Shanghai Jiao Tong University

软件工程

Module: 软件运维

上海交通大学软件工程中心

软件运维

- 软件维护
- DevOps

@第9章.教材

软件维护 “冰山”

- 有关调查结果表明，软件维护是软件工程中最消耗资源的活动，很多软件公司中软件维护的成本已经达到了整个软件生存周期资源的40%到70%，甚至达到了90%。
- 软件系统趋于大型化和复杂化，大多数软件在设计时没有考虑到将来的修改问题，常常还伴有开发人员变动频繁、文档不够详细、维护周期过长等等问题，这些问题导致维护活动的时间与花费不断增加。

维护类型

	纠错	增强
积极的	预防性维护	完善性维护
被动的	纠错性维护	适应性维护

- 纠错性维护（Corrective maintenance）
  - 由于软件中的缺陷引起的修改
- 完善性维护（Perfective maintenance），
  - 根据用户在使用过程中提出的一些建设性意见而进行的维护活动
- 适应性维护（Adaptive maintenance）
  - 为适应环境的变化而修改软件的活动
- 预防性维护（Preventive maintenance）
  - 为了进一步改善软件系统的可维护性和可靠性，并为以后的改进奠定基础

维护的技术问题

- 有限理解（Limited understanding），对他人开发软件进行维护时，如何快速理解程序并找到需要修改或纠错的地方？
- 在软件维护过程中需要大量的回归测试，耗时耗力。
- 当软件非常关键以致不能停机时，如何进行在线维护而不影响软件的运行？
- 影响分析，如何对现有软件的变更进行全面影响分析？
- 如何在开发中促进和遵循软件的可维护性？
  - 易分析性（Analyzability）、易改变性（Changeability）、稳定性（Stability）和易测试性（Testability）

维护成本

- 维护的工作可划分成：
  - 生产性活动 如，分析评价、修改设计、编写程序代码等
  - 非生产性活动 如，程序代码功能理解、数据结构解释、接口特点和性能界限分析等
- 维护工作量的模型

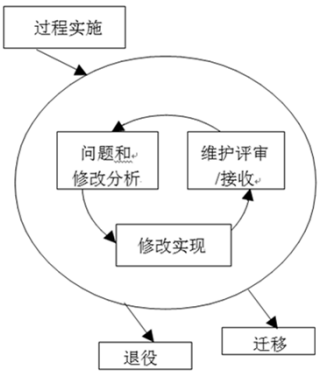
$$M = p + Ke^{c-d}$$

  - M：维护的总工作量；
  - P：生产性工作量；
  - K：经验常数；
  - e：软件的规模；
  - c：复杂程度；
  - d：维护人员对软件的熟悉程度

影响维护成本的因素

- 操作环境：硬件和软件
  - 软件的规模越大、复杂性越高、年龄越大，硬件的能力越低，软件维护的成本和工作量就越大。
- 组织环境：策略、竞争力、过程、产品和个人
  - 软件开发过程和维护过程越规范，采用的设计方法和编程语言模块化程度越高，工程师对软件的熟悉程度越高、能力越强，产品的可靠性和安全性要求越低，软件维护的成本和工作量就越小。

维护过程



维护活动

- 过程实施（Process Implementation）。建立维护过程期间应执行的计划和规程，包括维护计划、维护规程、问题解决规程、用户反馈计划、移交计划、配置管理计划。
- 问题和修改分析（Problem and Modification Analysis）。在修改软件前，要分析修改请求/问题报告，以确定其对组织、现行系统和接口系统的影响，提出可能的方案建议并形成文档，通过核准形成期望的解决方案。
- 修改实现（Modification Implementation）。根据计划和方案更新相应的需求、设计和代码，并进行测试等软件验证工作。

- 维护评审/接收（Maintenance Review/Acceptance）。对上述的维护进行评审，以确保对软件的修改是正确的，并且这些修改是使用正确的方法按批准的要求完成的。
- 迁移（Migration）。在软件的生存周期期间，如果需要将它迁移到一个新环境，则应制订迁移计划、通告用户迁移、提供迁移培训、把软件迁移至新环境、通告迁移完成情况、评估新环境的影响、并进行旧软件和数据的归档。在迁移实施时，旧环境和新环境可以并行进行工作，以便平稳迁移到新环境。
- 退役（Retirement）。软件一旦结束使用生存周期，必须退役。退役时，要制定退役计划、通知用户退役、提供退役培训、实施退役、通告退役完成情况、并进行旧软件和数据的归档。在制定退役计划时，要分析退役的成本和影响、决定是局部还是全部退役、是否用新软件来代替退役软件等。

软件维护技术

- 程序理解
- 逆向工程（Reverse Engineering）
- 再工程（Reengineering）

程序理解

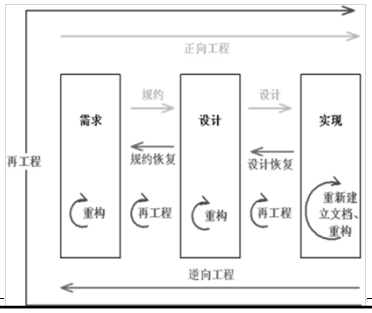
- 软件维护的总工作量大约一半被用在理解程序上。
- 程序理解通过提取并分析程序中各种实体之间的关系，形成系统的不同形式和层次的抽象表示，完成程序设计领域到应用领域的映射。
- 程序员在理解程序的过程中，经常通过反复三个活动——阅读关于程序的文档，阅读源代码，运行程序来捕捉信息。
- 程序理解工具
  - 基于程序结构的可视化工具，通过分析程序的结构，抽取其中各种实体，使用图形表示这些实体和它们之间的关系，可以直观地为维护者提供不同抽象层次上的信息。
  - 帮助维护者导航浏览源代码，为浏览工作提供着眼点，缩小需要浏览的代码范围。

逆向工程

- 逆向工程是分析软件，识别出软件的组成成份及其相互的关系，以更高的抽象层次来刻画软件的过程，它并不改变软件本身，也不产生新的软件。
- 逆向工程主要分为以下几类:
  - 重新文档化（redocumentation）：分析软件，改进或提供软件新的文档。
  - 设计恢复（design recovery）：从代码中抽象出设计信息；
  - 规约恢复（specification recovery）：分析软件，导出需求规约信息；
  - 重构（refactoring, restructuring）：在同一抽象级别上转换软件描述形式，而不改变原有软件的功能；
  - 数据逆向工程（data reverse engineering）：从数据库物理模式中获取逻辑模式，如实体关系图。

再工程

- 再工程是在逆向工程所获信息的基础上修改或重构已有的软件，产生一个新版本的过程，它将逆向工程、重构和正向工程组合起来，将现存系统重新构造为新的形式。



讨论

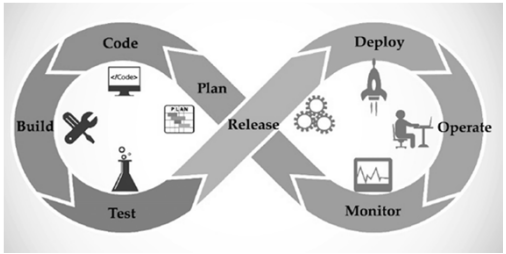
- 一个大型大学有一个大型计算机系统，用于存储和管理所有学生和教职工的信息。该系统：已经使用了25年，它采用Cobol结构化程序设计技术开发，并与关系数据库通信；它运行在一台IBM主机上；有50多万行代码。该系统已经进行过多次修改，既有经过策划的修改，也有快速修改，现在维护的成本过高。认识到有这些问题，大学希望利用面向对象的开发优势，但是不幸的是，维护这个系统的90%以上的员工都是新人，并不熟悉系统的实现。
- 如何办？

软件运维

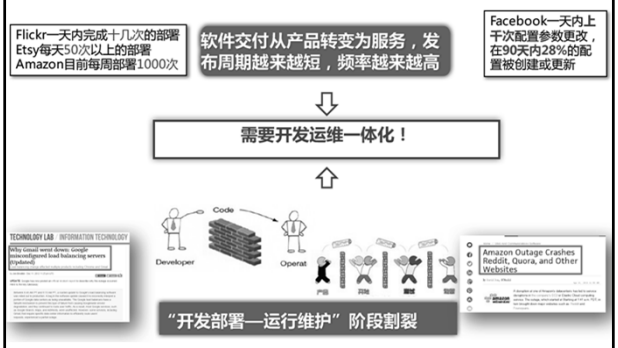
- 软件维护
- DevOps

DevOps

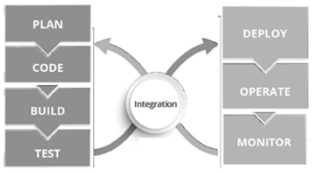
- DevOps（Development & Operations）开发运维一体化
  - （持续交付）快速实现一行代码的变更，到软件交付到用户手中
  - （自动化）从代码提交到最终交付用户只需要按下按钮，自动化每一个工作环节，及时收到用户反馈



背景



DevOps过程



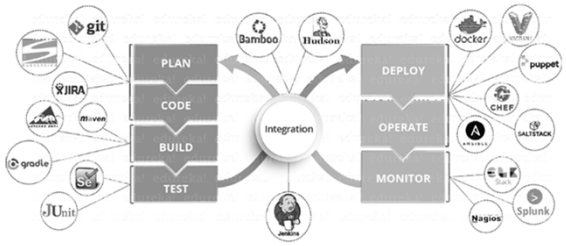
Plan：项目规划、设计等　　Code：代码开发和审阅，版本控制等

Build：项目构建、打包　　Test：单元测试、集成测试、负载测试等

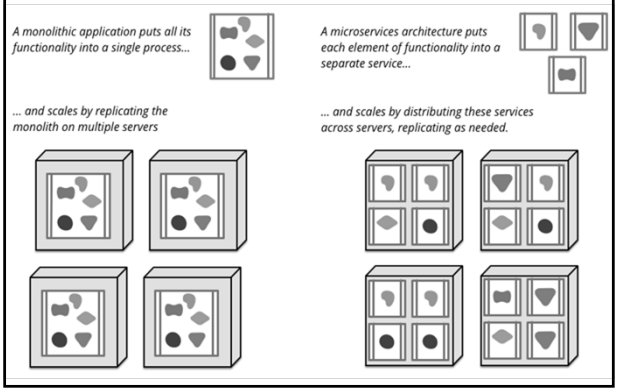
Deploy：应用的配置与部署　　Operate：应用的再配置、再部署、升级等

Monitor：应用运行时监控、最终用户体验

DevOps工具集



单体应用 Vs. 微服务应用



容器

- 微服务同样存在一些劣势：
  - 因为服务通常部署在多个主机上，很难持续跟踪指定服务究竟运行在某台主机上。
  - 因为微服务架构使用的主机容量往往小于Monolithic架构，随着微服务架构不停的横向扩展，主机数量将以一个非常恐怖的速度增长。
- 解决方案：容器
  - 不同容器共享相同的内核，容器的共享和发布非常简单
  - 容器之间进行了完全的隔离，简易了不同语言开发的微服务代码部署
  - 例如： Docker、 Kubernetes等