Ch7 集成测试

2018年12月18日 11:53

- 多个单元测试出来以后，再进行集成测试

7.1集成测试介绍

- Integration testing （集成测试、组装测试、联合测试、子系统测试、部件测试）
- 集成测试与单元测试的区别：
  - 在集成测试之前，单元测试已经完成
  - 集成测试所集成的单元是经过单元测试保证的单元
  - 单元测试与集成测试所关注的范围不同，所发现问题的集合包含不相交区域，不能相互代替
- 集成测试与系统测试的区别：
  - 系统测试对象是整个系统以及与系统交互的硬件软件平台，对系统能够做各种功能性和非功能性的验证
  - 集成测试测试对象是模块与模块之间的接口，包括整体架构的问题
- 集成测试的**根本目的**就是根据系统的高层设计，或者叫架构设计，验证模块组装后是否满足功能、性能、可靠性是否满足需求
- 集成测试的主要优点：
  - 缺陷发现的较早
  - 更容易修复早期检测到的缺陷。
  - 我们得到关于单个模块和整个系统的健康状况和可接受性的早期反馈
  - 缺陷修复的调度是灵活的，并且它可以与开发重叠
- 集成测试关注的方面：
  - 多个模块或组件的接口
  - 集成的功能特征
  - 交互协议和消息
  - 系统架构
- 集成测试的步骤：
  - 步骤1:创建一个测试计划
  - 步骤2:设计测试用例并准备测试数据
  - 步骤3:如果适用，创建脚本来运行测试用例
  - 步骤4:一旦组件被集成，执行测试用例
  - 步骤5:修复错误(如果有的话)并重新测试代码
  - 步骤6:重复测试周期，直到组件成功集成
- 集成测试周期数和总集成时间由以下因素决定:
  - 系统中的模块数量
  - 模块的相对复杂度(圈复杂度)
  - 模块之间接口的相对复杂性
  - 在每个测试周期中需要聚集在一起的模块数量
  - 要集成的模块之前是否经过充分的测试
  - 每个测试-调试-修复周期的周转时间

- 集成测试的两种方式；
  - 增量
  - 非增量
- 什么时候完成集成测试
  - 这个系统是完全集成在一起的
  - 所有的测试用例都已经执行
  - 所有发现的严重和中度缺陷都已修复
- 集成测试的特征
  - 单元测试不完全，它将无力检查内容是否正确，相互调用模块之间的关系。
  - 与系统测试相比，集成测试用例从程序结构出发，其目的和目标更强，测试可以更有效地发现和定位问题。
  - 它可以更容易地模拟特殊困难的异常流，然后系统可以测试测试用例。
  - 快速定位问题。
- 集成测试与开发之间的关系
  - 集成测试与概要设计相对应，概要设计是系统结构集成测试的基础。
  - 集成是面向对象的开发关键活动。
  - 在结构设计开发中，集成测试同样重要。
  - 集成测试与架构设计之间的依赖关系。

7.2集成测试策略

- 增量集成测试:增量测试逐步扩展集成模块集。
- 非增量集成测试:通过非增量测试，软件模块被随机组合和测试。

  Big bang（*大爆炸*） integration 非增量

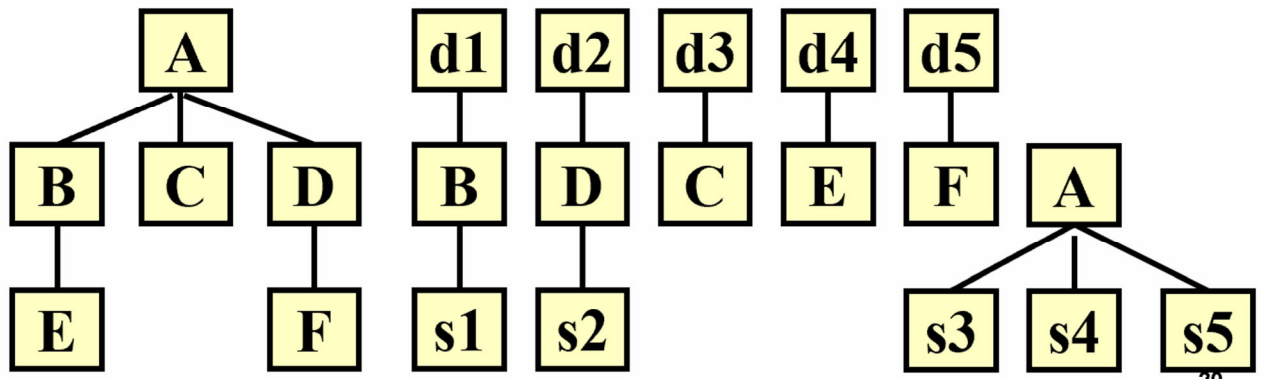
  Top-down （*自顶向下*）integration

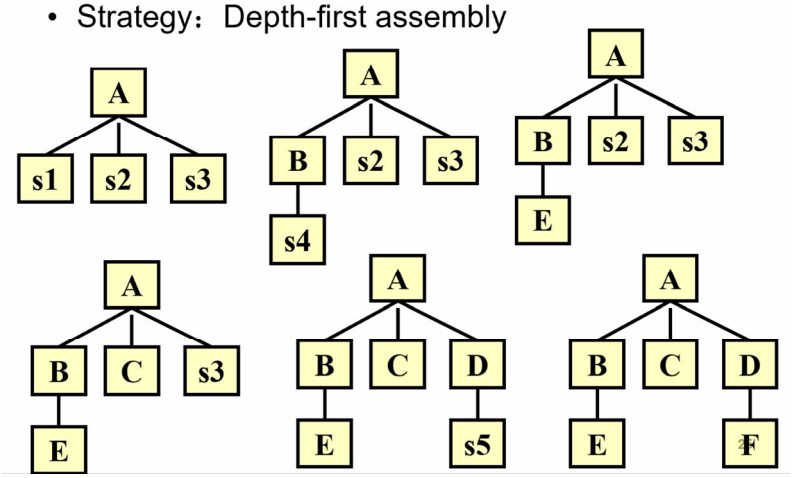
  Bottom-up （*自底向上*）integration
- Sandwich（*三明治*） integration

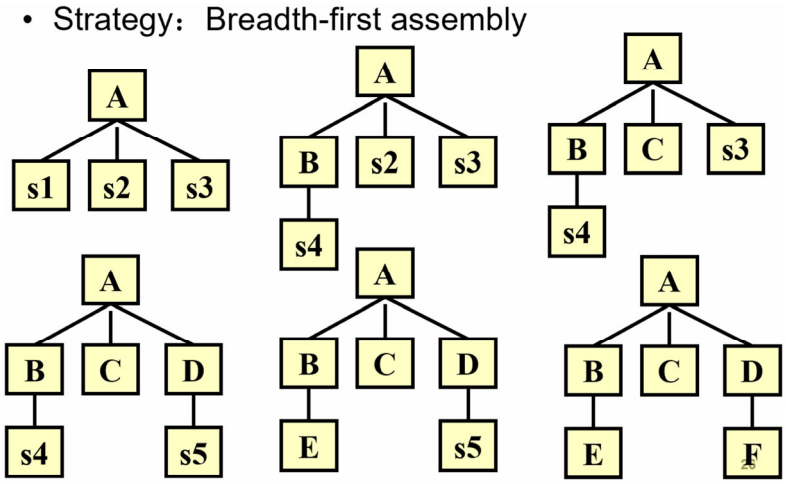
  Layers（*分层*） integration

  High-frequency （*高频*）integration

  Event-based（*基于事件*） integration
- 大爆炸集成：
  - 它是一种非增量集成方法，将所有系统组件一次性集成在一起，不考虑组件的依赖性和可能的风险。

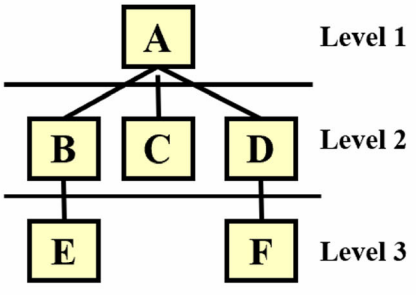


    - 优点：集成测试可以快速完成，只需要很少的存根和驱动程序；多个测试人员可以并行工作，人力和物力资源利用率更高
    - 缺点：一旦出现问题，问题的定位相对困难；许多接口错误直到系统测试后才会被发现。
- 自顶向下集成：
    - (1)首先集中在顶层组件,然后逐步测试组件的底部。
    - (2)可以使用深度优先和breadth-first策略。
    - (3)进行回归测试,排除可能的错误造成的整合。
    - (4)所有模块集成到系统完成测试,否则转(2)。
    - 

    - 

    - 当存根stub不能传递正确和有用的信息时，可以使用其他一些方法
        - 许多测试可以推迟到存根替换为真正的模块
        - 进一步开发的模块可以模拟存根的实际功能
        - 自底向上集成
    - 自顶向下的优点：
        - 早期展示主要的控制点和判断。
        - 采用深度优先装配，可以先实现和验证一个完整的软件功能。
        - 最多只需要一个驱动模块。
        - 支持故障隔离。
    - 自顶向下的缺点：
        - 开发和维护存根的成本都较高。
        - 底层组件的需求无法预测，可能导致对顶层组件的许多修改。
    - 使用范围：

        - 产品控制结构比较清晰稳定。
        - 高级产品界面变化相对较小。
        - 产品底部接口未定义或可能经常更改。
        - 产品控制模块技术风险较大，需要尽快进行验证。
- 2018.12.21
- 测试并不能保证软件质量，质量保证在设计阶段就应该关注
- 自底向上集成：
    - 从底层用最少的依赖组件开始，根据依赖结构，分层向上集成来检测整个系统的稳定性
        - 1、对于给定层次的模块，它的子模块（包括子模块的下属模块）已经组装并测试完成，所以不再需要桩。
        - 2、从底层模块开始，也可以把两个或多个叶子模块合并到一起进行测试。
        - 3、开发驱动模块对上面的模块进行测试，用实际模块代替驱动模块，与已测子模块一起组成大模块进行测试。
        - 4、重复这一过程直到顶层模块测试完成。
    - 自底向上的优点：
        - 允许对底层模块进行早期认证，任何准备进行集成测试的叶子节点都可以进行测试。
        - 减少存根开发的工作量
        - 支持故障隔离
    - 自底向上的缺点：
        - 驱动模块开发工作量比较大。
        - 高级验证被推迟到最后，无法及时发现设计错误。
        - 底部异常硬盖。
    - 使用范围：
        - 底层接口相对稳定、高层接口变化较多的产品。
        - 底部模块已经提前完成的产品
- 三明治集成（融合了自顶向下和自底向上的优点）
    - Test A first, develop stubs substitute B, C, D.
    - Test E, F, use drivers substitute B, D.
    - Test (B, E) (D, F), use driver substitute A.
    - Integrate together at last.



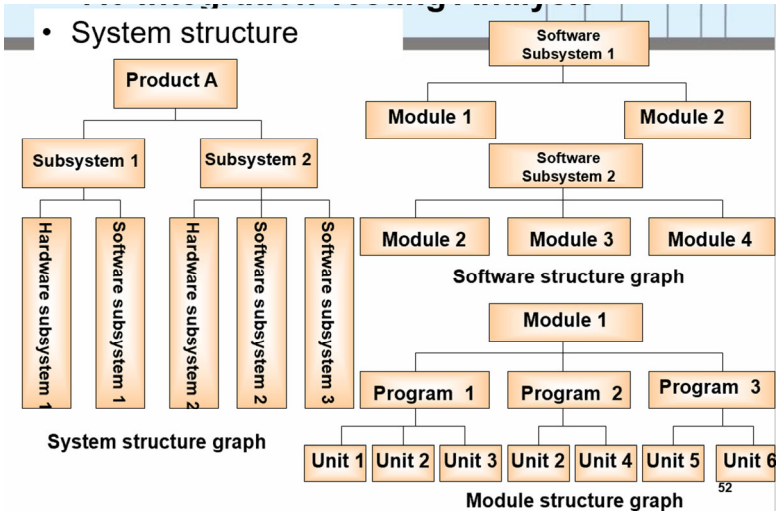
    - 优点：结合自顶向下策略和自底向上策略的优点。
    - 缺点：在集成测试之前，对中间层的测试是不够的。
    - 使用范围：被大多数软件开发项目所使用。
- 分层/层次集成Layer：
    - 通过增量集成方法验证了某一特定层次体系结构应用系统的稳定性和互操作性
    - 策略：
        - 系统的层分区。
        - 确定集成策略的内部级别。

        - 确定级别之间的集成策略
    - 使用范围：
        - 通信软件。
        - 一个清晰的层次产品系统。
- 高频率集成High-Frequency：（迭代时）
    - 为了检测集成错误和控制可能的基线偏差，经常向稳定基线添加新代码。
    - 高频集成所需的条件：
        - 获得一个稳定的增量，并且一个完整的子系统已经被证明没有问题。
        - 大多数有意义的递增函数可以在一个频繁的间隔内固定，例如每日构建daily build。
        - 测试包和代码并行开发，并始终维护最新版本。
        - 使用自动化。
        - 使用配置管理工具，否则增量版本将失去控制
    - 策略：
        - 开发人员同时提供完整的增量代码和测试人员对相关测试包的完整开发。
        - 测试人员一起修改或添加组件，形成一个新的集成体，并在其上运行集成测试套件。
        - 评价结果
    - 优点：
        - 有效地防止错误。
        - 严重的错误、遗漏和错误的假设可以在更早的时候被发现。
        - 误差定位相对容易。
        - 减少存根代码和驱动程序代码的开发。
        - 开发和集成可以同时进行。
    - 缺点：
        - 在最初的几个周期中可能很难顺利集成。
        - 高频集成频率需要很好的把握。
    - 使用范围：
        - 由迭代过程模型开发的产品
    - 创建每日构建在许多组织中非常流行（特别是在迭代的系统中）
        - 它有助于更快地交付系统
        - 它强调小的增量测试
        - 它稳定地增加了测试用例的数量
        - 系统使用自动化的、可重用的测试用例进行测试
        - 努力修复在24小时内发现的缺陷
        - 构建的前一个版本保留为引用和回滚
        - 一个典型的实践是保留过去的7-10构建
- 基于事件的集成Event-based——面向对象软件开发
    - 从消息的认证路径正确性出发，对系统进行增量集成，验证系统的稳定性。
    - 策略：
        - 从外部系统出发，分析了可能的系统输入集。
        - 选择一条消息，分析所传递的模块。
        - 集成这些模块并测试消息接口。

        - 重复上述步骤，直到测试所有消息。
    - 优点：
        - 验证一条消息可能需要几个模块，并且进展会更快。
        - 减少驱动模块的开发
    - 缺点：
        - 对于复杂系统，消息之间的相互关系可能是复杂的，难以分析。
        - 对于接口测试，它是不够的。
    - 使用范围：
        - 面向对象的系统。
        - 基于有限状态机的嵌入式系统。
- 为每个新创建一个测试策略构建(模块集)和解决以下问题,计划测试策略：
    - 需要从SIT测试计划中选择哪些测试用例?
    - 为了测试新构建中的补丁，现在应该重新执行以前失败的测试用例?
    - 如何确定部分回归测试的范围?
    - 测试此构建的估计时间、资源需求和成本是什么?

7.3集成测试分析

集成测试的关注内容：

- 1.体系结构分析
    - 从需求跟踪入手，划分系统实现的结构层次
        - 目的：综合水平考试是有帮助的。
    - 找出组件之间的依赖关系
        - 目的：识别集成测试的粒度，即基础模块的粒度。
    - 

      系统的体系结构是集成测试的依据，这是系统的体系结构图。
      产品的开发是一个分层的设计和逐步细化的过程，从最初的产品到最后的单元。
      一般单元测试针对最底层的叶子节点进行测试，系统测试对应的是产品级的。
      其余所有的层次都要通过集成测试来完成。

- 2.模块分析
    - 清除测试模块。
    - 首先确定各模块之间的关系，然后对最相关的模块进行集成。
    - 依次集成低耦合模块
- 3.接口分析

    - 接口部门：
        - 确定系统、子系统和模块的边界。
        - 确定模块内部接口、子系统内部接口和系统内部接口。
        - 确定模块或子系统之间的接口。
        - 确定系统与操作系统、软硬件接口与第三方的接口
    - 接口分类：
        - 函数接口
        - 消息接口
        - 类接口
    - 接口数据分析：
        - 跨接口的数据分析。
        - 功能接口侧重于参数号、顺序、属性等。
        - 消息接口主要关注消息类型、消息域等。
        - 类接口主要关注属性和行为。

- 4.可测试性分析
    - 主要将可测性衰落集中在积分递增上。所以把全部注意力放在无法测试的接口上，尽快找到解决方案。
- 5.集成测试策略分析
    - 良好的集成测试策略是主要关注的问题
        - 测试对象可以被充分测试，特别是关键模块。
        - 模块和接口清晰。
        - 投入资源得到充分利用。
- 6.集成测试常见故障
    - 配置/版本控制错误。
    - 功能的省略、重叠或冲突。
    - 文档或数据库使用不正确或不一致。
    - 错误的对象和信息绑定。错误的参数或不正确的参数值。
    - 组件之间的冲突
    - 资源竞争。

7.4集成测试用例设计

- 这节介绍集成测试用例设计思路，集成测试应用的是介于白盒和黑盒之间的灰盒测试技术。在测试用例设计中要综合使用白盒和黑盒测试方法。
- 一般经过集成测试分析以后，测试用例的大体轮廓已定，测试用例设计就是充分保证测试用例完成测试的既定目标。
- 采用的设计思想和方法：
    - 1.为系统的运行设计测试用例。
        - 在集成测试中，需要考虑的一个问题是可以使用接口，而且它不会阻碍集成测试执行的后续工作。因此，测试人员可以根据这一原理设计一些基本的功能测试用例来进行最小功能极限验证。
        - 方法和技术：
            - 等价划分法

- 规范导出（输出）法Standardize export method
  - 根据相应的规范描述来设计测试用例。每个测试用例被用来测试一个或多个规范陈述句。基本上是根据陈述规范所用语句的顺序来设计相应测试用例。
  - 这个方法在用例和规范陈述之间做到了很好的对应。加强了规范的可读性和可维护性。
  - 是一种正向的测试技术，需要逆向的测试技术对测试用例进行补充。
  - 当规范中的某种情形需要不同的处理的时候，每个陈述句可能需要多个用例。
  - 例子: 计算平方根的函数的规范
    输入：实数；输出：实数
    规范：当输入一个0或比0大的数的时候，返回其正平方根；当输入一个小于0的数时，显示错误信息“平方根非法-输入值小于0”并返回0；库函数Print-Line可以用来输出错误信息。
    - 由规范的3个陈述，可以得到两个用例
    - 用例1：输入4，输出应该为2。
      对应于第一个陈述(当输入一个0或比0大的数时…)
      用例2：输入-1，输出0，并显示错误提示信息。
      对应于第二，第三个陈述（当输入一个小于0的数时，显示错误信息“平方根非法-输入值小于0”并返回0；库函数Print-Line可以用来输出错误信息。）

- 2.为正向positive测试设计测试用例(正常功能)。
  - 接口的设计以及模块功能设计需求是明确的、无误的。
  - 测试的重点在于验证接口的需求和集成后而模块功能被正确无误的满足了。
  - 根据概要设计文档导出相关测试用例
  - 方法和技术：
    - 规范输出法
    - 输入测试
    - 输出覆盖测试
    - 等价划分
    - 状态转换测试
- 3.为逆向reverse测试设计测试用例（健壮性）。
  - 规格中可能出现的接口遗漏或接口定义错误，分析可能出现的接口异常情况，包括接口数据本身的错误，接口数据顺序错误。
  - 方法和技术：
    - 错误猜测法
    - 基于测试的风险
    - 基于故障检测
    - 边界值分析
    - 特殊值测试
    - 状态转换方法
- 4.为特殊需求设计测试用例。
  - 安全性能测试、性能测试、可靠性测试等应该在系统测试阶段进行，

但有一些产品在模块设计文档中就明确了这些指标，那就应该尽早展开接口这些特殊需求的测试。
        - 方法和技术：规范输出法
    - 5.设计满足覆盖率的测试用例。
        - 功能覆盖
        - 接口覆盖
        - 方法和技术
    - 6.测试用例的补充。
        - 新增功能、特性修改、缺陷修改
    - 注意：
        - 功能的增加、特性修改、缺陷修改
        - 测试重点要突出，关键的接口必须被覆盖到。
        - 充分考虑可回归性和执行自动化

7.5集成测试流程

- 计划阶段
- 设计阶段
- 实现阶段
- 性能阶段

7.6集成测试环境

- 对于简单系统来说（单进程软件），集成测试环境与单元测试环境比较类似。
- 集成测试环境也很复杂，需要依赖一些商用测试工具或测试仪，有时还需要开发一些接口模拟工具。
- 集成测试环境：
    - 硬件环境
    - 操作环境
    - 数据库环境
    - 网络环境

7.7集成测试原则

- 关键模块必须经过充分的测试
- 所有接口都必须测试
- 集成测试应该根据一定的级别进行
- 当接口更改时，所有相关接口都应该使用回归测试进行测试
- 集成测试按计划进行，防止随机测试
- 集成测试策略应该集成质量、成本和进度之间的关系。