COMP130015.02

软件工程

10. 软件集成与发布

复旦大学计算机科学技术学院

沈立炜

shenliwei@fudan.edu.cn

传统手工方式进行的软件集成与发布

- 以手工方式完成的软件项目集成与发布
  - ✓ 开发人员手工合并其他开发人员的代码，然后手工编译代码并运行测试
  - ✓ 交付团队手工安装与配置软件所需的操作系统、数据库、应用服务软件、以及第三方软件，然后手工将软件与相关的数据复制到试运行环境与生产环境，最后启动软件
- 问题和不足
  - ✓ 易出错：容易出错以及引入人为失误，集成与发布过程不可重复也不可靠，难以及时交付，同时浪费大量时间
  - ✓ 反馈晚：软件在完成大部分开发工作之后才第一次产生可运行的版本并进行部署、测试，到了较晚的集成与发布阶段才发现各种问题（如不满足需求、开发环境与生产环境不一致等），从而导致难以修复的错误甚至发布日期推迟

自动化的持续集成与发布

- 采用持续集成、持续交付、持续部署等软件开发实践，实现频繁且自动的软件集成与发布，每次对应的软件修改较小
  - ✓ 大大减小集成与发布的出错与调试风险，且更加容易回滚
  - ✓ 加快反馈速度，让相关涉众（如开发团队、测试团队、交付团队、客户等）都参与到反馈流程中，尽快解决集成与发布过程中的问题
- 构建、测试、部署等各环节都是自动化的，使得集成与发布过程变得可重复、可靠、高效，同时也避免了人为失误的影响

实现持续集成与发布的前提条件-1

- 采用版本控制系统
  - ✓ 所有相关制品都要进行统一的版本管理，包括产品和测试代码、数据库脚本、构建和部署脚本等
  - ✓ 开发人员使用持续集成服务器上的最新版本（通过构建和单元测试、并成功部署到测试环境、类生产环境、生产环境）作为每次开发任务的起点
  - ✓ 通过统一的版本控制系统确保各种软件制品及其变更可追溯、可掌控
- 小步提交
  - ✓ 开发人员频繁（一天多次）提交代码变更到版本库中，确保提交的代码修改较小，减小集成与发布失败的概率
  - ✓ 即使发生失败，也可以快速定位与修复错误并进行回滚
  - ✓ 还可以采用轻量级的代码审查机制，代码提交通过代码质量门禁以及提交者（committer）审核才能进入代码库

实现持续集成与发布的前提条件-2

- 自动化构建与部署
  - ✓ 编写简洁清晰的构建与部署脚本，并制定构建与部署规范，便于标准化、维护和调试构建与部署过程
  - ✓ 实现可重复与高质量的构建与部署
- 自动化测试
  - ✓ 创建全面的自动化测试套件，包括单元测试、组件测试、验收测试等，为软件质量提供自动化保障手段
  - ✓ 及时发现问题，提供及时和有效的反馈
- 构建和测试过程时间短
  - ✓ 确保开发人员能在较短时间内（例如不超过10-15分钟）获得关于构建和测试的反馈并尽快修复问题
  - ✓ 为此需要选择关键的测试用例（例如所有单元测试用例、大部分组件测试用例以及最核心或者涉及刚修改过代码的验收测试用例）作为构建中日常运行的自动化测试用例

实现持续集成与发布的前提条件-3

- 管理开发工作区
  - ✓ 开发人员拥有与构建环境基本一致的开发环境，从而让测试在构建环境中运行前能先在本地相同的环境下运行一遍，降低构建出错的可能性
  - ✓ 为此，需要确保所有代码、脚本、测试数据等都在版本控制库中，第三方库依赖通过集中的方式统一管理
- 团队共识
  - ✓ 开发团队中的每个人都要一致认同小步增量提交代码到主干上的开发方式，以及修复导致构建失败的修改是最高优先级的任务
  - ✓ 需要开发团队在成员的培训和实践指导方面提供必要的投入，建立起基本的团队共识

持续集成与发布的价值-1

- 显著提升开发效率
  - ✓ 早集成，早发布，早发现问题，早解决问题
  - ✓ 减少了人工集成与发布版本的重复过程
- 降低开发风险
  - ✓ 实施持续集成与发布，每天都可以生成可部署的软件版本，避免软件产品最终集成时爆发大量问题
  - ✓ 通过持续集成与发布，缺陷的发现和修复都变得更快，软件的质量也就可以度量
  - ✓ 提供了一张安全网，降低了缺陷引入到生产环节的风险
- 增强团队信心
  - ✓ 团队成员每天都可以看到可部署的软件版本，每次代码提交后都能及时了解自己编写的代码是否遵守编码标准和设计标准，是否通过测试验证，往前迈进的每一步都非常坚实
  - ✓ 可以在任何时间发布可以部署的软件，也增加了客户信心

持续集成与发布的价值-2

- 增强项目的可见性
  - ✓ 持续集成与发布可以真实地反映软件项目的开发进度，例如可以通过特性开发的完成率来反映开发进度
  - ✓ 开发人员每天的工作都立刻合入版本，集成与发布结果快速反馈给项目经理，项目的过程质量一目了然
- 有利于质量要求的落地
  - ✓ 质量管理要求可以嵌入到持续集成与发布工具中自动执行，如自动执行编程规范检查、代码圈复杂度检查、代码重复度检查、模块间依赖关系检查、缺陷扫描等

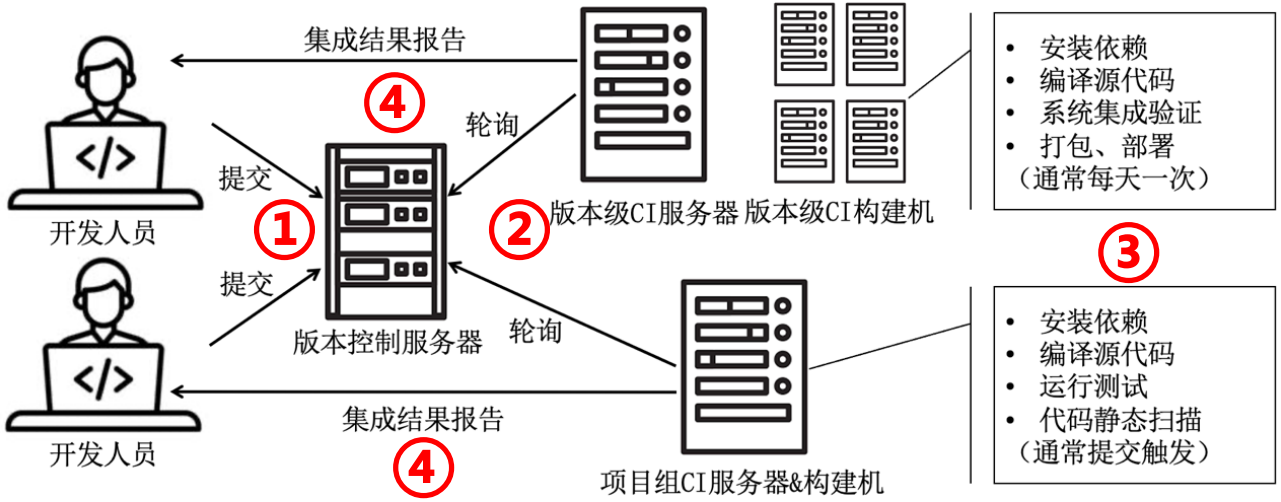
云化与本地持续集成与发布

- 云化（即使用公有云环境）持续集成与发布
  - ✓ 企业不需要专业的IT人员来搭建和维护持续集成与发布所需的基础设施（如存储、网络、CPU、操作系统等）
- 本地（即在企业自有环境中）持续集成与发布
  - ✓ 对于一些传统软件（如嵌入式软件、工业软件等），由于法律合规性、安全性、隐私等问题，企业只能在本地进行持续集成与发布

本地持续集成与发布	云化持续集成与发布
本地	远程
需要本地基础设施	由云服务商提供基础设施
需要专业人员进行维护与技术支持	由云服务商提供维护与技术支持
软件和数据具有自主可控性	软件和数据暴露给云服务商
可定制性高度灵活	可定制性较弱
部署耗时	部署快捷

持续集成

- 团队成员频繁地集成他们的代码变更，通常一天集成多次
- 每次集成都需要自动化的构建和测试来验证集成结果，尽可能快地发现集成错误



持续集成的“六步提交法”-1

- **检出最近构建成功的代码**：当开发人员开始工作时（如认领了一个开发任务），应该将最近一次构建成功的代码从开发团队的开发主干分支上检出到个人工作区
- **修改代码**：开发人员在个人工作区中对代码进行修改（如实现开发任务的代码，编写对应代码的自动化测试用例）
- **第一次个人构建**：当开发工作完成并准备提交时，执行自动化验证集，对个人工作区的新代码执行第一次个人构建，用于验证自己修改的代码质量是否达标

持续集成的“六步提交法”-2

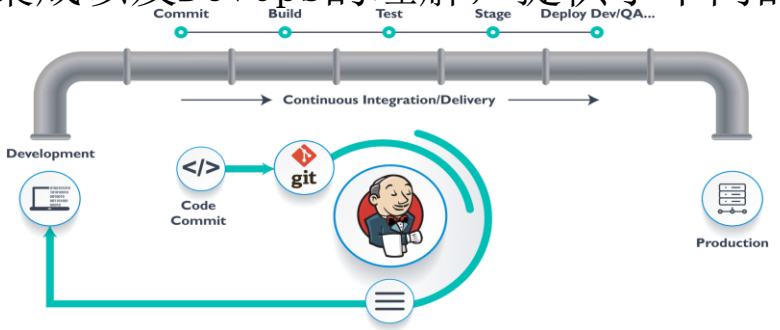
- **第二次个人构建**：将其他成员的新代码与自己本地修改的代码进行合并，并再次执行质量验证，确保自己与其他成员的代码都没有问题
- **提交代码到开发主干分支**：第二次个人构建成功之后，提交代码到开发主干分支
- **提交构建**：持续集成服务器发现这次代码变更提交然后开始执行构建，如果构建失败，则开发人员应该立即着手修复，并通知其他成员禁止再向开发主干分支提交代码，并且不要检出这个版本

持续集成实践

- 基本实践
  - ✓ 构建失败后不要提交新代码、提交前本地运行所有的提交测试或由持续集成服务器完成此事、等提交测试通过后再继续工作、结束当天工作前构建必须是成功状态、时刻准备回滚到前一个版本、在回滚前要规定一个修复时间、不要将失败的测试注释掉、为自己导致的问题负责以及坚持测试驱动的开发
  - ✓ 这些基本实践一般都应当遵循
- 推荐的实践
  - ✓ 采用极限编程的其他开发实践、如果违背架构原则则让构建失败、如果测试运行变慢则让构建失败、如果有编译警告或代码风格问题则让测试失败
  - ✓ 这些实践不是必须的，但有时对于提升开发效率和质量是有帮助的

持续集成的自动化支持工具

- 常用的开源持续集成工具：Jenkins
  - ✓ 提供自动化的测试、构建和部署能力
  - ✓ 支持多种插件，从而在持续集成过程中实现多种测试插件、代码扫描、构建方式等的支持
- 企业自研的持续集成平台
  - ✓ 例如Atlassian的Bamboo、腾讯的Coding平台、华为云软件开发平台DevCloud、阿里巴巴的云效流水线等
  - ✓ 通常提供代码托管，并且支持自动化测试、自动化构建与部署等典型的能力，也结合各自对持续集成以及DevOps的理解，提供了不同的DevOps工具链集成



软件构建

- 通过使用构建脚本自动管理与执行软件项目的依赖、编译、测试、打包等工作，可以极大地提升开发人员效率
- 软件构建过程需要自动化、可重复、防篡改、防植入，为此需要满足以下基本要求
  - ✓ 构建过程自动化：从构建启动开始到构建最终结束，中间过程不能手工干预，使构建变得高效、可靠
  - ✓ 构建脚本简洁清晰：构建脚本也是代码，也需要易于理解、维护和调试，有利于和运维人员更好地协作
  - ✓ 构建标准化：对构建目录结构、构建依赖、构建初始化、构建入口、命名等进行标准化约束，使得所有产品、平台和组件的构建风格保持一致，便于构建管理和维护

构建工具

- 使用构建工具可以更好地管理项目代码、构建输出、以及项目依赖
  - ✓ 大型C、C++工程业界普遍使用CMake、Bazel
  - ✓ Java项目一般使用Maven和Gradle

Maven项目结构

src目录包含项目源代码和资源文件，并将产品和测试代码分别放在main和test子目录下

target目录包含项目构建完成后的输出文件，如class文件以及打包后的包文件等，可以方便地清除前一次的构建结果，实现构建输出的管理

pom.xml用于声明与管理项目的依赖、插件、构建目标、以及相应的配置等信息

项目目录	含义
pom.xml	项目描述文件
src/main/java	项目源代码所在的目录
src/main/resources	项目资源文件所在的目录
src/main/filters	项目资源过滤文件所在 的目录
src/main/webapp	web应用源代码所在的目录
src/test/java	项目测试代码所在的目录
src/test/resources	项目测试相关的资源文件所在的目录
src/test/filters	项目测试相关的资源过滤文件所在目录
src/it	集成测试代码所在目录（供插件使用）
src/assembly	组件描述符所在的目录
src/site	站点文件所在的目录
target	项目构建的输出文件所在的目录
LICENSE.txt	项目的许可证文件
NOTICE.txt	项目依赖的库的注意事项
README.txt	项目的readme文件

依赖管理

- 依赖管理是构建工具的重要能力之一
  - ✓ 传统方式：将依赖库文件手工放置于根目录下的lib目录中
  - ✓ 构建工具：可以按照指定格式显式地声明依赖库及其版本，并在构建时自动从远程或者本地仓库中下载依赖库，便于依赖库的版本管理

Maven项目pom.xml中的依赖声明

声明了对commons-io的2.5版本以及guava的23.0版本的依赖声明，在构建项目时，这两个依赖库会被自动下载到本地的Maven仓库中

构建工具还提供了依赖链解析的能力，便于查看与分析软件项目所直接依赖和间接依赖的所有依赖库，例如：项目依赖了库A，而库A又依赖了库B和库C，那么库A就是软件项目的直接依赖，而库B和库C就是软件项目的间接依赖

在Maven中，可以通过dependency:tree命令来获得软件项目的依赖链信息

```
<dependencies>
  <dependency>
    <groupId>commons-io</groupId>
    <artifactId>commons-io</artifactId>
    <version>2.5</version>
  </dependency>
  <dependency>
    <groupId>com.google.guava</groupId>
    <artifactId>guava</artifactId>
    <version>23.0</version>
  </dependency>
<dependencies>
```

构建规范

- 需要建立并遵循一定的构建规范
- 可参考以下常用的构建基本原则
  - ✓ 构建过程中禁止删除或修改源代码文件及其目录结构
  - ✓ 每个组件提供clean命令
  - ✓ 禁止使用超级管理员用户（例如root账号）和系统用户执行构建，应该使用普通用户账户执行构建
  - ✓ 构建目录结构管理标准化
  - ✓ 构建目录遵从构建工具Maven的约定（针对Java程序）
  - ✓ Maven构建入口为根目录的pom.xml，在根目录下调用mvn命令构建，在使用构建脚本调用mvn命令时入口必须单一，脚本统一命名为build.suffix，且放在根目录的script目录下
  - ✓ 必须使用的javac编译选项：-source，-target，-Xlint:all。 同时，maven-compiler-plugin的showWarnings属性必须设置为true
  - ✓ 构建过程中不能污染Linux操作系统的/usr/bin、/etc等系统目录
  - ✓ 避免构建脚本嵌套过深，禁止超过3层
  - ✓ 构建脚本必须使用相对路径，禁止使用绝对路径

课堂讨论：规范化和自动化构建

课堂讨论：与大家所熟悉的手工构建过程相比，规范化和自动化的构建过程有什么优势？为什么成为很多软件项目的必要选择？

软件发布

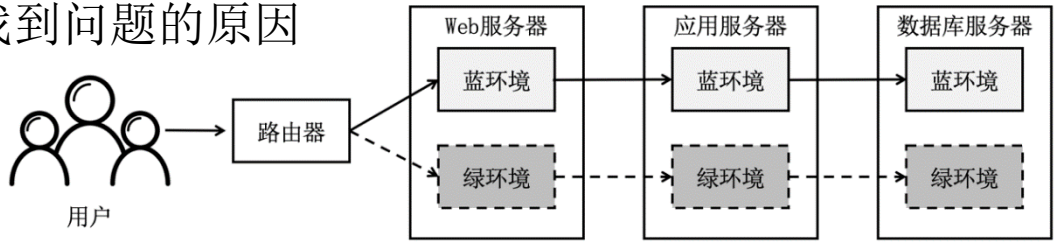
- 持续发布软件能够快速获得客户与用户的反馈，能够便于快速地发现、定位、修复错误，能够降低单次部署的成本
- 容易导致软件发布失败的反模式
  - ✓ 手工部署软件：通过文档描述部署步骤以及容易出错的地方，部署时按照文档进行操作，问题是难以确保部署文档总是最新的同时发布过程耗费长、易出错、难回滚
  - ✓ 开发完成以后才部署：临近交付和上线才会发现一些在开发环境中无法发现的问题，开发人员调试和修复错误的时间紧、压力大
  - ✓ 生产环境手工配置：运维团队通过手工修改配置参数在生产环境中部署软件的发布版本，不确定性高，问题可能重复出现且难以重现和定位

高效软件交付的基本原则

- 创建一个可重复且可靠的过程
  - ✓ 尽可能多地采用自动化地工作，例如自动化地测试、自动化地数据库升级、自动化地网络设置
  - ✓ 将所有涉及到发布和部署的内容纳入版本控制系统的管理，从而让这些自动化脚本的版本变化也能实现追踪管理
- “已完成”（Done）即“已发布”
  - ✓ 以发布和交付作为任务完成的标识，即在生产环境中部署所开发的新程序或者向客户演示并由客户试用了新功能
- 交付过程是每个成员的责任
  - ✓ 不论是开发人员、测试人员还是运维人员，应当尽早地参与到发布软件的过程中来，共同面对交付和部署中的困难
- 内建质量和持续改进
  - ✓ 内建质量来自精益思想，强调在过程中更早发现并解决问题
  - ✓ 定期召开回顾会议进行分析和总结，持续改进发布过程

蓝绿部署（Blue-Green Deployment）

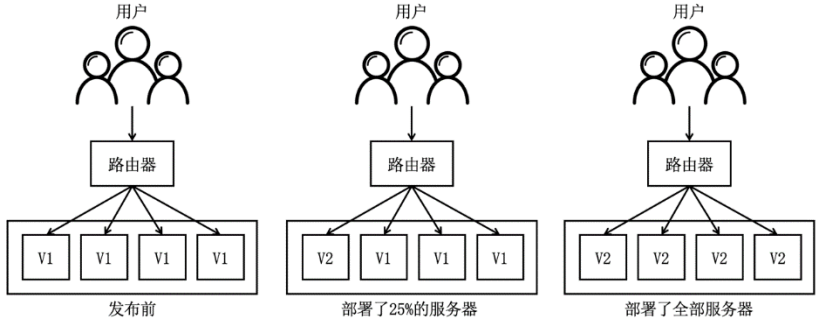
- 一种将用户流量从旧版本应用逐渐转移到新版本的发布方法，通过快速回滚来应对发布错误
- 需要维护两个环境
  - ✓ 旧版本的生产环境（蓝环境），用于对外提供软件服务
  - ✓ 新版本的预发布环境（绿环境），用于对新版本进行测试
- 部署过程
  - ✓ 将新版本发布到绿环境中并进行测试，确认是否正常工作
  - ✓ 如果没问题，修改路由将用户流量从蓝环境引流到绿环境
  - ✓ 如果引流之后出现问题，只需要修改路由再切回到蓝环境，并在绿环境中进行调试，从而找到问题的原因



金丝雀发布（Canary Release）

- 又称灰度发布，具体过程如下
  - ✓ 先引流一部分用户流量到新版本部署中，如果服务这些用户的新版本部署没有问题，那么再逐渐将更多用户流量引流到新版本部署中
  - ✓ 随着时间推移，可对新版本进行增量部署，直到所有的用户流量都引流到新版本部署中
  - ✓ 一旦出现问题，也能支持快速回滚，即通过路由配置禁止将用户流量引流到部署新版本的服务器上
- 能够尽早发现新版本中的问题，而不影响大多数用户

19世纪的矿工井下作业的时候通常会带上一只金丝雀，通过观察金丝雀的状况来判断是否有有害气体



课堂讨论：金丝雀发布

课堂讨论：想象一下金丝雀发布是如何实现的？这种发布方式如何促进软件开发质量和效率以及更新频率的提升？

部署流水线

- 提供了一种从软件开发完成到最终交付到用户手中的“端到端”的过程，提高了软件集成与发布的质量与效率
  - ✓ 实现了软件的构建、部署、测试、发布整个端到端过程的自动化，本质上提供了一个自动化的软件交付过程
  - ✓ 当软件的源代码、配置或者环境发生变化时，都可触发部署流水线建立一个自动化部署实例，使软件经过一系列处理后达到可交付状态
- 部署流水线最终的目标环境是生产环境，即系统为客户提供服务或正式运行的环境
- 也支持在中间步骤中向其他目标环境进行部署
  - ✓ 验收测试环境（用于运行系统的自动化验收测试）
  - ✓ 容量测试环境（用于开展系统的容量测试）
  - ✓ 试运行环境（与生产环境相同的测试环境）

部署流水线的几个阶段

- 提交阶段
  - ✓ 代码提交到版本控制系统后，由部署流水线进行编译、自动化单元测试，从技术角度断言系统是可工作的
  - ✓ 还可能包括对代码进行质量分析以及打包成二进制包
- 自动化验收测试阶段
  - ✓ 将二进制包部署到验收测试环境，开始运行自动化验收测试用例，如果通过则表明系统符合预计的功能需求
  - ✓ 后续可根据构建目标，利用自动化脚本和统一的环境配置，将系统部署到验收测试环境、容量测试环境或试运行环境
- 手工测试阶段
  - ✓ 测试人员通过手工测试发现自动化测试未能发现的缺陷
  - ✓ 也可包括在演示环境中向客户进行演示
- 发布阶段
  - ✓ 通过自动化部署脚本部署到生产环境或者试运行环境上

部署流水线的应用实践

- 部署流水线提供了一种可重复和可靠的自动化系统，将修改的代码尽快上线到生产环境中
  - ✓ 自动化测试能力越完善，完成新版本系统的测试周期越短，那么能交付和部署新版本产品的频率也就可以越高
  - ✓ 即使不是每次修改都要交付并部署到生产环境中，也可以此来推进日常开发工作，让系统随时处在可交付和可部署状态下，由此产生了持续交付和持续部署的实践
- 企业在实践部署流水线时可能设置不同阶段
  - ✓ 发布：建设一个发布平台，所有“可发布”的软件需要发布到发布平台上
  - ✓ 交付：最终向客户交付软件前，只能从发布平台上获取待部署的制品，然后向生产环境部署
- 部署流水线的可信问题：权限管理、防恶意程序植入、全程可追溯等

本章小结

- 采用持续集成、持续交付、持续部署等软件开发实践的收益
  - ✓ 能够实现频繁且自动的软件集成与发布
  - ✓ 可以大大减小集成与发布的出错与调试风险
  - ✓ 加快反馈速度，尽快解决集成与发布过程中发现的问题
- 需要了解以下内容，从而更好地将持续集成与发布付诸实践
  - ✓ 持续集成与发布的前置条件及其价值
  - ✓ 软件构建的基本要求与规范
  - ✓ 软件发布的反模式与基本原则
  - ✓ 软件发布中降低发布风险的常用方法
  - ✓ 典型工具中部署流水线的配置流程

COMP130015.02
软件工程

End

10. 软件集成与发布