软件测试 Software Testing
06. 功能性测试与结构性测试

程适

cheng@snnu.edu.cn

计算机科学学院

2016 年 10 月 13 日

陕西師範大學
SHAANXI NORMAL UNIVERSITY

Outline

- 测试层次
- 功能性测试
- 结构性测试

测试层次

1. 单元测试（Unit Testing）
2. 集成测试（Integration Testing）
3. 系统测试（System Testing）
4. 验收测试（Acceptance Testing）

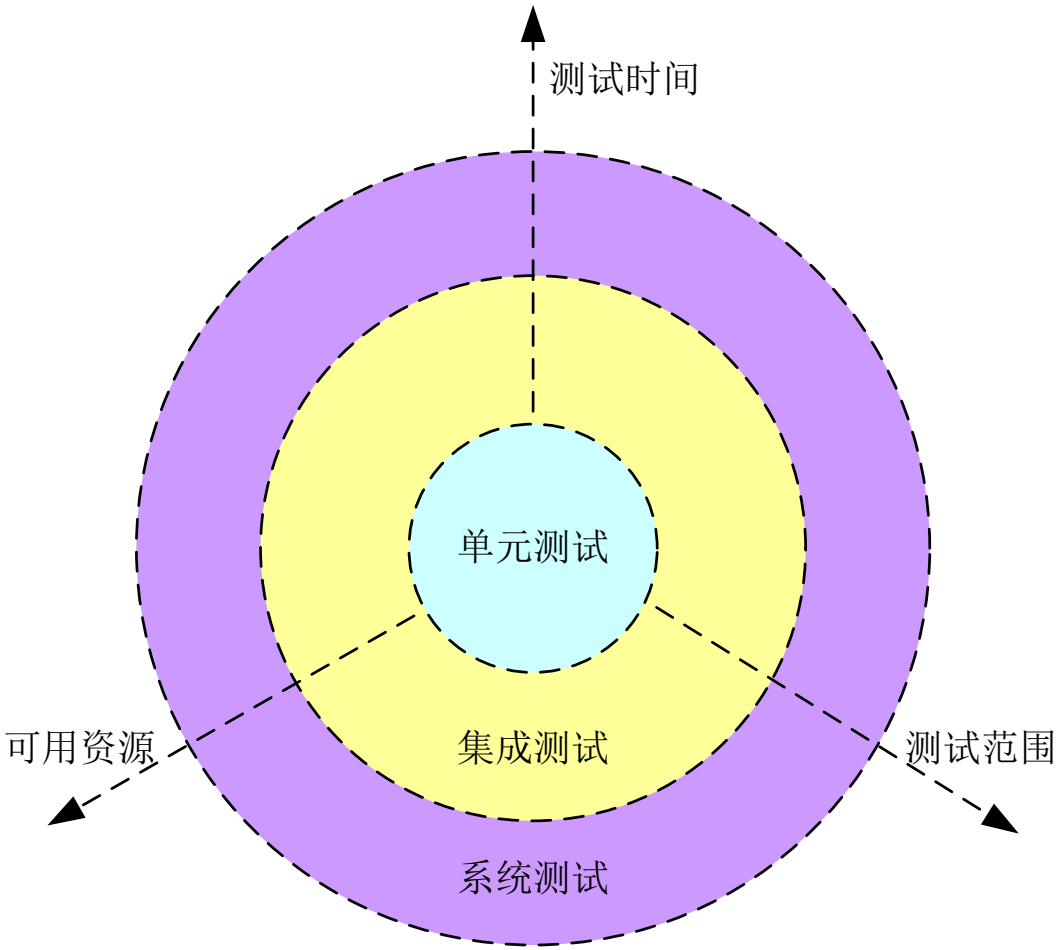
测试层次

- 单元测试
  - 测试的最早期阶段，焦点在于最小的被测软件的组成部分
  - 检验每个模块能否单独工作
- 集成测试
  - 在运行（可能是不完整）的应用中保证软件单元被结合后能正常操作的测试执行的阶段
  - 检验概要设计中模块接口设计问题

测试层次

- 系统测试
  - 当应用作为整体运行时的测试执行阶段（测试最终的应用）
- 验收测试
  - 以用户为主，由用户参加设计测试用例，对程序的功能、性能，以及可移植性、兼容性、可维护性、错误的恢复功能等进行确认。

测试层次



测试层次

名称	测试对象	侧重点	参照物	充分性的评价方法	时机	测试执行者
单元测试	软件的最小单元，如函数、方法等	逻辑的正确性	详细设计、源程序	代码、分支等覆盖率	软件中的基本组成单位完成后，边开发边测试	一般是开发人员
集成测试	软件的模块、子系统	接口的正确性	概要设计、详细设计	接口覆盖率	软件系统集成过程中，边集成，边测试	开发人员与测试人员
系统测试	系统	需求的满足性	产品需求	用户场景覆盖率	系统开发完成后，交付客户之前	测试人员
验收测试	系统	需求的满足性	客户需求	需求覆盖率	交付客户后，正式投入使用之前	客户

功能性测试

1. 等价类划分法
2. 边界值分析法
3. 决策表(判定表)方法
4. 因果图法
5. 正交实验法
6. 功能图法
7. 错误推测法

功能性测试

1. 黑盒测试也常被称为功能测试，虽然这不是一种准确的说法。在功能测试的时候，也可以采用白盒方法或灰盒方法，如查看源代码、变量在数据库中的值等，但多数情况下采用黑盒测试方法。
2. 功能测试主要是根据产品规格说明书，来检验被测试的系统是否满足各方面功能的使用要求。
3. 对于功能测试，针对不同的应用系统，其测试内容的差异很大，但都可以归为界面、数据、操作、逻辑、接口等几个方面。

功能性测试

- 程序安装、启动正常，有相应的提示框、错误提示等；
- 每项功能符合实际要求；
- 系统的界面清晰、美观；
- 菜单、按钮操作正常、灵活，能处理一些异常操作；
- 能接受正确的数据输入，对异常数据的输入可以进行提示、容错处理等；
- 数据的输出结果准确，格式清晰，可以保存和读取；

功能性测试

- 功能逻辑清楚，符合使用者习惯；
- 系统的各种状态按照业务流程而变化，并保持稳定；
- 支持各种应用的环境；
- 能配合多种硬件周边设备；
- 软件升级后，能继续支持旧版本的数据；
- 与外部应用系统的接口有效。

Web 页面功能测试举例

1. 页面链接测试
   1. 该页面是否存在，如页面不可显示信息，则视为页面链接无效；
   2. 该页面是否跳转到所规定的页面，主要是验证页面正确性。
2. Web 图形测试
3. 表单测试

功能性测试

- 如果变量引用的是物理量，可采用定义域测试和等价类测试
- 如果变量是独立的，可采用定义域测试和等价类测试
- 如果变量不是独立的，可采用决策表测试
- 如果可保证是单缺陷假设，可采用边界值分析和健壮性测试

功能性测试

- 如果可保证是多缺陷假设，可采用最坏情况测试，健壮最坏情况测试和决策表测试
- 如果程序包含大量例外处理，可采用健壮性测试和决策表测试
- 如果变量引用的是逻辑量，可采用等价类测试用例和决策表测试

结构性测试

1. 语句覆盖
2. 判定覆盖
3. 条件覆盖
4. 判定–条件覆盖
5. 条件组合覆盖
6. 路径覆盖
7. 基本路径测试法

静态测试与动态测试

- 静态测试:
  - 括代码检查、静态结构分析、代码质量度量等。它可以由人工进行，充分发挥人的逻辑思维优势，也可以借助软件工具自动进行。
  - 检查项：
    - 代码风格和规则审核
    - 程序设计和结构的审核
    - 业务逻辑的审核
- 动态测试

静态白盒测试

1. 静态白盒测试是在不执行的条件下有条理地仔细审查软件设计、体系结构和代码，从而找出软件缺陷的过程。
2. 优点：尽早发现软件缺陷。

代码检查和走查

1. 要求人们组成一个小组来阅读或直观检查特定的程序。
2. 代码走查中，一组开发人员（3~4人）对代码进行审核。参加者当中只有一人是程序编写者。
3. 优点：一旦发现错误，通常就能在代码中对其进行精确定位，这降低了调试成本。通常这个过程发现成批的错误。
4. 在典型的程序中，可以有效地查找出30%~70%的逻辑设计和编码错误。但不能有效地查找出更高层次的设计错误。

代码检查

- 选择要检查的代码模块的准则：
  1. 对于正确操作产品起关键作用的模块
  2. 复杂度较高的模块
  3. 与过去发生错误率较高的模块功能类似的模块
  4. 相对较新的或缺乏经验的软件程序师编写的模块。

代码检查

- 坚持编码标准和规范：
  1. 可靠性
  2. 可读性/维护性
  3. 移植性

代码检查的过程

- 代码检查小组的分工
  - 协调人：优秀的程序员但不是被检查代码的编码人员
  - 为代码检查分发材料、安排进程；记录发现的错误；确保所有错误随后得到改正
- 代码检查的主要活动
  - 由程序编码人员逐条语句讲述程序的逻辑结构。在讲述的过程中就有可能发现错误。
  - 对着常见的编码错误列表分析程序

代码检查的过程

- 代码检查会议的时间
  - 90~120分钟
  - 150行/小时
- 对代码检查必须树立正确的态度：程序员必须怀着非自我本位的态度对待检查；代码检查的目标是发现程序中的错误，从而改进软件质量。对代码检查的结果进行保密，仅限于参与者内部范围内，管理人员不能利用代码检查的结果。
- 代码检查的其他作用：
  - 程序员会得到编程风格、算法选择及编程技术方面的反馈信息
  - 代码检查时早期发现程序中最易出错部分的方法之一

用于代码检查的错误列表

1. 数据引用错误；
2. 数据声明错误；
3. 计算错误；
4. 比较错误；
5. 控制流程错误；
6. 子程序参数错误；
7. 输入/输出错误；
8. 其他检查。

数据引用错误

- 是否引用了未初始化的变量？
- 数组和字符串的下标是整数值吗？
- 是否在应该使用常量的地方使用了变量？
- 变量是否被赋予不同类型的值？
- 为引用的指针分配内存了吗？
- 一个数据结构是否在多个函数或者子程序中引用，在每一个引用中明确定义结构了吗？

数据声明错误

- 所有变量都赋予正确的长度和类型了吗？
- 变量是否在声明的同时进行了初始化？
- 存在声明过、但从未引用或者只引用过一次的变量吗？
- 在特定模块中所有变量都显示声明了吗？

计算错误

- 计算中是否使用了不同数据类型的变量？
- 计算中是否了解和考虑到编译器对类型或长度不一致的变量的转换规则？
- 在数值计算过程中是否可能出现溢出？
- 除数/模是否可能为零？
- 变量的值是否超过有意义的范围？
- 对于包含多个操作数的表达式，求值的次序是否混乱，运算优先级对吗？

比较错误

- 比较得正确吗？能否比较（例如字符串String之间比较）？是否混淆小于与小于等于？
- 存在分数或者浮点值之间的比较吗？如果有，精确问题会影响比较吗？
- 每一个逻辑表达式都正确表达了吗？逻辑计算如期进行了吗？求值次序有疑问吗？
- 逻辑表达式的操作数是逻辑值吗？

控制流程错误

- 如果程序包含begin...end和do ...while 等语句组，end是否对应？
- 程序、模块、子程序和循环能否终止？
- 可能存在永远不停的循环吗？
- 循环可能从不执行吗？

子程序参数(接口)错误

- 被调用模块接收到的形参(parameter)与调用模块发送的实参(argument)是否一致？
- 实参的属性（数据类型、大小）
- 实参的量纲
- 内置函数的调用是否正确
- 全局变量作为参数使用？
- 常数是否以实参形式传递？

输入/输出错误

- 软件是否严格遵守外部设备读写数据的专用格式？
- 文件或者外部不存在或者未准备好的错误情况有处理吗？
- 软件是否处理外部设备未连接、不可用，或者读写过程中存储空间占满等情况？

其他检查

- 标示符交叉引用列表，变量是否定义了却没被引用？
- 变量的默认值
- 编译通过，但出现了“警告”或“提示”信息
- 程序模块是否具有足够的鲁棒性？对输入的合法性进行检查
- 程序是否遗漏了某个功能？

代码走查

- 代码走查与代码检查很相似，差别在于规程和错误检查技术
  1. 不仅阅读程序和参照错误检查列表，代码走查的参与者可“使用计算机”。测试人员会带着一些书面的测试用例参加会议。

桌面检查

- 一个人阅读程序，对照错误列表检查程序，对程序推演测试数据。
- 程序员可以相互交换各自的程序，而不是桌面检查自己的程序。
- 桌面检查胜过没有检查，但效果远逊色于代码检查和走查。

同行评分

- 一种依据程序整体质量、可维护性、可扩展性、易用性和清晰性对匿名程序进行评价的技术
- 6~20人

小结

- 测试层次
- 功能性测试
- 结构性测试

小结

谢谢，欢迎提问!