2020年6月18日星期四

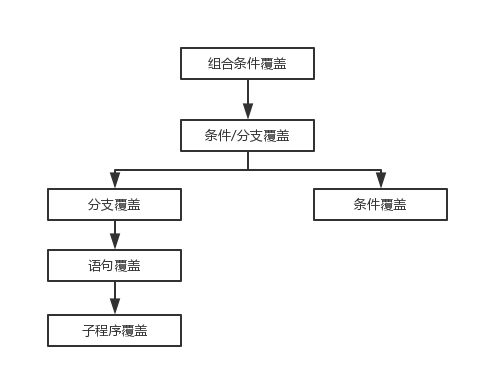
* 避免过长的子程序
* 合并重复代码
* 合并两个类
* 让if语句更短
* 不适用全局变量
* 让类的关系更合理
  + 避免同时修改多个类
  + 降低类之间的耦合度
  + Is-a and has-a
* 使用结构体保存数据
  + 当不测试的代价比测试更大时，我们就要进行软件测试。
  + 单元测试（unit test）
    - 是针对程序模块（软件设计的最小单位）来进行正确性检验的测试工作。
    - 单元测试往往由开发人员自己编写，软件开发人员修改一次代码通常就要进行至少一次单元测试，并且往往使用的开发语言或框架会提供现成的单元测试工具。
    - 使用断言进行单元测试
* 白盒测试
  + 白盒测试是一种考虑系统或组件内部机制的测试方法，常被程序开发者用于对代码进行自测。比如我们前面提到的单元测试，就需要软件开发者自己设计单元测试代码，并执行测试。
  + 除了单元测试外，白盒测试还包括代码审查、集成测试等常用的测试方法。
  + 代码审查（code review）
    - 是指对程序进行系统化地审查，往往通过同事或同行评审的方式进行。
    - 目标是找出并修正代码中的错误及不符合规范的地方，提升软件质量和可维护性，减少之后发现错误的几率。
    - 代码审查有很多种形式，包括结对编程、非正式的代码阅读和正式的代码审查。
  + 集成测试（integration testing, I&T）
    - 是一种将若干独立的软件模块组装起来进行测试的方法。
    - 集成测试往往会在单元测试之后，测试的是多个模块之间的接口，包括模块之间的调用关系和同步控制、模块之间的通信、第三方中间件等。
  + 在进行单元测试时，我们往往使用代码覆盖（code coverage）来确保源代码被测试完全。
    - 根据预期覆盖程度的不同，有如下一些对应的覆盖准则：
    - 子程序覆盖（function coverage）：测试用例是否调用了程序的每一个子程序。
    - 语句覆盖（statement coverage）：测试用例是否执行了程序的每一条语句。
    - 分支覆盖（branch coverage）：测试用例是否能让程序中的每个判定至少取值为true、false各一次。
    - 条件覆盖（condition coverage）：测试用例是否能让程序中每个判定的每个条件都至少取值为true、false各一次。
    - 条件/分支覆盖（condition/decision coverage）：同时满足分支覆盖和条件覆盖。
    - 组合条件覆盖（multiple condition coverage）：测试用例使得每种条件取值组合都被执行一次。

2020年6月19日星期五

* 黑盒测试
  + 是一种无需观察程序内部，只通过给定的输入和输出对程序进行测试的方法。
  + 黑盒测试由测试人员而非开发人员执行



* + 优势
    - 测试人员无需了解程序内部代码实现
    - 测试用例不依赖程序内部的设计
    - 可以从用户的角度出发进行测试
  + 缺点
    - 代码中很多隐藏的缺陷难以被发现



* + - 单元测试流程
      * 选择预期覆盖程度对应的覆盖准则；
      * 选择测试路径以满足选定的测试准则；
      * 根据测试路径设计测试用例的输入数据；
      * 根据输入数据确定预期输出数据，完成单元测试的开发。