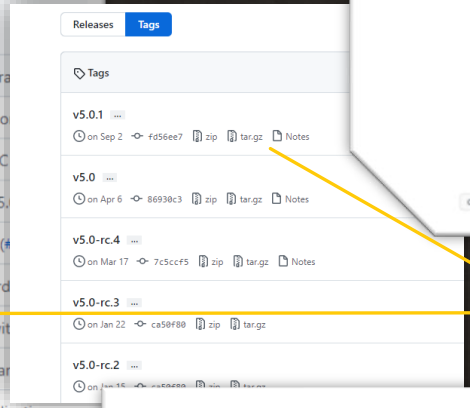
自动评审- 持续集成测试

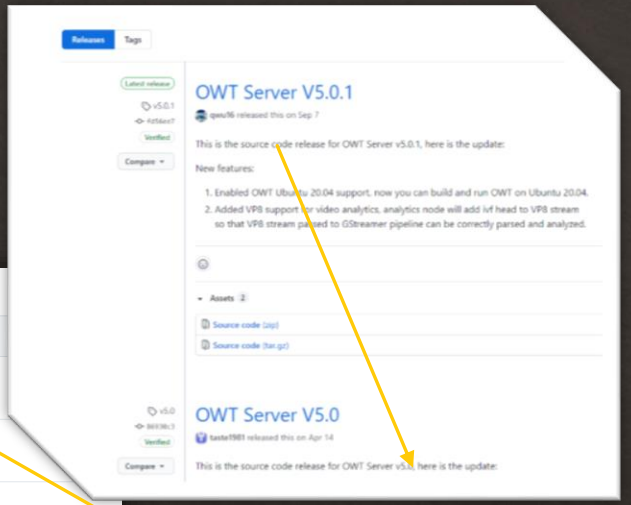
人工评审- comments

人工评审- 允许入库

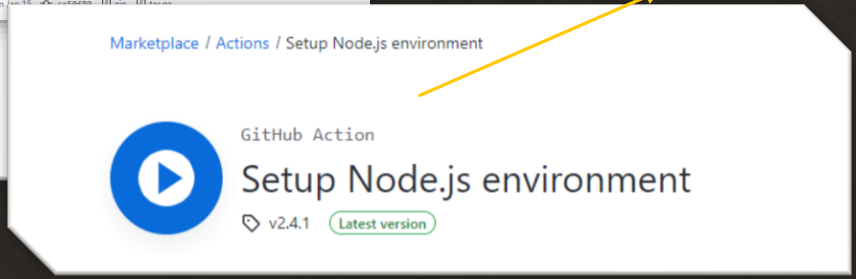
开源分支管理机制





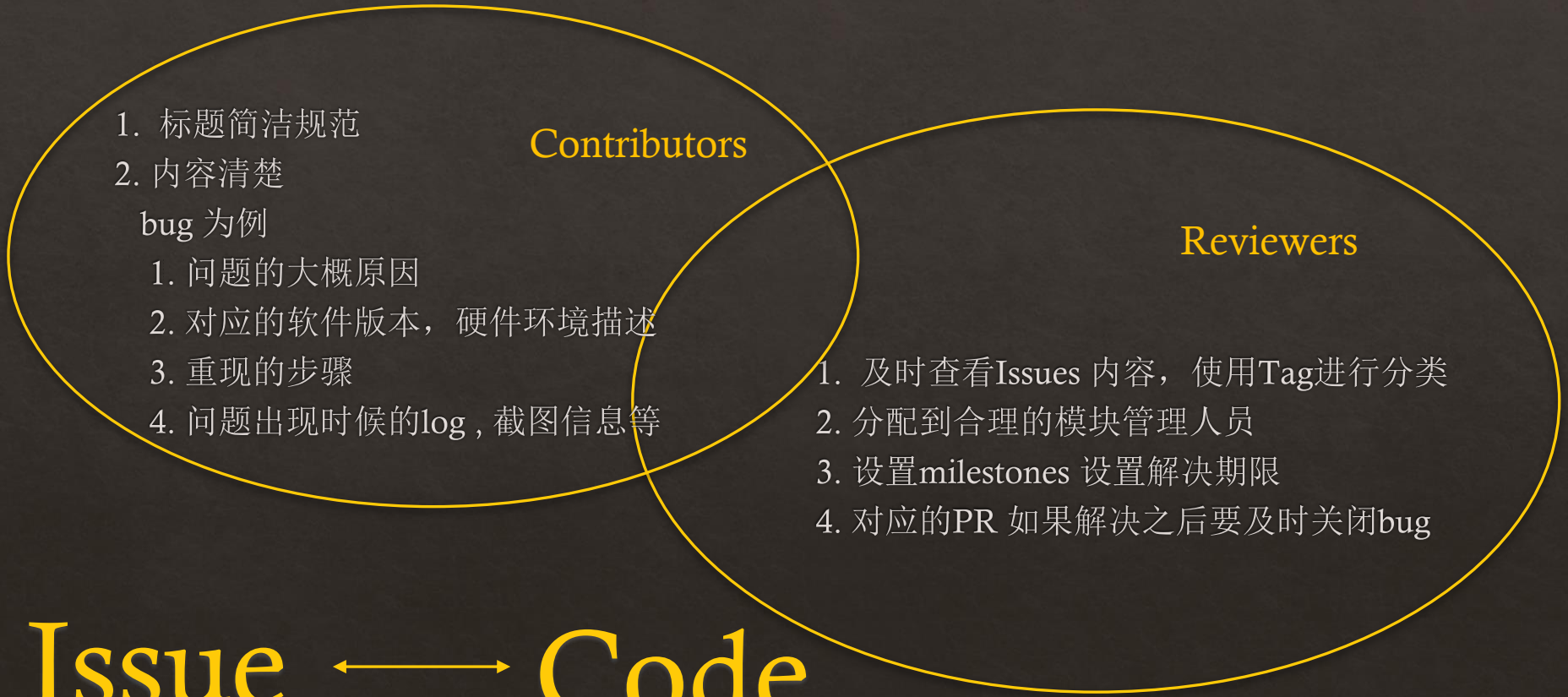


Release



branches

开源需求、问题、评审、分支 管理机制



Issue ⟷ Code

测试体系

代码扫描

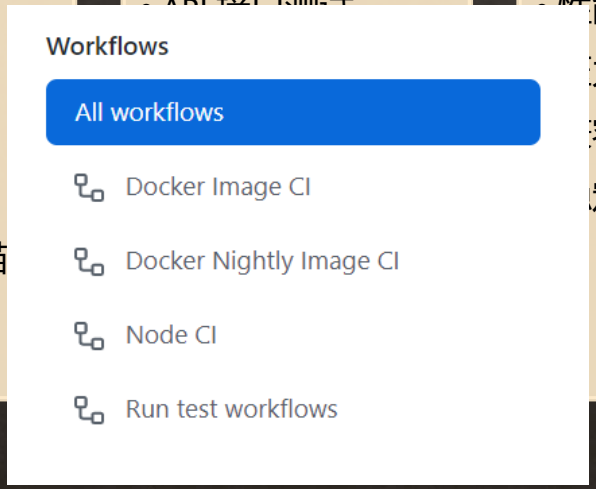
- 静态代码扫描
- 代码格式检测
- 动态代码扫描
- 安全检测
- 漏洞检测
- 第三方依赖扫描
- ...

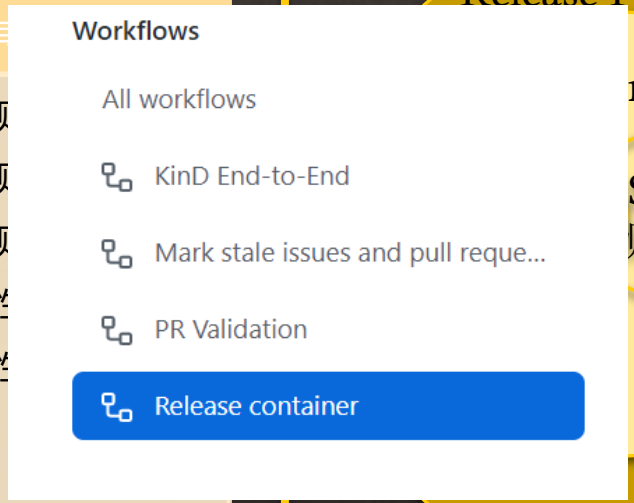
功能测试

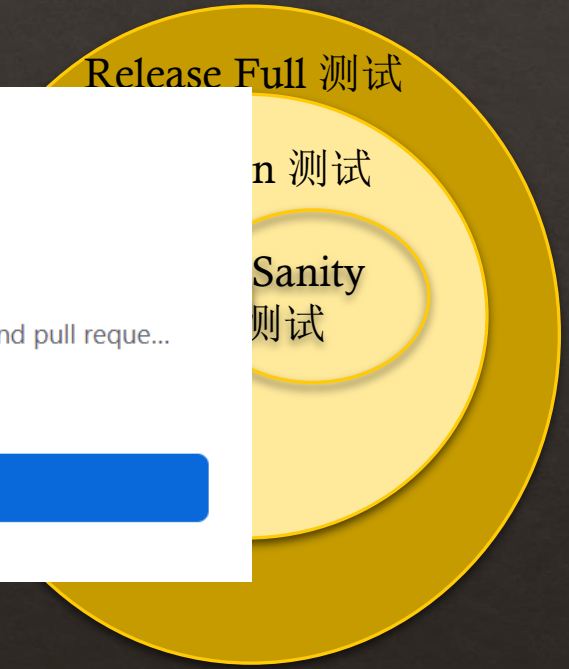
- 单元测试

其

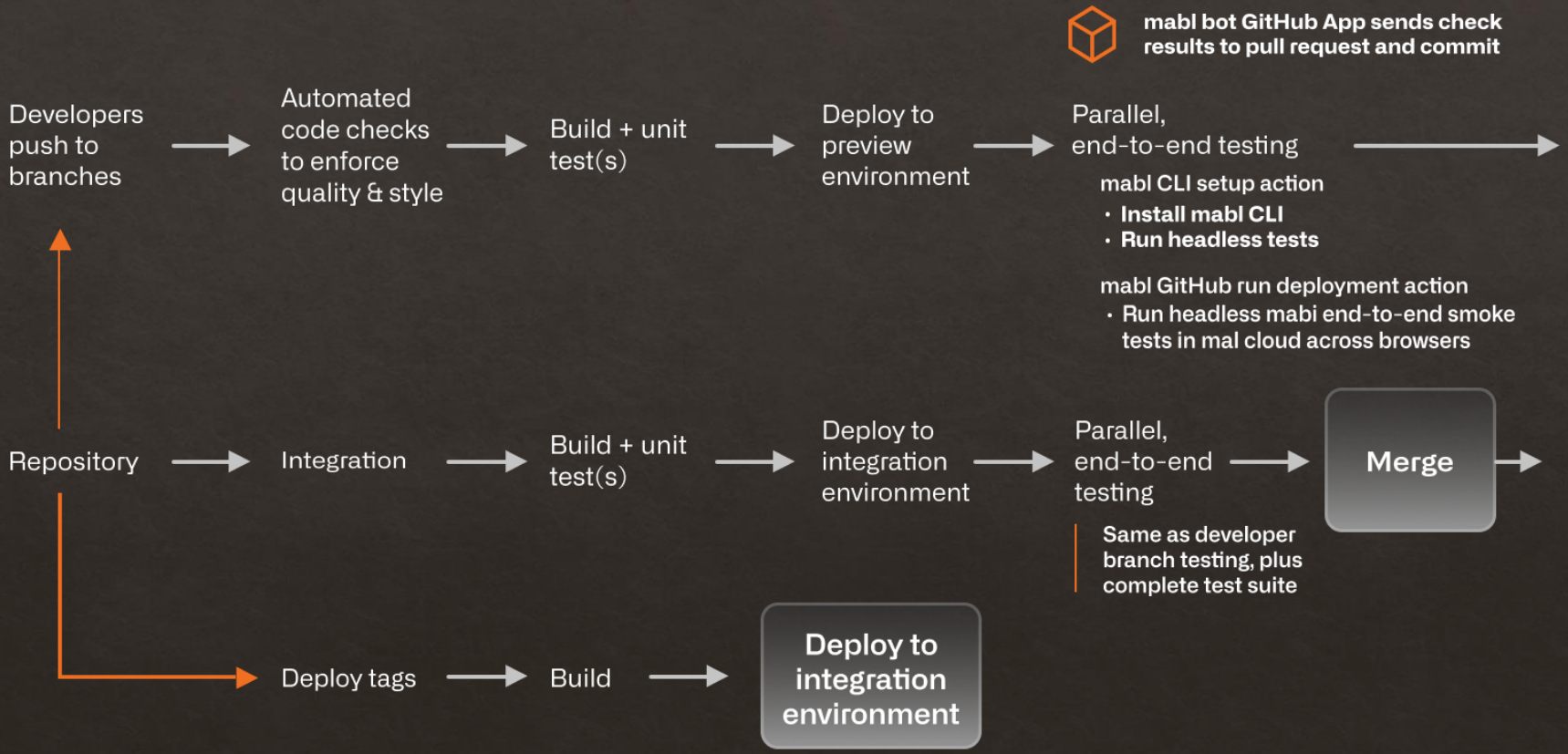
- 安全测
- 性能测
- 力测
- 容性
- 定性



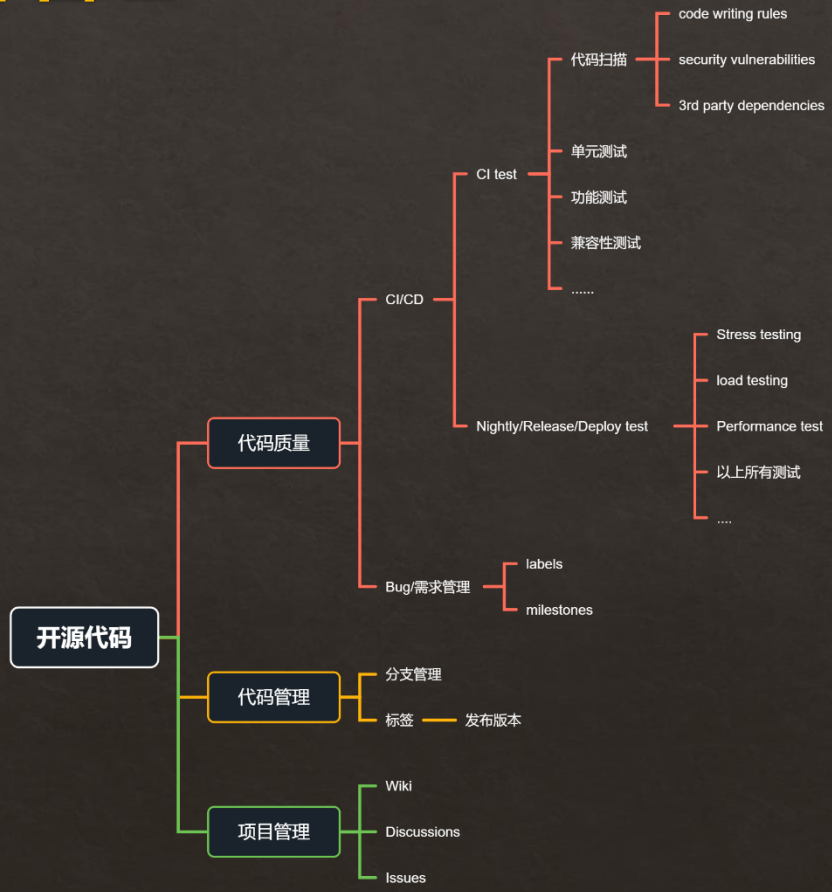




开源Workflow


mabl bot GitHub App sends check results to pull request and commit
Developers push to branches
Automated code checks to enforce quality & style
Build + unit test(s)
Deploy to preview environment
Parallel, end-to-end testing
mabl CLI setup action
Install mabl CLI
Run headless tests
mabl GitHub run deployment action
Run headless mabi end-to-end smoke tests in mal cloud across browsers
Repository
Integration
Build + unit test(s)
Deploy to integration environment
Parallel, end-to-end testing
Merge
Same as developer branch testing, plus complete test suite
Deploy tags
Build
Deploy to integration environment

开源项目整体管理


开源代码
代码质量
CI/CD
CI test
代码扫描
code writing rules
security vulnerabilities
3rd party dependencies
单元测试
功能测试
兼容性测试
.....
Nightly/Release/Deploy test
Stress testing
load testing
Performance test
以上所有测试
....
Bug/需求管理
labels
milestones
代码管理
分支管理
标签
发布版本
项目管理
Wiki
Discussions
Issues

Q&A