【软件构造】第二章第一节 软件生命周期和版本控制（配置管理）

软件构造第二章 第一节 软件生命周期和版本控制（配置管理）

第一章解释了软件构造的对象、结果和评判标准之后，我们将会了解到软件是如何从0开始被开发出来的。

Outline

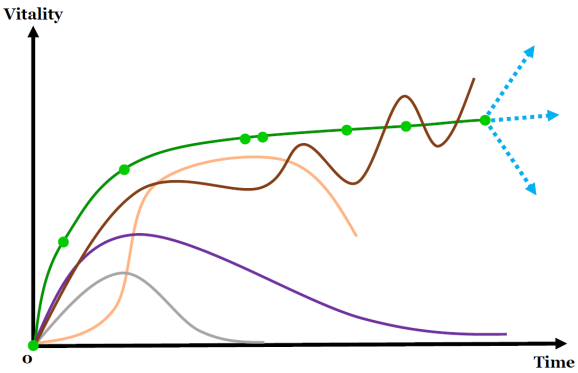
- 软件周期
- 经典软件过程模型
- 敏捷开发
- 协同软件开发
- 配置管理（SCM）
- 版本控制（Git）

Notes

软件生命周期

【生命周期】

- 两种形态
  - 从0到1：SDLC
    - 策划阶段：获取需求、制定计划
    - 架构师：系统分析（业务领域，what）、软件设计（语言、架构，how）
    - 编码实现、测试
    - 维护直至消失
  - 从1到n：运用版本控制技术实现迭代更新
- 软件还活着的标志：Age and vitality(活力)
- 我们期待生命周期长而且具有较高活性的软件，但开发失败、软件老化、需求不及需求是软件死亡的主要原因

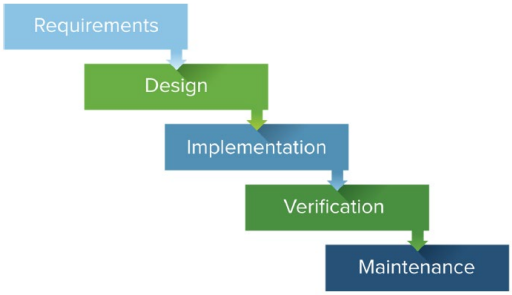


经典软件过程模型（侧重于计划）

- 大体上分为两类，线性和迭代（迭代大体上就是线性上增加反馈）
- 关键考虑指标：用户参与度、复杂度、软件质量

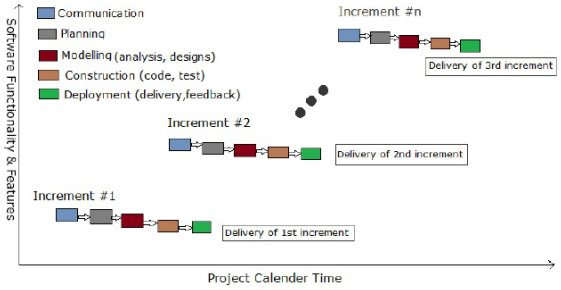
【瀑布模型】

- 瀑布模型将软件生存周期的各项活动规定为依固定顺序而连接的若干阶段工作；
- 瀑布模型规定了每一个阶段的输入，以及本阶段的工作成果，作为输出传入下一阶段；
- 早期主流开发过程，适用于需求稳定的项目；
- 优点：有设计前的规约和编码前的设计，易于管理；
- 缺点：应对变化时，成本十分高。



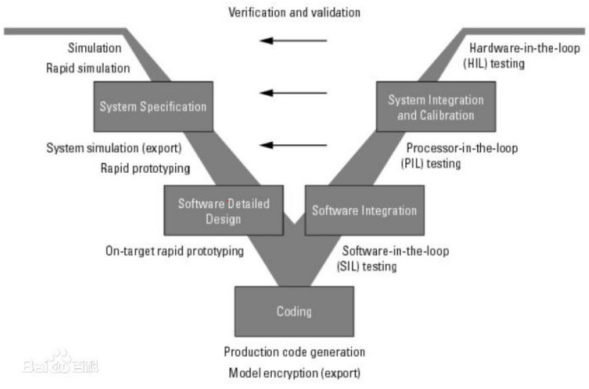
【增量模型】

- 运用分治的思想，将需求分段，成为一系列增量产品，每个增量内部仍使用瀑布模型；
- 增量模型是瀑布模型的变形，拥有后者的全部优点，此外可以很快的迭代出第一版本；
- 选择最核心需求首先实现显得十分重要。



【V模型】

- 对瀑布模型的改进
- 强调测试与继承，对代码、分析文档进行质量保证



【原型法】

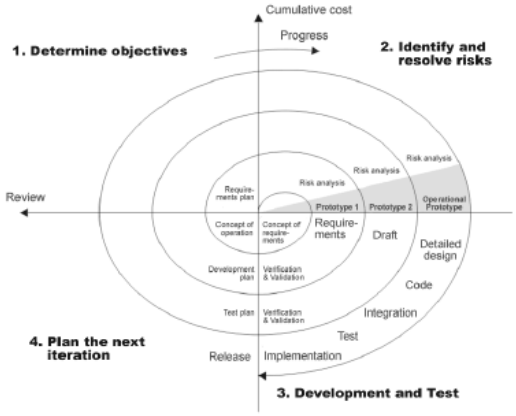
- 迭代法， 原型法是指在获取一组基本的需求定义后，利用高级软件工具可视化的开发环境，快速地建立一个目标系统的最

初版本，并把它交给用户试用、补充和修改，再进行新的版本开发。反复进行这个过程，直到得出系统的“精确解”，即用户满意为止。

- 其核心是用交互的，快速建立起来的原型取代了形式的、僵硬的（不允许更改的）大部分的规格说明，用户通过在计算机上实际运行和试用原型系统而向开发者提供真实的、具体的反馈意见。
- 优势：
  - 软件设计者和实施者可以在项目早期从用户那里获得有价值的反馈。
  - 客户可以比较软件制作的软件是否符合软件规范。
  - 它还使软件工程师能够深入了解初始项目估算的准确性以及提出的最后期限和里程碑是否可以成功实现

【螺旋模型】

- 采用一种周期性的方法来进行系统开发。
- 优点：
  - 设计上的灵活性,可以在项目的各个阶段进行变更。
  - 以小的分段来构建大型系统,使成本计算变得简单容易。



敏捷软件开发

【宣言】

- 人的作用 胜于 过程管理和工具的使用（结对编程）
- 可运行的软件 胜于 面面俱到的文档
- 客户合作 胜于 合同谈判
- 响应变化 胜于 遵循计划

【12个原则】

1. 我们最优先要做的是通过尽早的、持续的交付有价值的软件来使客户满意
2. 即使到了开发的后期，也欢迎改变需求。敏捷过程利用变化来为客户创造竞争优势。
3. 经常性的交付可以工作的软件，交付的间隔可以从几周到几个月，交付的时间间隔越短越好。
4. 在整个项目开发期间，业务人员和开发人员必须天天都在一起工作。
5. 围绕被激励起来的人个来构建项目。给他们提供所需要的环境和支持，并且信任他们能够完成工作。
6. 在团队内部，最具有效果并且富有效率的传递信息的方法，就是面对面的交谈。
7. 工作的软件是首要进度度量标准。
8. 敏捷过程提可持续的开发速度。责任人、开发者和用户应该能够保持一个长期的、恒定的开发速度。
9. 不断地关注优秀的技能和好的设计会增强敏捷能力。
10. 简单----使未完成的工作最大化的艺术----是根本的。
11. 最好的构架、需求和设计出自与自组织的团队。

12. 每隔一定时间，团队会在如何才能更有效地工作方面进行反省，然后相应地对自己的行为进行调整。

【核心特点】

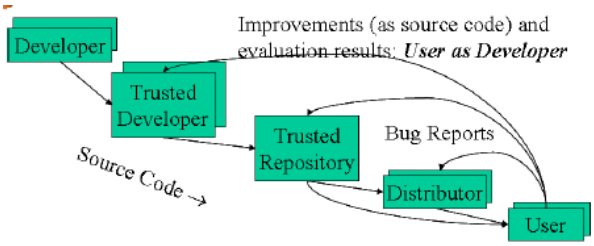
从需求与过程驱动 变为 由成果驱动

【极限编程】

- 描述需求（利用story，情景对话表达用户需求）
- 设计阶段：做原型
- Coding：（TDD）测试驱动开发、结对编程、自动构建
- 测试阶段：持续集成、持续发布
- 冲刺模型
  - 项目管理方式：任务墙、目标图

协同软件开发

有关协同软件开发请参考 冬刻忆的博客



配置管理和版本控制

有关配置管理和版本控制的内容请参考 能量启示录的博客

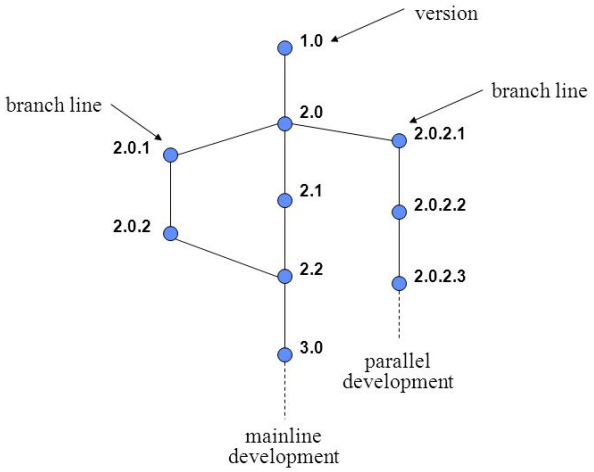
- 版本控制是实现软件配置管理的最主要工具之一，此外，建立基线也是十分重要的工具
- 软件配置项： 软件生存周期各个阶段活动的产物经审批后即可称之为软件配置项，其包括文档、源代码、可重用软件等。
- 基线：
  - 定义：软件文档或源码(或其它产出物)的一个稳定版本,它是进一步开发的基础
  - 建立基线的原因： 重现性、可追踪性和报告。
- 配置管理数据库：配置管理数据库是指这样一种数据库，它包含一个组织的IT服务使用的信息系统的组件的所有相关信息以及这些组件之间的关系。配置管理数据库提供一种对数据的有组织的检查和从任何想要的角度研究数据的方法。
- 对库进行检入、检出：权限封印和权限解锁

【版本控制系统的优点】

- 回到历史版本作参考、作比较
- 项目迁移
- 方便版本合并（最好有相同的基线）
- 具有日止功能，便于开发团队的沟通交流

【分支】

- 部分人员并行开发有意义的活动
- 其他人员不想在新功能完成之前插入新功能



【版本操作系统】

- 本地的VCS
- 集中式VCS（CVS、SVN）：通过服务器进行共享，客户端可以是全集或子集（Git只能是全集）
- 分布式VCS（Git）：用户之间可以直接进行推送，也可以通过云

版本控制工具—— Git

关于Git的使用请参考

1.Git使用教程

2.追着太阳晒的博客

- Git的整体架构——四个仓库（本地有三个）
  - 工作目录
  - 暂存区域（在menmory中，对用户不可见）（隐藏的.git文件夹中的stage）
  - 本地库：源代码
  - 云端软件服务器（远程仓库）
- 利用对象图结构，
  - 每个结点保存：父结点、如提交时间的信息
  - VCS还原差异，Git保存完整文件
  - Git对于重复文件，不复制文件，只修改指针
  - 减少冗余
  - 访问速度快
- 分支代码
  - git(创建) branch(切换) -b(branch) iss53
  - git merge hitfix（合并）
  - 是用`git add`把文件添加进去，实际上就是把文件修改添加到暂存区；
  - 用`git commit`提交更改，实际上就是把暂存区的所有内容提交到当前分支。

关于Git的分支代码的详细解析请参考https://www.2cto.com/kf/201607/529363.html

- 本地库和远程库
  - clone：将整个库完整的复制
  - fetch：将某一分支复制下来

push：将分支推送到服务器上

- pull：将某一分支复制下来并合并在当前分支上