Shanghai Jiao Tong University

软件工程

Module: 编码和版本管理

上海交通大学软件学院

编码的目的和质量要求

模块的构件级设计（不可执行的） —编码→ 源程序（可执行的）

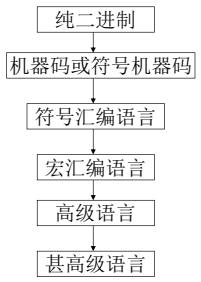
- 程序设计语言的特性和程序设计风格会深刻地影响软件的质量和可维护性。
- 为了保证程序编码的质量，程序员必须深刻理解、熟练掌握并正确地运用程序设计语言的特性。此外，还要求源程序具有良好的结构性和良好的程序设计风格。

编码和版本管理

- 程序设计语言
- 编码准则和规范
- 软件版本管理
- 软件持续集成

@第12.3.4节.教材

计算机上语言的层次



Byte或word，指令、数据不分

用一些符号来代表指令，如sub代表减，Add代表加等，机器地址用十进制。有时汇编语言等同于符号机器码。

变量名用符号，地址也可用符号而非数字。编制的程序称为汇编语言程序。

用户可以定义新指令及子程序

源程序编译为目标程序，或解释执行

高级程序语言的高层规约语言，提供比高级程序语言更高级的语言设施。又称为“可执行的规约语言”。有时也不区别于高级语言。

高级程序设计语言的分类

- 说明式（*declarative*）语言
  - 函数式 Lisp/Scheme, ML, Haskell, Clean, Erlang, Miranda…
  - 数据流 Id, Val …
  - 逻辑式 或基于约束的 Prolog, spreedsheets …
  - 基于模板的 XSLT …
- 命令式（*imperative*） 语言
  - 冯. 诺伊曼 C, Ada, Fortran …
    - 脚本式 Perl, Python, PHP…
  - 面向对象 Smalltalk, Eiffel, C++, Java …

语言的选择

- 选择编码语言的标准
  - 应用领域
  - 算法、计算、数据结构的复杂性
  - 运行效率、开发效率、可移植性等的考虑
  - 库

语言选择举例

- 如果编写对性能要求苛刻，或和操作系统结合紧密的程序，必然选择c。
- 如果需要跨平台，又要广泛的支持的话，选择java。
- 如果编写大程序，可能的化尽量用python，不行了再用java和c。因为python带来了生产力。
- 编写文本的处理程序用perl。在linux下，最方便的工具语言是perl，它有强大的社区和代码库的支持。
- 编写知识的处理程序用prolog。
- 编写最灵活，最模糊的程序用lisp。
- 编写office程序用vba。
- 编写服务器端程序，用php、perl、python、asp、jsp、Java。
- 编写数据库程序最简单的语言是vb或delphi。
- 编写机器学习软件采用python、R、Java
- 如果只作为简单应用的工具语言，python和ruby是更好的选择，他们的跨平台移植性好，应用也比较广泛。

编码和版本管理

- 程序设计语言
- 编码准则和规范
- 软件版本管理
- 软件持续集成

编码准则－1

- Preparation. Before you write one line of code, be sure you:
  - Understand of the problem you're trying to solve
  - Understand basic design principles and concepts.
  - Pick an appropriate programming language.
  - Select an appropriate programming environment and tools.
  - Create unit tests for each component you plan to create.

编码准则－2

- Coding. As you begin writing code, be sure you:
  - Follow structured programming practice.
  - Select data structures that will meet the needs of the design.
  - Create interfaces consistent with the software architecture.
  - Keep conditional logic as simple as possible.
  - Create nested loops in a way that makes them easily testable.
  - Select meaningful variable names and follow other local coding standards.
  - Write code that is self-documenting.
  - Create a visual layout that aids understanding.

编码准则－3

- Validation. After you've completed the first pass, be sure you: 演示
  - Conduct a code walkthrough when appropriate.
  - Perform unit tests and correct errors you've uncovered.
  - Refactor the code.

重构

编码的风格

- 追求“聪明”和“技巧” → 提倡“简明”和“直接”
- 使用标准的控制结构
- 清晰的前提下求取效率
  - Make it right before you make it faster.
  - Make it clear before you make it faster.
  - Keep it right when you make it faster.
    （求快不忘保持程序正确）
  - Keep it simple to make it faster.
    （保持程序简单以求快）
  - Don't sacrifice clarity for “efficiency”.
    （书写清楚，不要为“效率”牺牲清楚）

源程序的文档化 （code documentation）

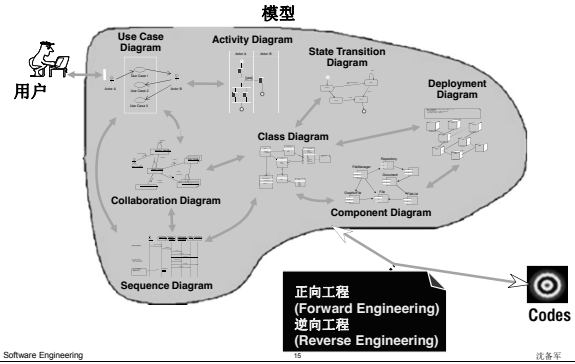
- 有意义的变量名称
- 适当的注释
- 标准的书写格式
  - 用分层缩进的写法显示嵌套结构的层次；
  - 在注释段与程序段、以及不同程序段之间插入空行；
  - 每行只写一条语句；
  - 书写表达式时，适当使用空格或圆括号等作隔离符；

好的源程序本身就是详细设计文档！

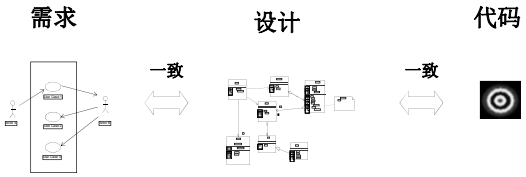
编码规范

浏览C++/Java 的编码规范

双向工程



确保模型和代码的一致



著名的竞赛

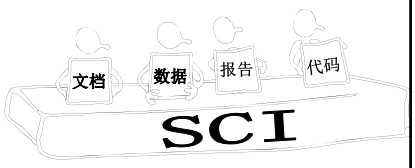
- ICSE软件工程竞赛 **SCORE**
  - http://score-contest.org/
  - 按软件工程开发一个软件
- ACM ICPC
  - http://icpc.baylor.edu/icpc/ 算法
- topcoder
  - http://www.topcoder.com/ 编程之美
- ACM研究竞赛 SRC: Student Research Competition
  - http://www.acm.org/src/
  - 以学术论文来参加竞赛
  - 论文全部发表在各国际会议或杂志上

编码和版本管理

- 程序设计语言
- 编码准则和规范
- 软件版本管理
- 软件持续集成

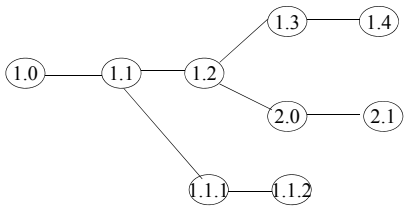
软件配置项

- 软件版本管理的基本实体是软件配置项(Software Configuration Item, SCI)。
- 一个软件配置项是在软件生存周期所产生或使用的一个工作产品或一组相关的工作产品，包括代码、文档、模型、数据等。



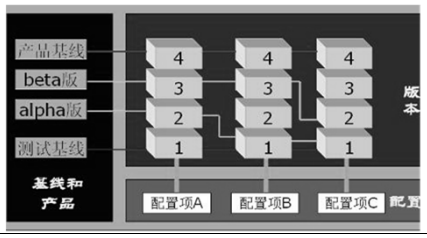
版本 version

- 配置项在软件开发过程中会不断变更，形成多个版本(Version)。
- 每个配置项的版本演化历史可以形象地表示为图形化的版本树(Version Tree)。



基线 baseline

- 基线是项目储存库中每个配置项版本在特定时期的一个“快照”，它提供一个正式标准，随后的工作基于此标准，并且只有经过授权后才能变更这个标准。
- 通常将交付给客户的基线称为一个“发布”（Release），为内部开发用的基线则称为一个“构建”（Build）。

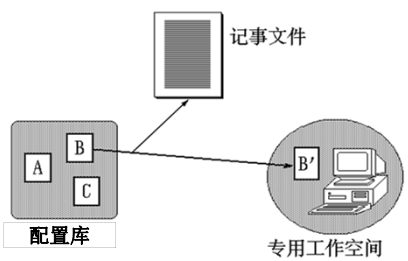


版本控制

- 版本管理记录每个配置项的变更履历，并控制基线的生成。
- 目的：
  - 对软件开发过程中配置项的各个版本提供有效的追踪手段，保证在需要时可回到旧版本，避免文件丢失和相互覆盖；
  - 通过对版本库的访问控制避免未经授权的访问和修改，达到有效保护软件资产和知识产权的目的；
  - 实现团队并行开发、提高开发效率。

配置库CM Repository

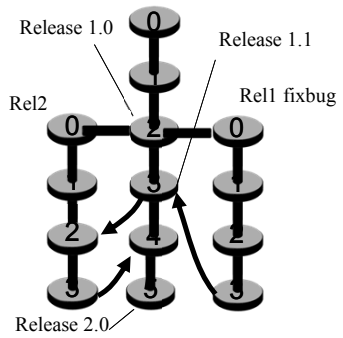
- 存储配置管理信息及软件软件配置项的版本信息。



Check-in和Check-out

- 软件配置项通过检入(Check-in)，进入配置库，开始“冻结”
- 由于各种原因需要变更，从配置库中检出（Check-out）配置项
- check in和check out通过加锁协调多用户操作
- 每次check in时，在配置库上都会生成新的版本

并行开发

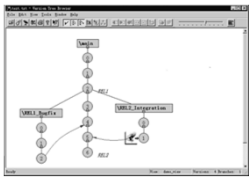


版本控制的最佳实践

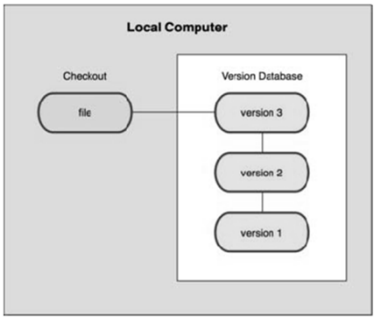
- 合理组织项目及子项目结构
- 避免多人Check-Out
- 合理管理权限
- 避免长时间不Check-in以及不Get Last Version
- 避免强行修改未Check-Out的本地文件的Read-Only属性
- 建议代码Check-in之前需通过单元测试
- 每天或定期备份所有数据

版本控制工具

- Merant PVCS
- IBM Rational Clearcase
- Microsoft Visual Source Safe
- CVS (Freeware)
- SVN (Freeware)
- Gitlab(Freeware)
- GitHub (云平台)
- …

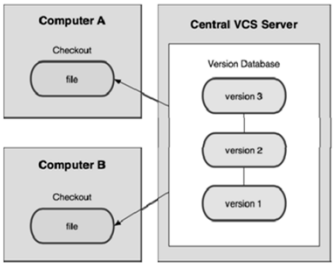


本地版本控制系统



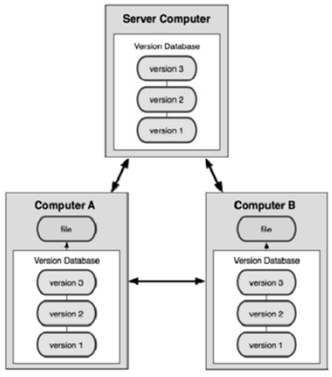
集中化的版本控制系统

- CVS，Subversion 以及 Perforce 等，有一个单一的集中管理的服务器，保存所有文件的修订版本，而协同工作的开发者通过客户端连到这台服务器，取出最新的文件或者提交更新。



分布式版本控制系统Git

- 客户端并不只提取最新版本的文件快照，而是把原始的代码仓库完整地镜像下来。
- 任何一处协同工作用的服务器发生故障，事后都可以用任何一个镜像出来的本地仓库恢复。
- 每一次的提取操作，实际上都是一次对代码仓库的完整备份。



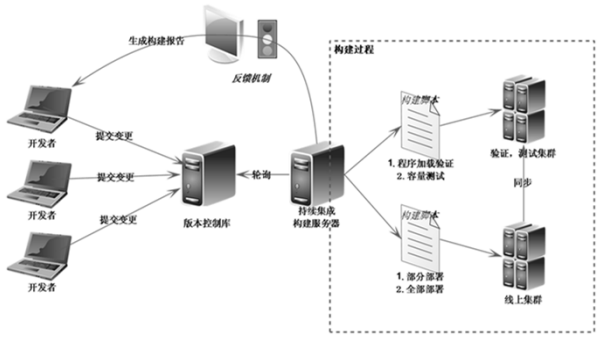
编码和版本管理

- 程序设计语言
- 编码准则和规范
- 软件版本管理
- 软件持续集成

持续集成概念

- 持续集成是一种软件开发实践，即团队开发成员经常集成它们的工作，通常每个成员每天至少集成一次，也就意味着每天可能会发生多次集成。
- 每次集成都通过自动化的构建（包括编译，发布，自动化测试)来验证，从而尽快地发现集成错误。

持续集成流程



持续集成示例