中山大學
SUN YAT-SEN UNIVERSITY

Chapter 4
Software Dynamic Testing

Software Testing: Approaches & Technologies
School of Data & Computer Science, Sun Yat-sen University

Outline

- 4.1 白盒测试
- 4.2 黑盒测试
- 4.3 灰盒测试
- 4.4 测试用例设计
- 4.5 单元测试
- 4.6 集成测试
- 4.7 确认测试
- 4.8 系统测试
- 4.9 动态测试工具

Outline

- **4.6 集成测试**
  - 集成测试概述
  - 集成测试内容
  - 集成测试步骤
  - 集成测试方法
    - ✧ 一次组装式
    - ✧ 自顶向下递增式
    - ✧ 自底向上递增式
    - ✧ 混合渐增式
  - 集成测试阶段性过程
  - 集成测试的完成标志

4.6 集成测试

- **集成测试概述**
  - 集成测试也称组装测试或联合测试。
    - ✧ 集成测试是单元测试 (组件测试/模块测试) 的多级扩展，是在单元测试的基础上进行的一种有序测试。
      - ◌ 集成测试将测试目标模块按照设计要求，逐步 (有序地) 装配成高层的功能模块，并进行测试。
    - ✧ 集成测试的目的是检验软件单元之间的接口关系。
      - ◌ 集成测试是发现和改正模块接口错误的重要阶段。
      - ◌ 通过集成测试发现各软件单元接口之间存在的问题，最终把经过测试的单元组合成符合设计要求的软件系统 (子系统)。
    - ✧ 集成测试验证程序和概要设计说明的一致性。
      - ◌ 任何不符合概要设计说明的程序模块行为都应该形成测试报告内容。

4.6 集成测试

- **集成测试概述**
  - 集成测试解决的具体问题
    - ✧ 各个单元模块连接起来后，穿越模块接口的数据是否丢失。
    - ✧ 一个单元模块的功能是否会对其它单元模块的功能产生不利的影响。
    - ✧ 各个子功能组合起来，能否达到预期要求的上层功能。
    - ✧ 全局数据结构是否有问题。
    - ✧ 共享资源访问是否有问题。
    - ✧ 各个单元模块连接起来后，模块的误差积累是否会大到不能接受的程度。
    - ✧ 引入一个单元模块后，是否会对其他单元模块产生非功能性的不利影响。

4.6 集成测试

- **集成测试概述**
  - 集成测试具备以下不可替代的特点
    - ✧ 与单元测试相比，集成测试可以确定模块间接口信息内容的正确性，以及模块之间的相互调用关系是否符合设计要求。
      - ◌ 这些保障是单元测试无法提供的。
    - ✧ 与系统测试相比，集成测试用例从程序结构出发，其目的性、针对性更强，测试发现问题的效率更高，定位问题的效率也较高。
      - ◌ 集成测试具有可重复强、对测试人员透明的特点，发现问题后容易定位，能够加快测试进度，及早减少隐患。
    - ✧ 集成测试易于对系统测试用例难以模拟的特殊异常流程进行测试。
      - ◌ 理论上集成测试能够模拟所有运行的实际情况。

4.6 集成测试

- **集成测试概述**
  - 集成测试在软件分级测试中的重要意义
    - ✧ 集成测试在单元测试和系统测试之间起到承上启下的作用。
      - ◌ 集成测试能够发现大量在单元测试阶段不易发现的接口类错误，保证在进入系统测试前及早发现错误，减少损失。
      - ◌ 接口类错误是最常见的错误。
    - ✧ 集成测试通常是多人执行或第三方执行，而单元测试很多时候是单人执行。
      - ◌ 集成测试通过模块间的交互作用和测试人员之间的理解和交流，更容易发现目标软件在实现上、理解上的不一致和差错。

4.6 集成测试

- **集成测试概述**
  - 集成测试与单元测试、系统测试的比较

测试类型	测试对象	测试目的	主要依据	测试角度	主要方法
单元测试	模块内部的程序错误	消除局部模块的逻辑和功能上的缺陷	模块详细设计 模块外部说明	开发人员	白盒方法
集成测试	模块间的集成和调用关系	消除与设计相关的程序结构、调用、接口等方面的缺陷	软件概要设计 程序结构设计	测试人员	黑盒为主的灰盒方法
系统测试	总装后的系统及支持环境	对完整体系的有效性测试	软件概要设计 系统结构设计 需求说明书	目标用户	黑盒方法

4.6 集成测试

- **集成测试概述**
  - 集成测试的分层
    - ✧ 传统结构的软件系统按集成粒度不同，可以把集成测试分为三个层次：
      - ◌ 模块内集成测试；
      - ◌ 子系统内集成测试；
      - ◌ 子系统间集成测试。
    - ✧ 面向对象的应用系统按集成粒度不同，可以把集成测试分为两个层次：
      - ◌ 类内集成测试；
      - ◌ 类间集成测试。

4.6 集成测试

- **集成测试概述**
  - 集成测试部门
    - ✧ 集成测试可在开发部门进行，也可由独立的测试部门执行。
    - ✧ 开发部门尽量进行集成测试，测试部门有选择地进行集成测试
  - 集成测试小组
    - ✧ 集成测试应由专门的测试小组来执行。
    - ✧ 集成测试小组由有经验的系统设计人员和程序员组成。
    - ✧ 整个集成测试活动在评审人员出席的情况下进行。
  - 集成测试工作原则
    - ✧ 集成测试是产品研发中的重要过程，需要分配足够的资源。
    - ✧ 集成测试需要经过严密的计划，并严格按计划执行。
    - ✧ 采取递增式分步集成方式，逐步进行软件部件的集成和测试。
    - ✧ 引入测试自动化技术，提高集成测试效率。
    - ✧ 注重测试用例积累和管理，方便回归以及测试用例补充。

4.6 集成测试

- **集成测试内容**
  - 集成测试的主要内容
    - ✧ 集成测试的主要内容是功能性测试，侧重于测试软件模块组装以后的功能是否达到预期效果。
    - ✧ 集成测试的其他测试内容包括：
      - ◌ 资源冲突、任务优先级冲突、性能和稳定性在内的兼容性、可靠性、可用性、效率、可维护性等。

4.6 集成测试

- **集成测试内容**
  - 集成测试的具体内容
    - ✧ 集成后的功能性测试
      - ◌ 考察多个模块间的协作结果，既要满足集成后实现的复杂功能，也不能衍生出不需要的多余功能 (错误功能)。
      - ◌ 主要关注：
        - 被测集成对象的各项功能是否实现；
        - 异常情况是否有相关的错误处理；
        - 模块间的协作是否高效合理。

4.6 集成测试

- **集成测试内容**
  - 集成测试的具体内容 (续)
    - ✧ 接口测试
      - ◌ 模块间的接口包括函数接口和消息接口，接口测试相应包括对函数接口的测试和对消息接口的测试。
      - ◌ 对函数接口的测试：
        - 关注函数接口参数的类型和个数的一致性、输入/输出属性的一致性、范围的一致性。
      - ◌ 对消息接口的测试：
        - 关注收发双方对消息参数的定义是否一致、消息和消息队列长度是否满足设计要求、消息的完整性如何、消息的内存是否在发送过程中被非法释放、有无对消息队列阻塞进行处理等。

4.6 集成测试

- **集成测试内容**
  - 集成测试的具体内容 (续)
    - ✧ 全局数据结构测试
      - ◌ 全局数据结构的值在两次被访问的间隔是否可预知；
      - ◌ 全局数据结构的各个数据段的内存是否被错误释放；
      - ◌ 多个全局数据结构间是否存在缓存越界；
      - ◌ 多个软件单元对全局数据结构的访问是否采用锁保护机制。

4.6 集成测试

- **集成测试内容**
  - 集成测试的具体内容 (续)
    - ✧ 资源测试
      - ◌ 资源测试包括共享资源测试和资源极限测试。
      - ◌ 共享资源测试通常应用于数据库测试和支撑环境的测试，主要关注：
        - 是否存在死锁现象；
        - 是否存在资源过度利用情况；
        - 是否存在对共享资源的破坏性操作；
        - 公共资源访问锁保护机制是否完善。
      - ◌ 资源极限测试关注系统资源的极限使用情况以及资源耗尽时的处理，保证软件系统在资源耗尽的情况下不会出现系统崩溃。

4.6 集成测试

- **集成测试内容**
  - 集成测试的具体内容 (续)
    - ✧ 性能测试
      - ◌ 性能测试依据某个部件的性能指标进行，及时发现软件的性能瓶颈。
      - ◌ 在多任务环境中，还需测试任务优先级的合理性。
        - 实时性要求高的功能是否在高优先级任务中完成；
        - 任务优先级设计是否满足用户操作相应时间要求。
    - ✧ 稳定性测试
      - ◌ 是否存在内存泄漏而导致长期运行后资源耗竭；
      - ◌ 长期运行后是否出现性能的明显下降；
      - ◌ 长期运行是否出现任务挂起等。

4.6 集成测试

- **集成测试步骤**
  - 集成测试过程包括下列6个步骤：

    (1) 体系结构分析

    - 跟踪需求分析，对要实现的系统划分出结构层次图；
    - 对系统各个组件之间的依赖关系进行分析，然后据此确定集成测试粒度，即集成模块的大小。

    (2) 模块分析

    - 确定本次要测试的模块；
    - 找出与该模块相关的所有模块，并且按优先级对这些模块进行排列；
    - 从优先级别最高的相关模块开始，把被测模块与其集成到一起然后依次集成其他模块。

4.6 集成测试

- **集成测试步骤**
  - 集成测试过程包括下列6个步骤： (续)

(3) 接口分析

- 接口分析以概要设计为依据；
- 确定系统的边界、子系统的边界和模块的边界；
- 确定模块内部的接口；
- 确定子系统内模块间接口；
- 确定子系统间接口；
- 确定系统与操作系统的接口；
- 确定系统与硬件的接口；
- 确定系统与第三方软件的接口。

(4) 风险分析

(5) 可测试性分析

(6) 集成测试策略分析

4.6 集成测试

- **集成测试方法**
  - 模块组装成系统的方式
    - ✧ 选择什么方式把模块组装起来形成一个可运行的系统，直接影响到模块测试用例的形式、所用测试工具的类型、模块编号和测试的次序、生成测试用例和调试的费用等等。
    - ✧ 一次性组装方式
      - ◌ 一次性组装是一种非增殖式组装方式，也称整体拼装。
        - 先对每个模块分别进行模块测试；
        - 再把所有模块组装在一起进行测试，最终得到要求的软件系统。
      - ◌ 优点：测试过程中基本不需要设计开发测试工具。
      - ◌ 不足：对于复杂系统，当出现问题时故障定位困难；和系统测试接近，难以体现和发挥集成测试的优势。

4.6 集成测试

- **集成测试方法**
  - 模块组装成系统的方式 (续)
    - ✧ 递增式组装方式
      - ◌ 先对各个模块分别进行模块测试，然后将它们逐步组装成较大的系统。
        - 在组装的过程中边连接边测试，以期发现连接过程中产生的问题；
        - 通过递增逐步组装成为要求的目标系统。
      - ◌ 递增式软件集成的精细度取决于集成策略。
        - 通常先做模块间的集成，再做部件间的集成。
        - 优点：测试层次清晰，出现问题能够快速定位。
        - 缺点：需要开发测试驱动模块和桩模块。
      - ◌ 递增式组装方式包括自顶向下的组装方式和自底向上的组装方式。

4.6 集成测试

- **集成测试方法**
  - 自顶向下的递增式组装方式
    - ✧ 将模块按系统程序结构，沿着控制层次自顶向下进行组装。
      - ◌ 在测试过程中可以较早验证主要的控制和判断点。
      - ◌ 自顶向下的集成策略包括深度优先方式和广度优先方式。
    - ✧ 按深度优先方向组装方式：先把结构中一条主要控制路径上的全部模块组装起来。
      - ◌ 主要路径的选择与特定的软件应用特性有关，尽可能选取程序主要功能所涉及的路径。
      - ◌ 深度优先方式可以首先实现和验证一个完整的软件功能。
    - ✧ 按广度优先方向组装方式：从结构的顶层开始逐层向下组装。
      - ◌ 把上一层模块直接调用的模块组装进去，然后对每一个新组装进去的模块，再把其直接调用的模块组装进去。

4.6 集成测试

- **集成测试方法**
  - 自顶向下的递增式组装方式 (续)
    - ✧ 自顶向下组装测试的具体步骤

(1) 以主控模块作为测试驱动模块，把对主控模块进行单元测试时引人的所有桩模块用实际模块替代；

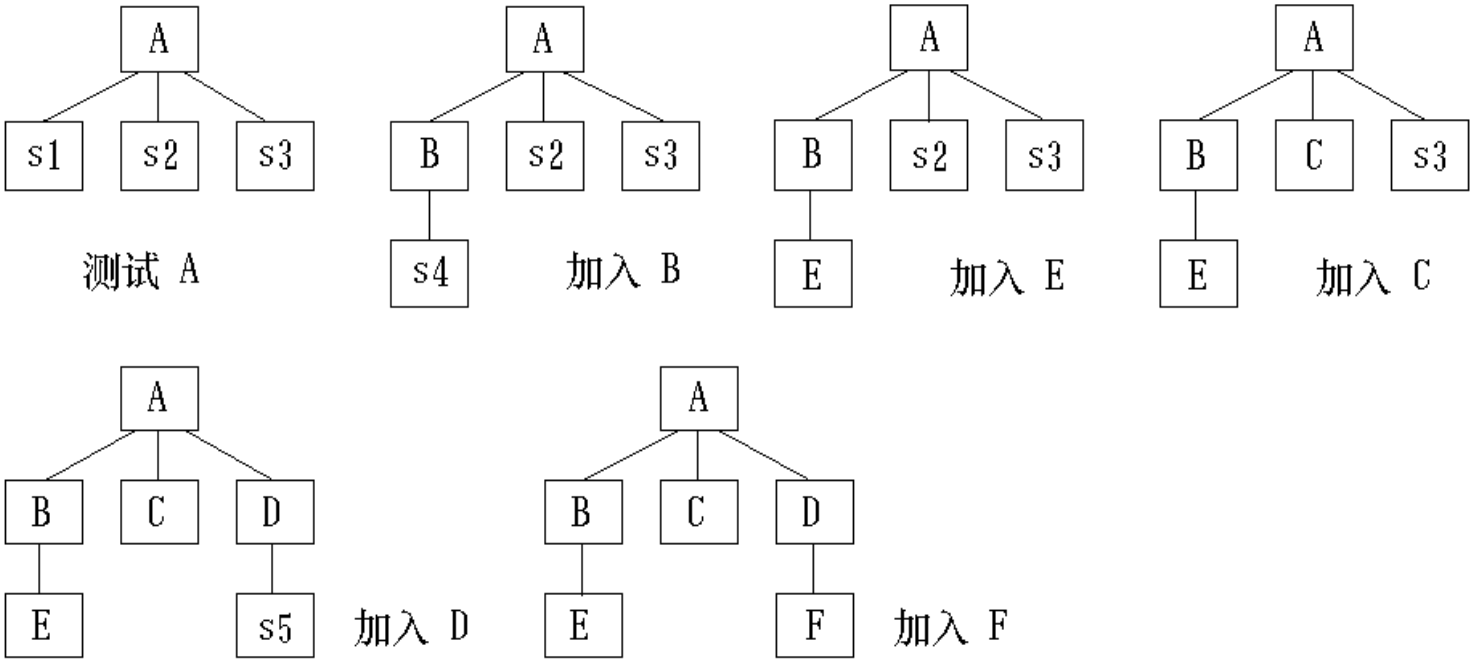
(2) 依据所选的集成策略 (深度优先或广度优先) 以及新模块的选择原则，每次用一个实际单元替换一个被调用的桩模块，并开发该单元可能需要的桩模块；

(3) 每集成一个模块的同时立即进行测试，排除组装过程中可能引进的错误。如果测试发现错误，则需要在修改后进行回归测试；

(4) 判断系统的组装测试是否完成，若还没有完成则转到第(2)步循环进行，直到集成结束。

4.6 集成测试

- **集成测试方法**
  - 自顶向下的递增式组装方式 (续)
    - ✧ 自顶向下组装测试的具体步骤 – 深度优先

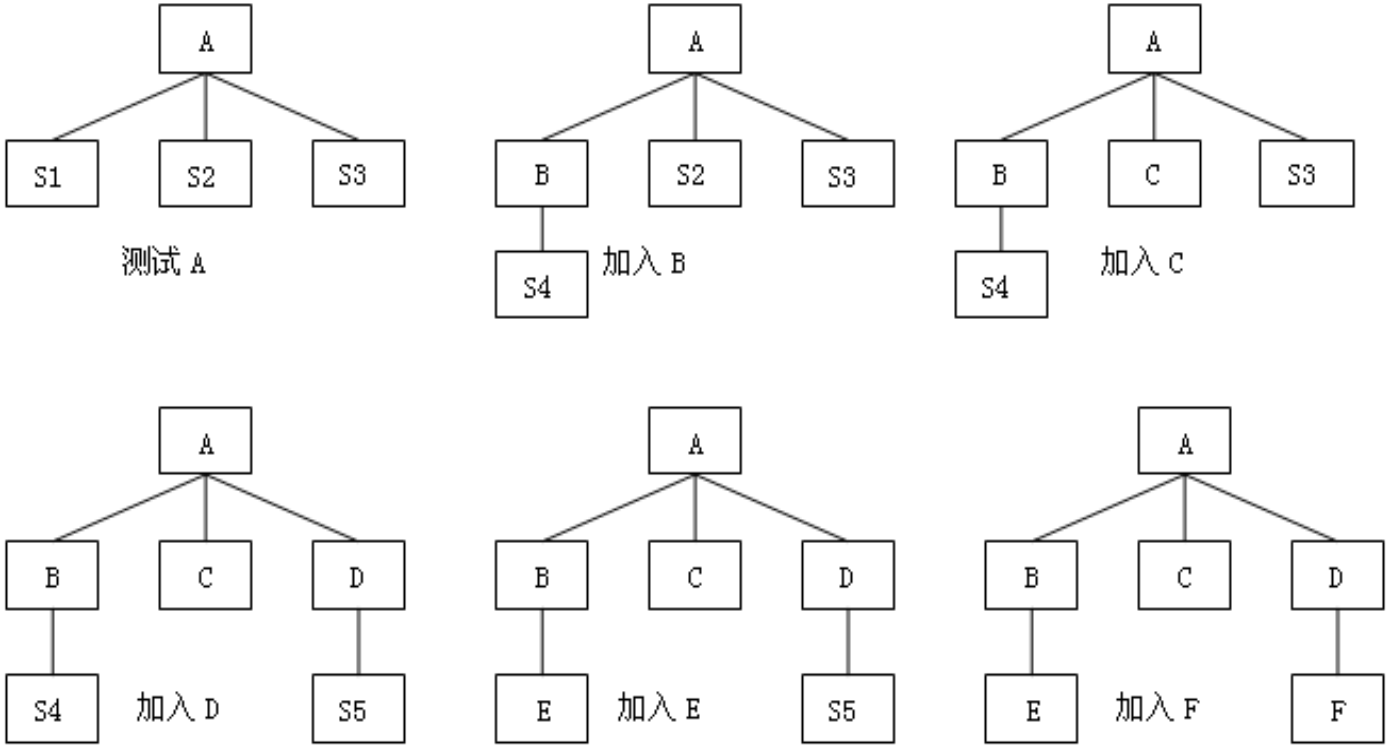


按深度方向组装的例子

S1-S5 是桩模块

4.6 集成测试

- **集成测试方法**
  - 自顶向下的递增式组装方式 (续)
    - ✧ 自顶向下组装测试的具体步骤 – 广度优先



自顶向下按广度方向组装的例子

4.6 集成测试

- **集成测试方法**
  - 自顶向下的递增式组装方式 (续)
    - ✧ 自顶向下组装测试的优点
      - ◌ 自顶向下组装测试在测试过程的早期对主要的控制点或判决点进行检验。
        - 在结构良好的软件系统中，判决需要在结构层次的较高层确定，因此需要及早发现主要控制点的问题。
      - ◌ 选用按深度优先方向组装的方式，可以首先实现和验证一个完整的软件功能。
        - 对逻辑输入的分支进行组装和测试，检查和克服潜藏的错误和缺陷，验证其功能的正确性，为此后主要分支的组装和测试提供保证。

4.6 集成测试

- **集成测试方法**
  - 自顶向下的递增式组装方式 (续)
    - ✧ 自顶向下组装测试的优点 (续)
      - ◌ 能够较早地验证功能可行性，给开发者和用户带来成功的信心。
      - ◌ 只有在个别情况下，才需要驱动程序 (最多不超过一个) 的支持，减少了测试驱动程序的开发维护费用。
      - ◌ 可以和开发设计工作并行执行集成测试，能够灵活适应目标环境。
      - ◌ 容易进行故障隔离和错误定位。

4.6 集成测试

- **集成测试方法**
  - 自顶向下的递增式组装方式 (续)
    - ✧ 自顶向下组装测试的缺点
      - ◌ 在测试时需要为每个模块的下层模块提供桩模块，带来较大的额外开发和维护费用；
      - ◌ 底层组件的需求变更可能会影响到全局组件，需要修改整个系统的多个上层模块；
      - ◌ 要求控制模块具有比较高的可测试性；
      - ◌ 在测试较高层模块时，低层处理采用桩模块替代，不能反映真实情况，重要数据不能及时回送到上层模块，可能导致测试不充分。
    - ✧ 解决上述问题的办法
      - ◌ 把某些测试推迟到用真实模块替代桩模块之后进行；
      - ◌ 开发能够模拟真实模块的桩模块。

4.6 集成测试

- **集成测试方法**
  - 自顶向下的递增式组装方式 (续)
    - ✧ 自顶向下组装测试的适用范围
      - ◌ 控制结构比较清晰和稳定的应用程序；
      - ◌ 系统高层的模块接口变化的可能性比较小；
      - ◌ 产品的低层模块接口还未定义或可能会经常因需求变更等原因被修改；
      - ◌ 产品中的控制模块技术风险较大，需要尽可能提前验证；
      - ◌ 需要尽早看到产品的系统功能行为；
      - ◌ 在极限编程 (Extreme Programming) 中使用测试优先的开发方法。

4.6 集成测试

- **集成测试方法**
  - 自底向上的递增式组装方式
    - 从程序模块结构的最底层模块开始组装和测试。
      - 因模块自底向上组装，对于一个给定层次的模块，它的子模块 (包括子模块的所有下属模块) 已经组装并测试完成，所以不再需要桩模块。
      - 模块测试过程中需要的子模块信息可通过直接运行子模块得到。

4.6 集成测试

- **集成测试方法**
  - 自底向上的递增式组装方式 (续)
    - ✧ 自底向上组装测试的优点
      - ◌ 驱动模块模拟了所有调用参数，测试模块返回结果不影响驱动模块，生成测试数据也没有困难；
      - ◌ 可以尽早验证底层模块的行为，如果关键模块是在结构图的底部，自底向上的测试具有优越性；
      - ◌ 自底向上的组装测试不必开发桩模块，提高了测试效率；
      - ◌ 对实际被测模块的可测试性要求较低，容易对错误进行定位。

4.6 集成测试

- **集成测试方法**
  - 自底向上的递增式组装方式 (续)
    - 自底向上组装测试的缺点
      - 被测软件系统的雏形只有当最后一个模块测试完毕时才能呈现；
      - 时序问题和资源竞争问题到测试过程的后期才能被发现；
      - 驱动模块的设计工作量大；
      - 不能及时发现高层模块设计上的错误。
    - 自底向上组装测试的适用范围是
      - 底层模块接口比较稳定的产品；
      - 高层模块接口变更比较频繁的产品；
      - 底层模块开发和单元测试工作完成较早的产品。

4.6 集成测试

- **集成测试方法**
  - 混合渐增式集成测试方法 (三明治集成方法)
    - ✧ 衍变的自顶向下的渐增式测试
      - ◌ 强化对输入/输出模块和引人新算法模块进行测试；
      - ◌ 再自底向上组装成为功能相对完整且相对独立的子系统；
      - ◌ 然后由主模块开始自顶向下进行渐增式测试。
    - ✧ 自底向上-自顶向下的渐增式测试
      - ◌ 首先对含有读操作的子系统自底向上直至根结点模块进行组装和测试；
      - ◌ 然后对含有写操作的子系统做自顶向下的组装与测试。
    - ✧ 回归测试
      - ◌ 采取自顶向下的方式测试被修改的模块及其子模块；
      - ◌ 然后将这一部分视为子系统，再自底向上测试，以检查该子系统与其上级模块的接口是否匹配。

4.6 集成测试

- **集成测试方法**
  - 混合渐增式集成测试方法
    - ✧ 混合渐增式集成测试的策略
      - ◌ 在组装测试时，应当确定关键模块并及早进行测试；
      - ◌ 对该模块及所在层的下面各层使用自底向上的集成策略；
      - ◌ 再对该模块所在层的上面各层使用自顶向下的集成策略；
      - ◌ 最后对系统进行整体测试。
    - ✧ 混合渐增式集成测试方法的优缺点
      - ◌ 集成了自顶向下和自底向上两种集成策略的优点。
        - 可以降低桩模块和驱动模块的开发成本
      - ◌ 缺点：在被集成之前，中间层不能尽早得到充分的测试。
    - ✧ 混合渐增式集成测试方法的适用范围
      - ◌ 多数软件开发项目可以应用。

4.6 集成测试

- **集成测试阶段性过程**
  - 集成测试的4个阶段
    - (1) 集成测试计划阶段
    - (2) 集成测试设计与开发阶段
    - (3) 集成测试执行阶段
    - (4) 集成测试评估阶段
  - 集成测试方案
    - ✧ 在开始进行体系结构设计的时候介入并制定测试方案；
    - ✧ 在进入详细设计之前完成集成测试方案；
    - ✧ 在进入系统测试之前结束集成测试。

4.6 集成测试

- **集成测试阶段性过程**

(1) 集成测试计划阶段

- ✧ 概要设计评审通过后，参考需求规格说明书、概要设计文档、产品开发计划时间表尽快制定集成测试计划。
- ✧ 集成测试计划所包含的内容：
  - ◌ 集成测试策略、方法、内容、范围、通过准则的确定；
  - ◌ 工具与复用分析；
  - ◌ 基于项目人力、设备、技术、市场要求等各方面的决策；
  - ◌ 集成测试进度计划；
  - ◌ 工作量估算、资源需求、进度安排、风险分析和应对措施
  - ◌ 集成测试方案编制；
  - ◌ 接口分析、测试项、测试特性分析。

4.6 集成测试

- **集成测试阶段性过程**

(1) 集成测试计划阶段 (续)

- ✧ 集成测试计划的编制要与单元测试的完成时间协调。
  - ◌ 采用何种系统组装方法来进行集成测试；
  - ◌ 组装测试过程中要考虑集成的层次、软件的层次、模块的复杂度和重要性以及连接各个模块的顺序；
  - ◌ 模块代码编制和测试进度是否与组装测试的顺序一致；
  - ◌ 测试过程中是否需要专门的硬件设备。
- ✧ 解决了上述问题之后，列出各个模块的编制、测试计划表，标明每个模块单元测试完成的日期、首次集成测试的日期、集成测试全部完成的日期、以及需要的测试用例和所期望的测试结果。

4.6 集成测试

- **集成测试阶段性过程**

(2) 集成测试设计与开发阶段

- ✧ 在软件生命周期的详细设计阶段同步进行集成测试设计；
- ✧ 以需求规格说明书、概要设计、集成测试计划文档作为集成测试设计的参考依据；
- ✧ 完成测试规程/测试用例的设计与开发，确定测试步骤，设计测试数据，完成测试工具、测试驱动和桩的开发。

(3) 集成测试执行阶段

- ✧ 所有的集成测试工作准备完毕，测试人员在单元测试完成以后就可以执行集成测试；
- ✧ 执行阶段搭建好测试环境，开展测试工作，确定测试结果，处理测试过程中的异常。

4.6 集成测试

- **集成测试阶段性过程**

(4) 集成测试评估阶段

- ✧ 集成测试执行结束后，召集相关人员对测试工作进行度量，对测试结果进行评估，确定是否通过集成测试。
- ✧ 执行阶段的度量：
  - ◌ 集成测试对象的数量、运行的用例数量、通过/失败的用例数量、发现的缺陷数量、遗留的缺陷数量、集成测试执行的工作量。
- ✧ 评估阶段要完成的工作：
  - ◌ 按照集成测试报告模板出具集成测试报告，如有必要对集成测试报告进行评审，将所有测试相关工作产品纳入配置管理。

4.6 集成测试

- **集成测试的完成标志**
  - 集成测试的完成标志
    - ✧ 成功地执行了测试计划中规定的所有集成测试；
    - ✧ 修正了所有发现的错误；
    - ✧ 测试结果通过了专门小组的评审；
    - ✧ 形成测试报告。
      - ◌ 测试报告记录实际的测试结果、在测试中发现的问题、解决这些问题的方法以及解决之后再次测试的结果。
      - ◌ 测试报告还要记录目前不能解决、还需要管理人员和开发人员注意的一些问题，提供测试评审和最终决策，提出处理意见。

Outline

- 4.1 白盒测试
- 4.2 黑盒测试
- 4.3 灰盒测试
- 4.4 测试用例设计
- 4.5 单元测试
- 4.6 集成测试
- 4.7 确认测试
- 4.8 系统测试
- 4.9 动态测试工具

Outline

- **4.7 确认测试**
  - 确认测试概述
  - 确认测试过程
  - 确认测试结果
  - α 测试和 β 测试

4.7 确认测试

- **确认测试概述**
  - 确认测试也称合格性测试
    - ✧ 确认测试是严格遵循有关标准的一种符合性测试，以确定软件产品是否满足所规定的要求 (即是否合格)。
      - ◌ 规定要求指的是软件规格说明书中确定的软件功能和技术指标，或是专门为测试所规定的确认准则。
    - ✧ 在软件测试周期中，确认测试介于集成测试和系统测试之间，是必不可少的一项测试。
  - 确认测试的方法和内容
    - ✧ 确认测试是在模拟的环境 (如开发环境) 下，用黑盒测试的方法，验证被测软件是否满足需求规格说明书列出的要求。
    - ✧ 确认测试的内容主要包括安装测试、功能测试，性能测试、可靠性测试、安全性测试、效率测试、易用性测试、可移植性测试、可维护性测试、文档测试等。

4.7 确认测试

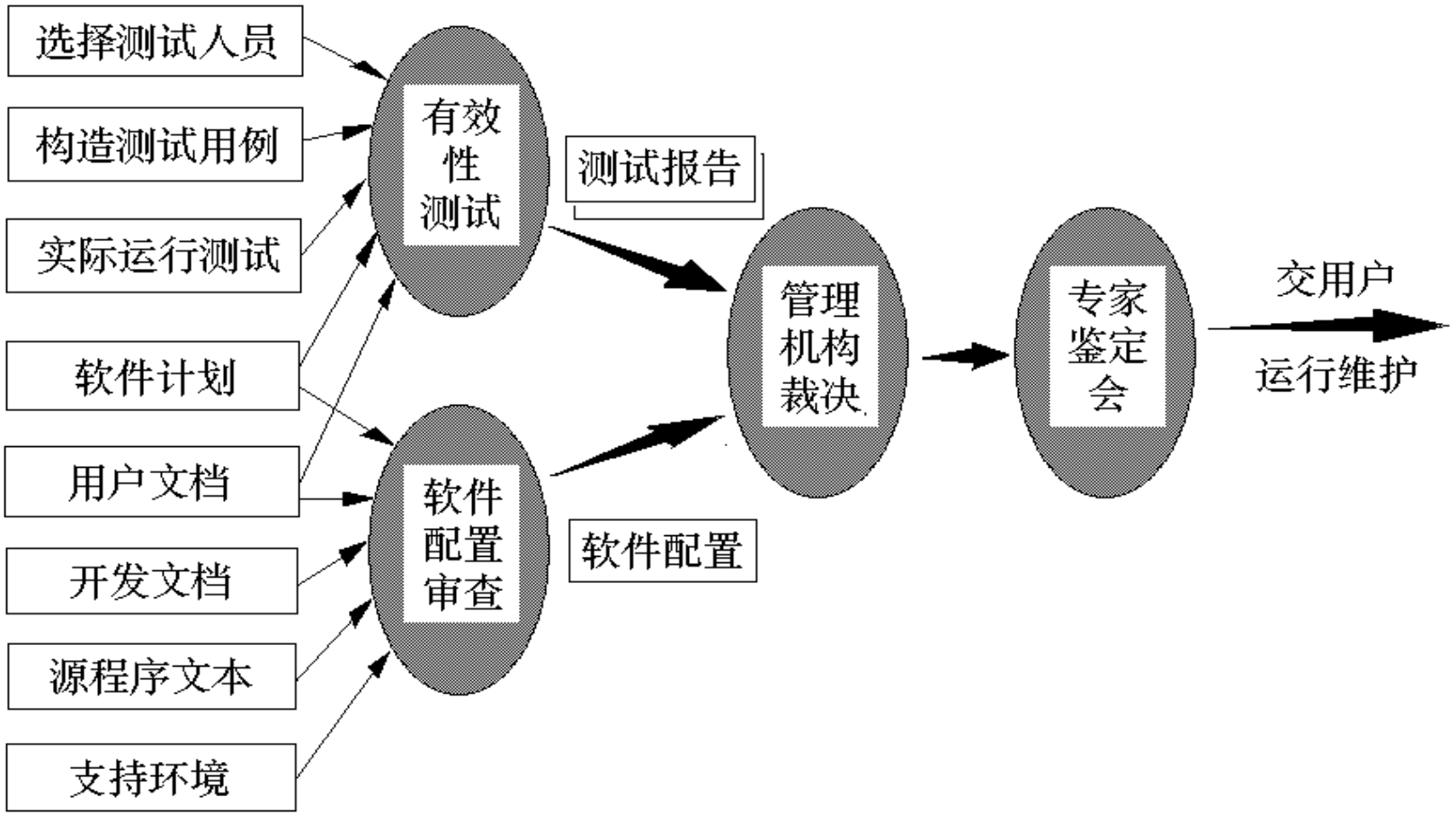
- **确认测试概述**
  - 确认测试是直接针对用户需求的测试
    - ✧ 确认测试应在尽可能真实的环境中进行。
    - ✧ 确认测试必须有用户参与，或以用户为主进行。
    - ✧ 在确认测试种类中，*alpha* 测试在开发场所进行，有用户参与；而 *beta* 测试在客户场所进行。
  - 确认测试规范
    - ✧ 依照相关的软件测试评价标准制定相应的测试规范，利用确认测试可能的测试方法，借助可能的测试工具。
    - ✧ 确认测试的具体组织与实施在已建立的软件测试管理体系下展开。
    - ✧ 确认测试的相关结论以规范的测试报告形式给出。

4.7 确认测试

- **确认测试过程**
  - 制定计划
    - ✧ 首先制定确认测试计划，规定要测试的种类；还需要制定一组测试步骤，描述具体的测试用例。
  - 通过实施预定的测试计划和测试步骤，确定
    - ✧ 软件的特性是否与需求相符
    - ✧ 所有的文档是否正确且便于使用
    - ✧ 对其它软件需求，例如可移植性、兼容性、出错自动恢复、可维护性等，也都要进行测试

4.7 确认测试

- **确认测试过程**



4.7 确认测试

- **确认测试结果**
  - 确认测试的结果
    (1) 功能和性能与需求文档及用户的要求一致，软件可以接受，即通过测试。
    (2) 功能和性能与需求文档及用户的要求有一定的差距
    - 详细列出软件各项缺陷或问题的清单或列表
    - 提交对应的问题报告
    - 必要时，要与用户协商来解决所发现的缺陷或错误
  - 确认测试应交付的文档有
    - ✧ 确认测试分析报告
    - ✧ 最终用户手册和操作手册
    - ✧ 项目开发总结报告

4.7 确认测试

- α 测试和 β 测试
  - 进行 α 测试和 β 测试的原因
    - ✧ 如果软件是专为某个用户开发的，那么可以进行一系列验收测试，以便用户确认所有需求都得到满足。
      - ◌ 验收测试是由最终用户而不是系统的开发者进行的。
    - ✧ 如果一个软件是为多个用户的使用开发的产品 (市场化产品)，让每个用户逐个进行正式的验收测试是不切实际的。
      - ◌ 在这种情况下，软件开发商使用被称为 α 和 β 测试的过程，来发现那些看起来只有最终用户才能发现的错误。

4.7 确认测试

- α 测试和 β 测试
  - α 测试
    - ✧ α 测试是一种用户在软件开发环境下进行的测试，或是开发机构内部人员在模拟实际操作环境下进行的测试。
    - ✧ 进行 α 测试时，软件被设置在一个自然状态下使用。
      - ◌ 开发者在用户旁边，对用户进行“指导”，并负责记录发现的错误和使用中遇到的问题。
      - ◌ α 测试是在受控制的环境下进行的测试。
    - ✧ α 测试的目的是评价软件产品的功能、可使用化、可靠性、性能和支持，尤其注重产品的界面和特色。
    - ✧ α 测试人员是除开发人员外，首先使用产品的人，他们提出的功能和修改意见通常很有价值。

4.7 确认测试

- α 测试和 β 测试
  - β 测试
    - ✧ β 测试是由软件最终用户在客户使用场所进行的测试。
    - ✧ β 测试是软件在开发者不能控制的环境中的“真实”应用。
      - ◌ 开发者不在 β 测试的现场，用户记录在 β 测试过程中遇到的一切问题，并定期把这些问题报告给开发者。
      - ◌ 开发者收到 β 测试报告后，对软件产品进行必要的修改，并准备向全体客户发布最终的软件产品。
    - ✧ β 测试主要衡量产品的功能、易用性、可靠性、性能和支持性
      - ◌ 着重于产品的支持性，包括文档、客户培训和支持产品生产能力。
      - ◌ 只有当 α 测试达到一定的可靠程度时，才能开始 β 测试。
      - ◌ β 测试处于整个测试的最后阶段，通常不能发现主要缺陷
      - ◌ 产品的所有手册文本应该在此阶段完全定稿。

Thank you!