软件工程

第17章 软件测试策略

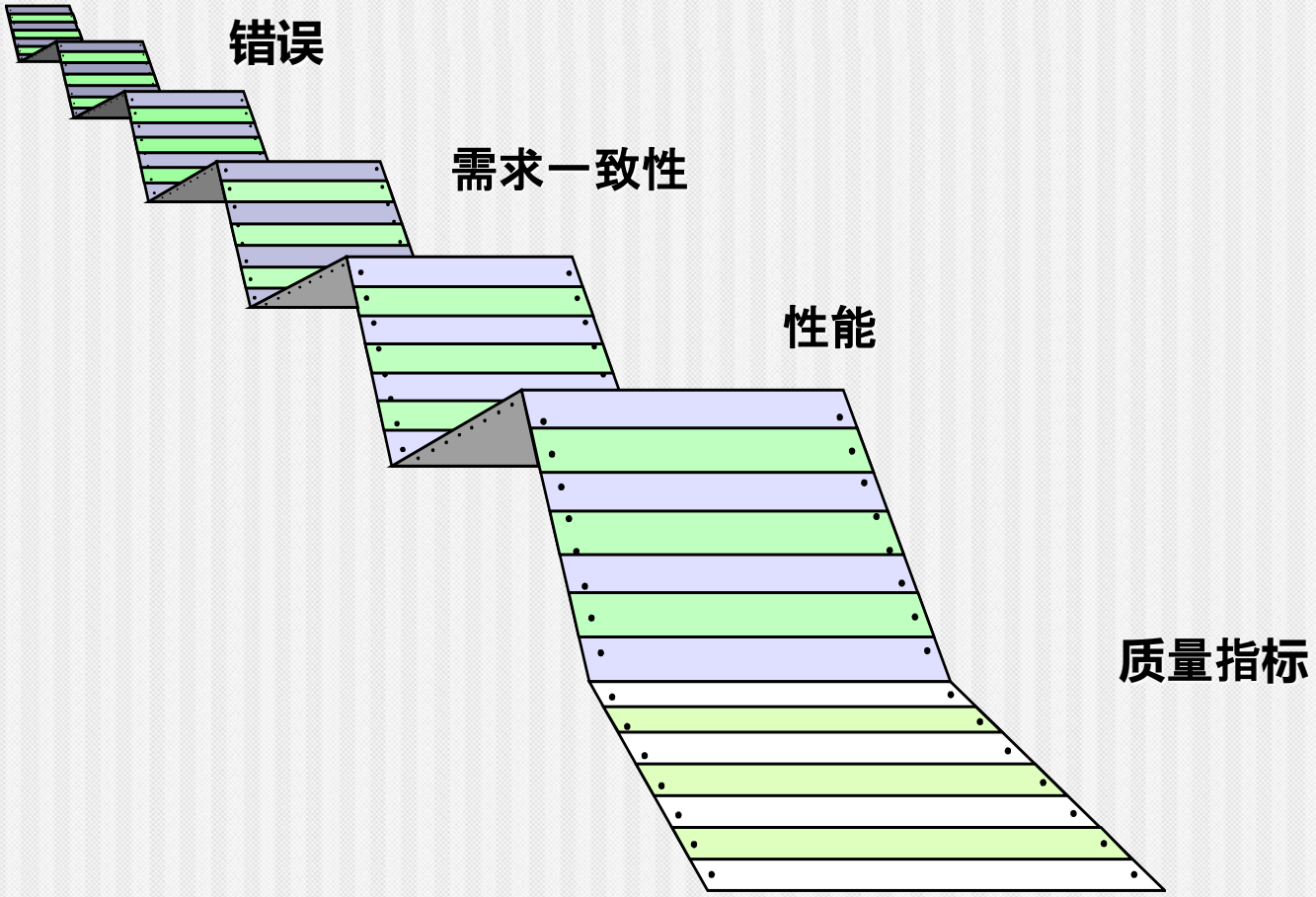
徐本柱 软件学院

2018-10

主要内容

- 软件测试的策略性方法
- 策略问题
- 传统软件的测试策略
- 面向对象软件的测试策略
- 确认测试
- 系统测试

测试显示的内容



概念

- 为了发现软件设计和实现过程中的疏忽所造成的错误。
- 制定测试策略说考虑的问题：
  - 如何进行测试？
  - 是否应该制定正式的测试计划？
  - 是将整个程序作为整体测试，还是只测试其中的一部分？
  - 向大型系统加入新的构件时，是否重新测试已经测过的部分？
  - 什么时候需要客户参与测试工作？

测试是将软件发布给最终用户之前，以发现错误为特定目的运行程序的过程。

重要性和步骤

- 测试所花费的工作量经常比其他任何软件工程活动都多
  - 若测试是无计划地进行，既浪费时间，又浪费不必要的劳动
  - 甚至更糟的是，错误会依然存在。
- 为测试软件建立系统化的测试策略是合情合理的。
- 测试步骤：
  - 测试从“小规模”开始，进展到“大规模”
  - 早期测试关注单个/组构件，发现其逻辑错误（单元测试）
  - 当单个构件测试完，需将构件集成直到建成系统（集成测试）
  - 执行一系列的高阶测试以发现在满足顾客需求方面的错误
  - 随着错误的发现，必须利用调试过程进行诊断和纠正
- 测试规格说明和评审

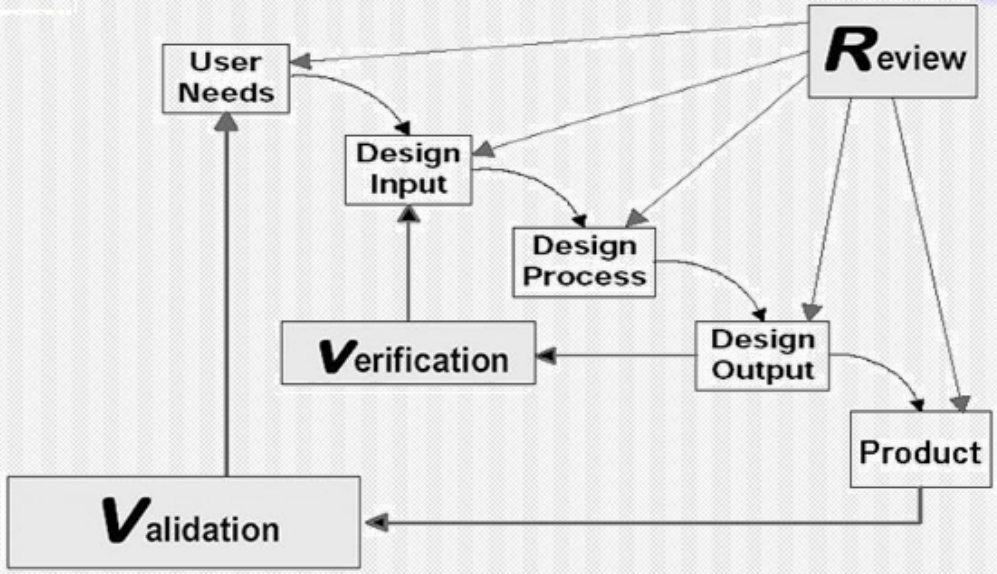
17.1 软件测试的策略性方法

- 为完成有效的测试，应该进行有效的、正式的技术评审，这样许多错误可以在测试开始之前排除。
- 测试始于构件级，并向外延伸到整个计算机系统集成中
- 不同的测试技术适合不同的软件工程方法和不同的时间点
- 测试→软件开发者或独立测试组执行（大型项目）
- 测试和调试是不同的活动，但任何测试策略都包括调试
- 软件测试策略必须提供
  - 用来验证小段源代码是否正确实现的必要的低级测试，
  - 用来确认系统的主要功能是否满足用户需求的高级测试。

17.1.1 验证与确认（V & V）

*验证（Verification）*是指确保软件正确实现特定功能的任务集合。

*确认（Validation）*是指不同的任务集，可以确保已经构建的软件可以追溯到客户的需求。



验证：我们在正确地构造产品吗？（注重过程）
确认：我们在构造正确的产品吗？（注重结果）

17.1.2 软件测试的组织

谁来测试软件?

开发人员

了解系统

但是会“温和地”进行测试，
并受到“移交”的驱动。

独立测试组(ITG)

必须学习系统

17.1.3 软件测试策略——宏观

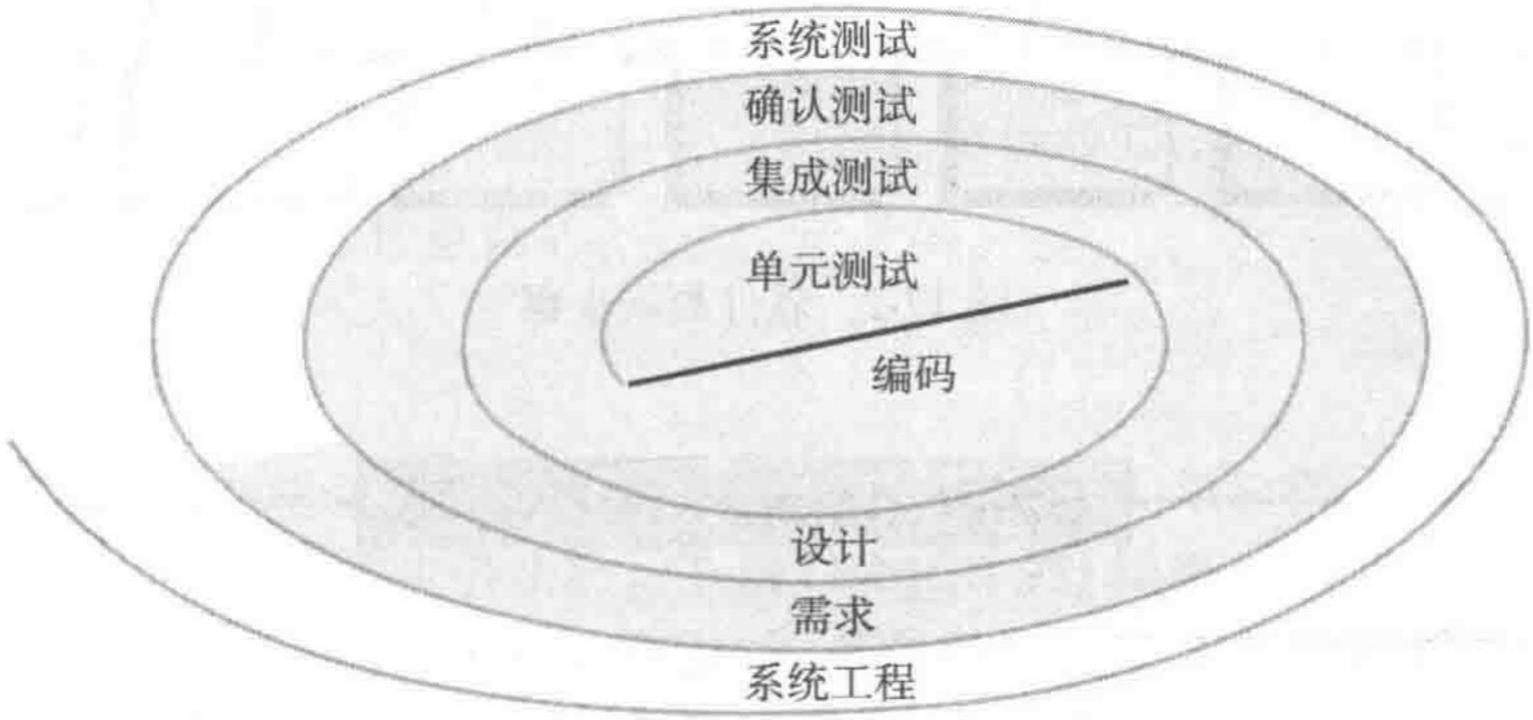
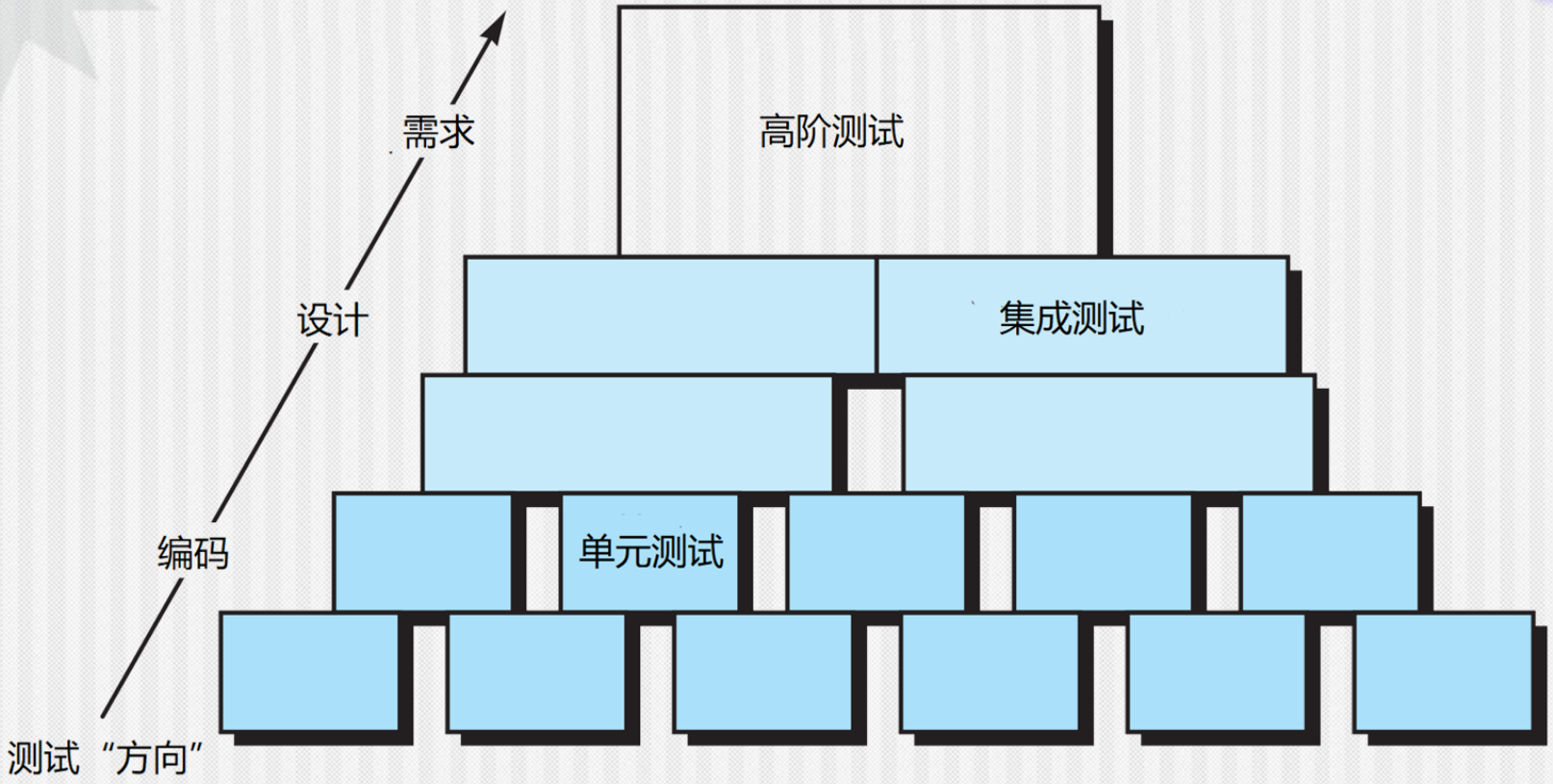


图 17-1　测试策略

17.1.3 软件测试策略——宏观



17.1.4 测试完成的标准

- 测试什么时候才算做完？怎么知道已做了足够的测试？
- 对上述问题的一个答复是：
  - 你永远也不能完成测试，这个担子只会从你（软件工程师）身上转移到你的客户身上。
  - 客户/用户每次运行计算机程序时，程序就在经受测试。
- 另一个答复是：
  - 当你的时间或资金不够时，测试就完成了。
- 提倡使用统计建模和软件可靠性理论来预测测试的完成

17.2 策略问题

- 早在开始测试之前，就要以量化的方式规定产品需求。
  - 尽管测试的主要目的是查找错误，
  - 好的测试策略也能评估其他质量特性（可移植性、可维护性性）
- 明确地陈述测试目标
- 了解软件的用户并为每类用户建立用户描述
- 建立强调”快速周期测试“的测试计划
- 建立能够测试自身的“健壮”软件
  - 防错技术
  - 自动化测试和回归测试
- 利用有效的正式技术评审作为过滤器。
- 实施正式技术评审以评估测试策略和测试用例本身
- 为测试过程建立一种持续的改进方法

17.3 传统软件的测试策略

- 系统完成后测试，任何部分建成后测试，二者之间

17.3.1 单元测试：侧重于软件设计的最小单元的验证工作

- 单元测试问题
- 单元测试过程

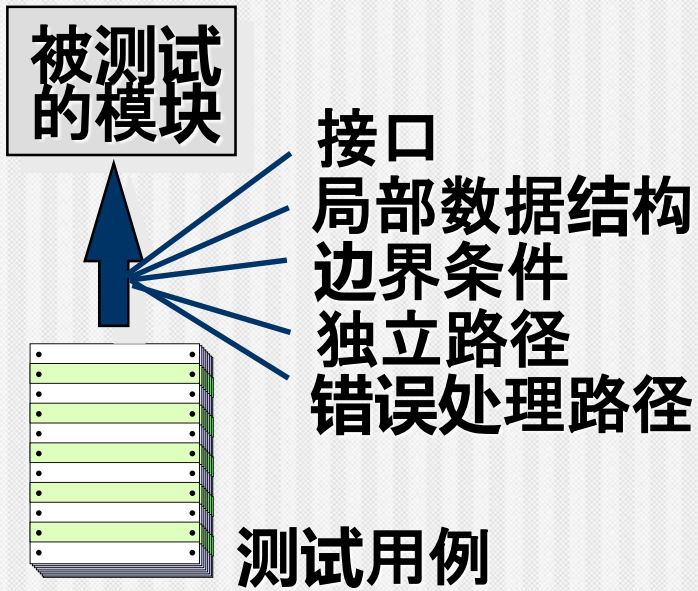


图17-3 单元测试

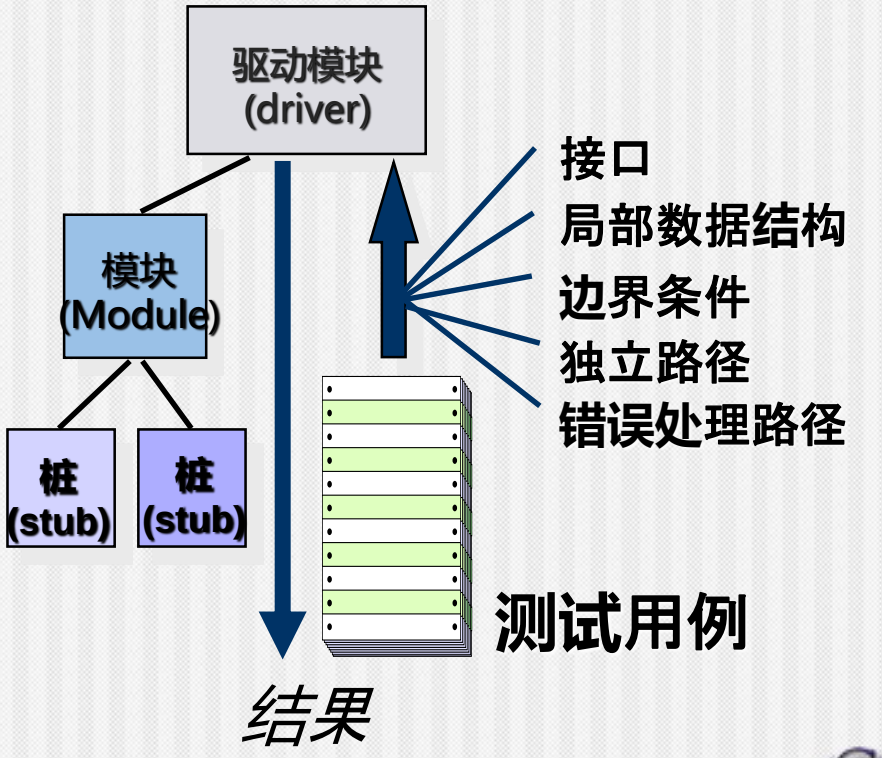


图17-4 单元测试环境

17.3.2 集成测试

- 构造软件体系结构的系统化技术，旨在发现接口相关错误
- 目标：利用已通过单元测试的构件建立设计中描述的程序结构

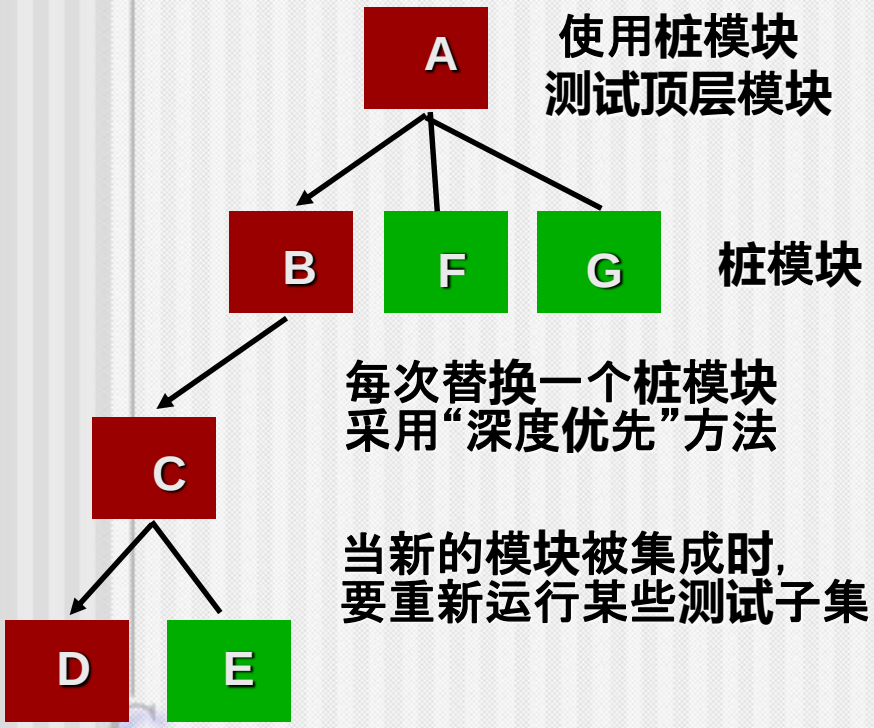


图17-5 自顶向下集成

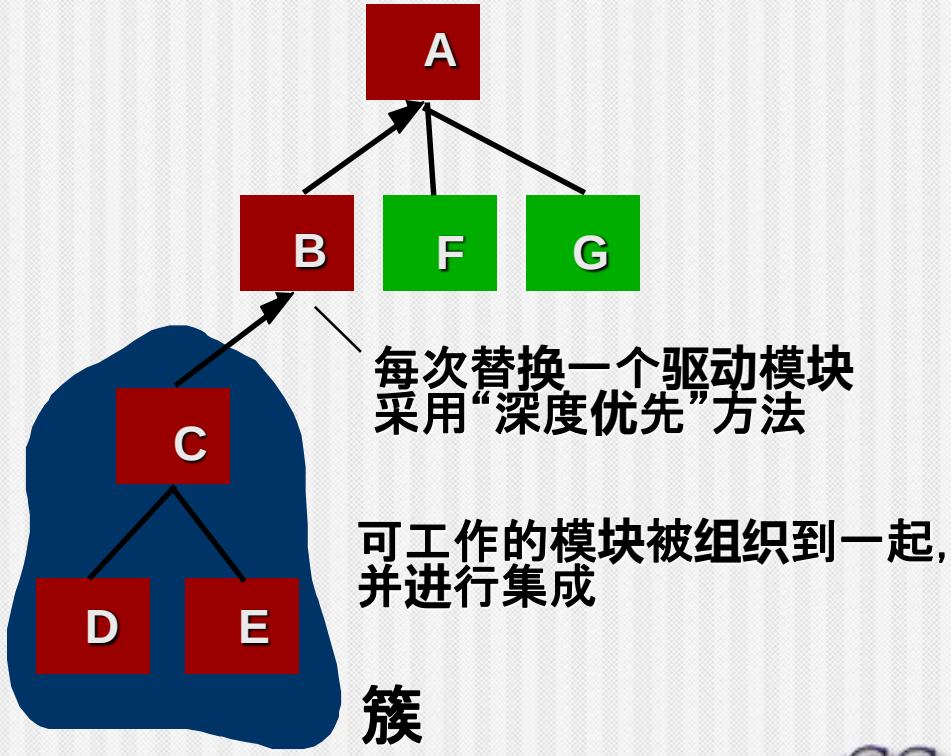


图17-6 自底向上集成

其他说明

- 回归测试：
  - 发生变更，重新执行已测试过的子集，
  - 确保无副作用
- 冒烟测试：常用的集成测试方法，时间关键性项目的步进机制
  - 1 编码完成的软件构件集成到系统
  - 2 设计一系列测试以暴露影响构建正确完成其功能的错误
  - 3 每日构建
- 策略的选择
- 集成测试工作产品

17.4 面向对象软件的测试策略

- 测试目标就是
  - 在现实的时间范围内
  - 利用可控的工作量尽可能多地找到错误。
- 对于面向对象软件，
  - 尽管这个基本目标是不变的，
  - 但面向对象软件的本质特征改变了
    - 测试策略和
    - 测试战术。

17.4.1 面向对象环境的单元测试

- 面向对象软件单元的概念发生了变化。
  - 封装导出了类的定义。
  - 每个类和类的实例(对象)包装有
  - 属性(数据)和
  - 处理这些数据的操作(函数)。
- 封装的类常是单元测试的重点，
  - 类中包含的操作是最小的可测试单元。
  - 类中可以包含一些不同的操作，
  - 特殊的操作可以作为不同类的一部分存在
  - 因此，必须改变单元测试的战术。

OO的单元测试

- 不再孤立地对单个操作进行测试(传统的单元测试观点)
  - 而是将其作为类的一部分。
- 考虑一个类层次结构，
  - 在此结构内对超类定义某操作X，
  - 并且一些子类继承了操作X。
  - 每个子类使用操作X，应用于它的私有属性和操作的环境。
  - 因操作X应用环境的细微差别，在子类环境中测试操作X是必要的
  - 这意味着在面向对象环境中，以独立的方式测试X往往是无效的。
- 面向对象软件的类测试等同于传统软件的单元测试
  - 传统的单元测试侧重于模块的算法细节和穿过模块接口的数据，
  - 类测试由封装在该类中的操作和类的状态行为驱动。

17.4.2 面向对象环境的集成测试

- 面向对象软件没有明显的层次控制结构，
  - 传统的自顶向下和自底向上集成策略已没有太大意义。
  - 类的成分间的直接/间接相互作用，不可能依次将操作集成到类中
- OO系统的集成测试有两种不同的策略
  - 一是基于线程的测试
  - 另一种方法是基于使用的测试
- 驱动模块和的使用也发生变化
  - 驱动模块可用于低层操作的测试和整组类的测试、代替用户界面
  - 桩模块用于需要类间协作，但协作类未完全实现的情况
- 簇测试——面向对象软件集成测试中的一步
  - 利用试图发现协作中的错误的测试用例
  - 来测试(CRC和对象-关系模型所确定的)协作的类簇

17.5 确认测试

- 始于集成测试结束，传统软件与OO软件的差别已经消失
  - 测试便集中于用户可见的动作和
  - 用户可识别的系统输出。
- 17.5.1确认测试准则
  - (1)功能或性能特征符合需求规格说明，因而被接受；
  - (2)发现了与规格说明的偏差，创建缺陷列表。
- 17.5.2 配置评审
  - 目的是确保所有的软件配置元素已正确开发、编目
  - 具有支持软件生命周期支持阶段的必要细节
- 17.5.3 α测试与β测试
  - α测试是由最终用户在开发者的场所进行
  - β测试在最终用户场所进行

17.6 系统测试

- 恢复测试
- 安全测试
- 压力测试
- 性能测试
- 部署测试

17.7 调试过程

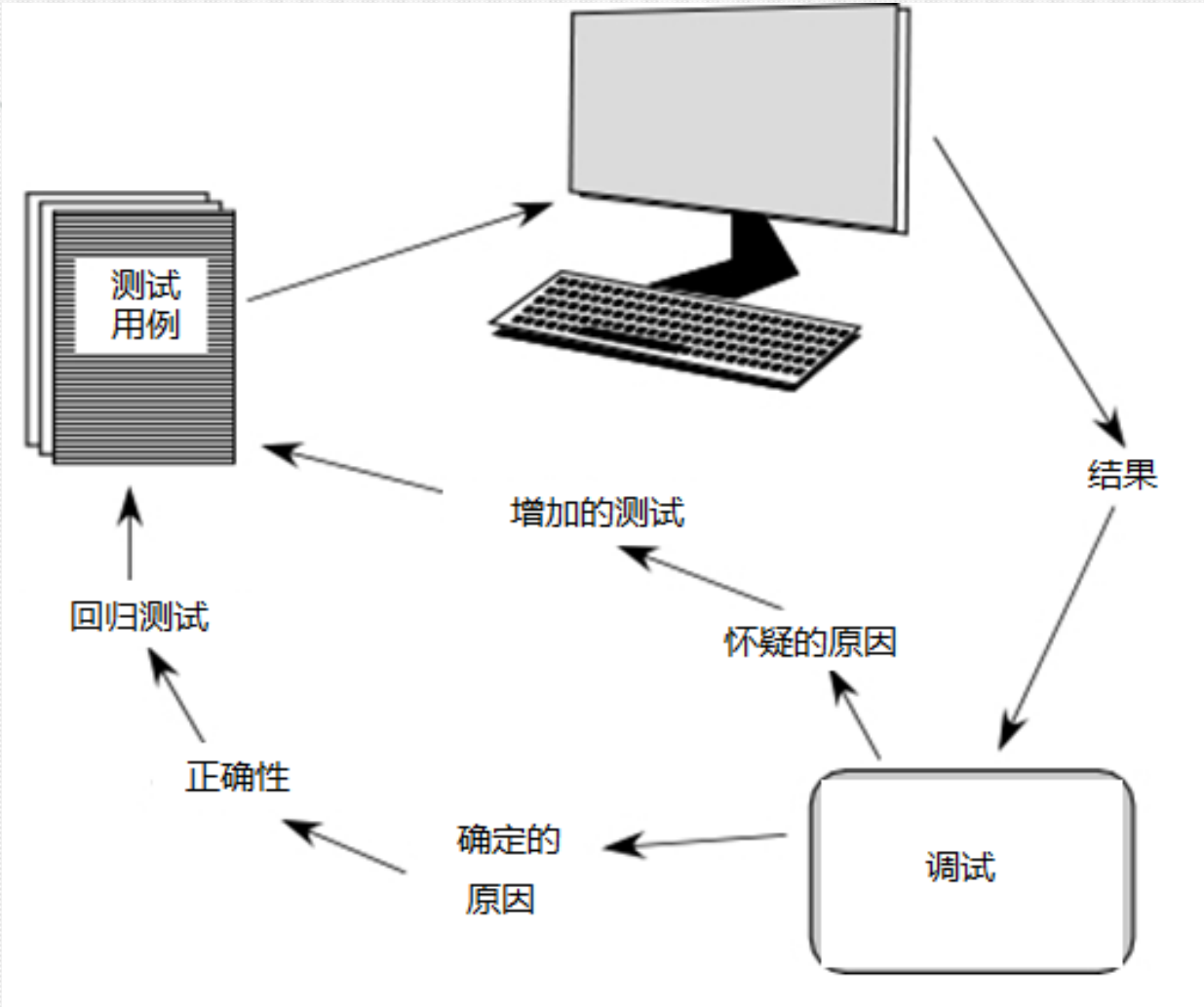


图17-7 调试过程

谢谢！