软件测试

广州爱孕记信息科技有限公司

2019.5

目录

软件测试概述

01

1.1软件测试的定义和概念

- **软件测试的IEEE定义：**
  - 使用人工或自动的手段来运行或测量软件系统的过程，目的是检验软件系统是否满足规定的需求，并找出与预期结果之间的差异。
- **软件测试的概念：**
  - 软件测试就是为了发现错误而执行程序的过程(狭义观点)。
  - 使用人工或自动的手段，来运行或测试软件系统的过程，目的是检验软件系统是否满足规定的需求，并找出与预期结果之间的差异。(标准定义IEEE)
  - 软件测试就是为了证明程序有错，而不是证明程序无错误(辨证观点) 。
  - 测试被定义为“对软件系统中潜在的各种风险进行评估的活动”。(风险观点)软件测试就是“验证（Verification）”和“有效性确认（Validation）”活动构成的整体，即软件测试V&V 。(标准观点)
  - 要完整理解软件测试，就要从不同方面去审视软件测试，概括起来，软件测试就是贯穿整个软件开发生命周期，对软件产品（包括阶段性产品）进行验证和确认的活动过程，其目的是尽快尽早地发现在软件的缺陷。

1.2软件测试的对象

- **软件**是计算机系统中与硬件相互依存的一部分，包括程序、数据、与其相关文档的完整结合。软件 = 程序 + 数据 + 文档。
- **测试对象：**
  - 源程序/目标代码
  - 各开发阶段的文档（需求规格说明、概要设计说明、详细设计说明及其它相关文档）

1.3软件测试的目的

- **从用户角度看的目的：**
  - 通过软件测试发现隐藏的错误和缺陷，考虑是否可以接受该产品。
- **从开发者角度看的目的：**
  - 表明软件产品不存在错误，验证软件实现了所有用户的要求。
- **从测试人员角度看的目的：**
  - 发现错误，预测错误，提供软件可靠性错误，对软件做出评价。

1.3软件测试的目的

- 帮助开发人员、测试工程师发现问题、分析问题。
- 减少软件的缺陷数目或者降低软件缺陷的密度。
- 提高软件的可靠性
- 评估软件的性能指标。
- 增加用户对软件的信心。
- 测试的最终目的是尽快尽早地发现在软件中的缺陷，通过修正各种错误和缺陷提高软件质量，回避软件发布后由于潜在的软件缺陷和错误造成的隐患所带来的商业风险。

1.4软件测试的原则

- 所有测试都应追溯到用户需求。
- 应当把“尽早地和不断地进行软件测试”作为软件测试者的座右铭。
- 由于软件的复杂性和抽象性，在软件生命周期各个阶段都可能产生错误，所以不应把软件测试仅仅看作是软件开发的一个独立阶段的工作，而应当把它贯穿到软件开发的各个阶段中。在软件开发的需求和设计阶段就应开始测试工作，编写相应的测试文档。
- 完全测试是不可能的，测试需要终止，想要进行完全的测试，在有限的时间和资源条件下，找出所有的软件缺陷和错误 使软件趋于完美是不可能的主要有三个原因： ① 输入量太大; ② 输出结果太多；③ 路径组合太多。要终止。

1.4软件测试的原则

- 测试无法显示软件潜在的缺陷：进行测试是可以查找并报告发现的软件缺陷和错误，但不能保证软件缺陷和错误全部找到。
- 充分注意集试中群集现象（二八定理）：经验表明，在所测试程序段中，若发现的错误数目多，则残存的错误数目也较多。缺陷的二八定理指的是，一般情况下，80%软件缺陷出现在20%的功能区域，在测试过程中，投入主要的人力和精力重点测试这20%的功能区域。
- 开发人员应避免检查自己的程序：基于心理因素，揭露自己程序中的问题总不是一件愉快的事，不愿否认自己的工作；由于思维定势，人们难于发现自己的错误。因此为达到测试的目的，应由客观、公正、严格的独立的测试部门或独立的第三方测试机构进行测试。
- 尽量避免测试的随意性：应从工程的角度理解测试，它是有组织、有计划、有步骤的活动。

1.5软件测试误区

- 误区一：如果发布出去的软件有质量问题，都是软件测试人员的错。
- 误区二：软件测试技术要求不高，至少比编程容易多了。
- 误区三：有时间就多测试一些，来不及就少测试一些。
- 误区四：软件测试是测试人员的事，与开发人员无关。
- 误区五：根据软件开发瀑布模型，软件测试是开发后期的一个阶段。

测试策略与模型

02

2.1软件基本测试策略

- 确定测试目标
- 确认测试对象
- 建立测试生命周期
- 制定和实施测试策略
- 选择测试类型
- 运用测试方法

2.1.1制定和实施测试策略

- 确定测试由谁执行
- 确定测试什么
- 何时进行测试
  - 软件开发经验和事实证明，随着开发过程深入，越是早期没有进行测试的模块对整个软件潜在隐患及破坏作用就越明显。
- 确定怎样进行测试
  - 规格说明界定软件要做什么，而程序实现表达了软件怎样做。根据软件“规格说明”和程序实现的对应关系，确定采用何种测试策略和技术方法，设计并生成有效测试用例，对其进行测试。

2.1.2明确测试类型

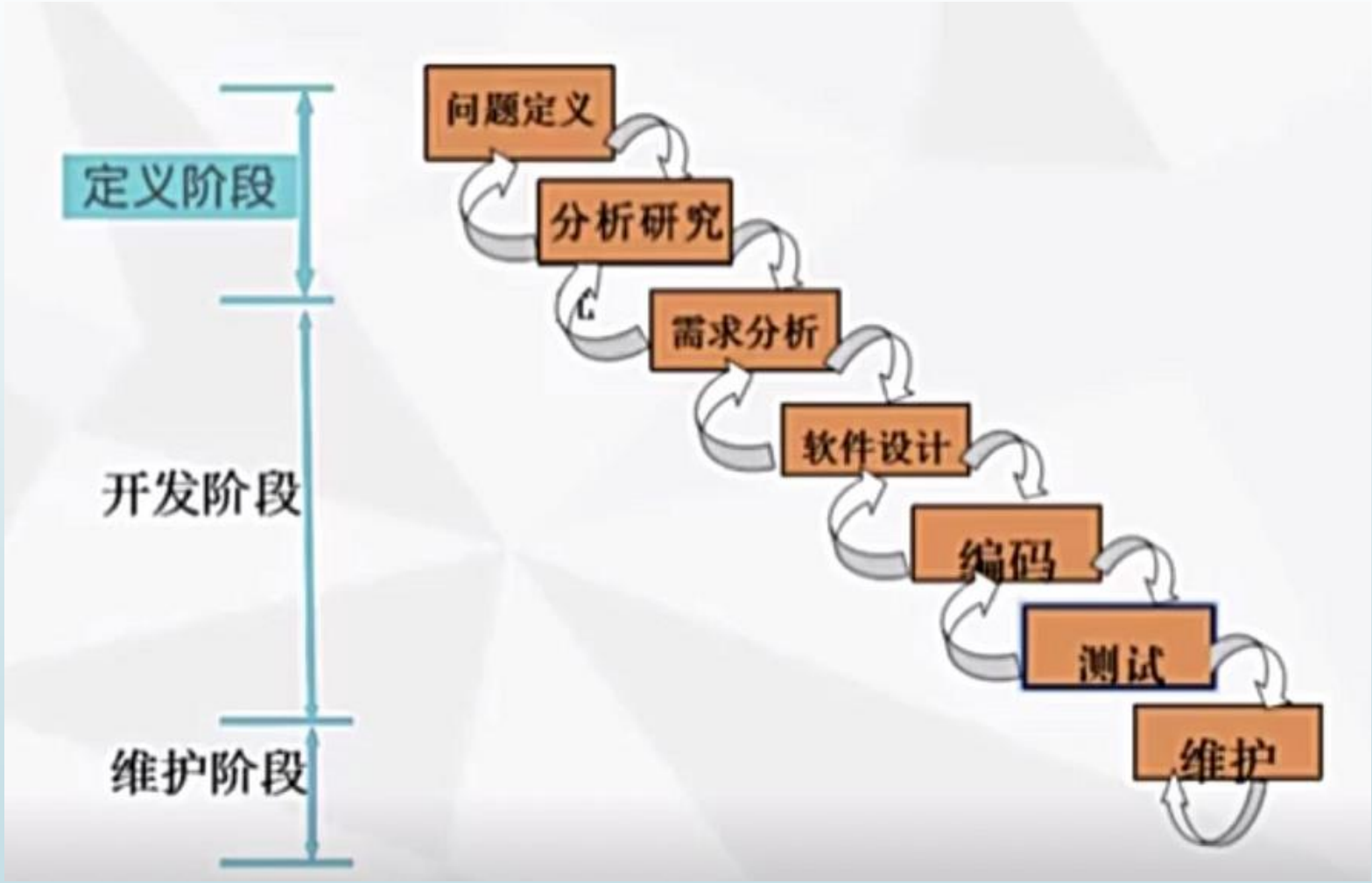
- 功能测试
- 非功能测试
- 恢复测试（Regression Testing）
  - 恢复测试首先采用各种办法强迫系统失败，然后验证系统能否尽快恢复。对于自动恢复需验证重新初始化、检查点、数据恢复和重新启动等机制的正确性。
  - 对人工干预的恢复系统，还需要评估平均修复的时间，确定其是否在可接受的范围内。
- 确认测试

2.2软件测试模型分析

- 软件开发模式
  - 从软件最初构思到发布软件产品的全过程。
- 主流开发模型：
  - 瀑布模型
  - 螺旋模型
  - RUP模型
  - 敏捷模型

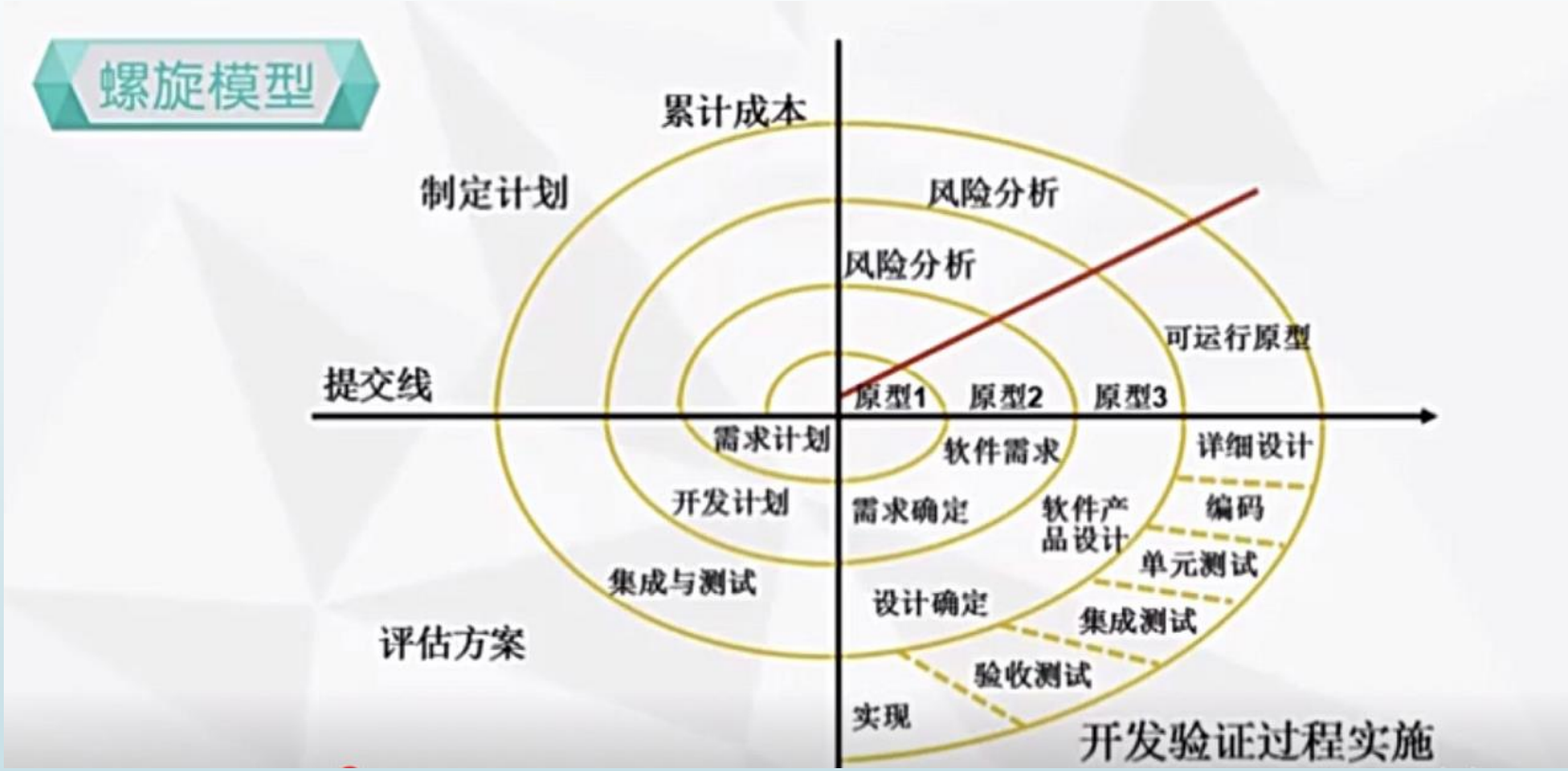
2.2.1瀑布模型

- 将软件生命周期各项活动规定为按固定顺序相连若干阶段性工作，最终得到软件产品。



2.2.2螺旋模型

- 初期无须定义细节，小规模初始定义主要功能并实现，评测风险与客户反馈，确定下阶段目标。
- 确定目标选择方案和限定条件。对方案风险进行评估并解决问题。进行本阶段开发与测试。计划如何进入下一阶段。确定进入下一阶段的方法。



2.2.3RUP模型（Rational Unified Process，统一软件过程）

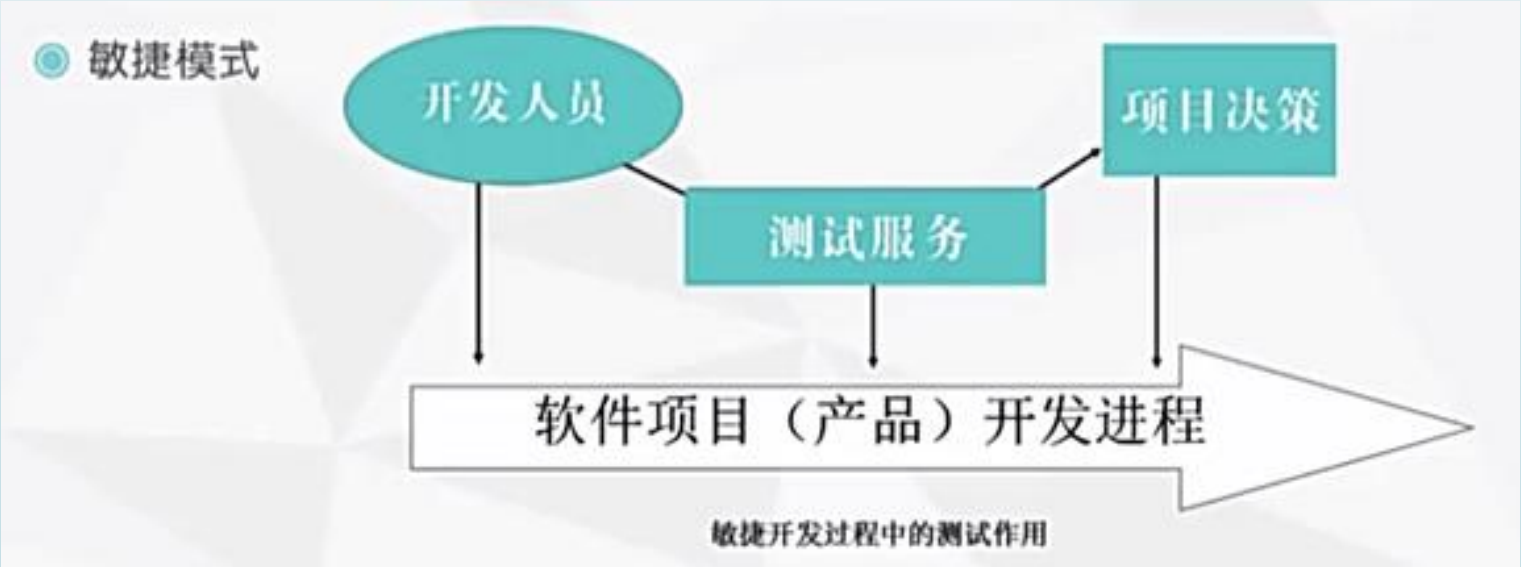
- 面向对象且基于网络的程序开发方法论。面向对象软件工程的通用业务流程。具有统一的软件流程构架。提供规范的开发任务分配和职责明确方法。确保可预期和预算内开发满足用户高质量产品。

2.2.4敏捷模式

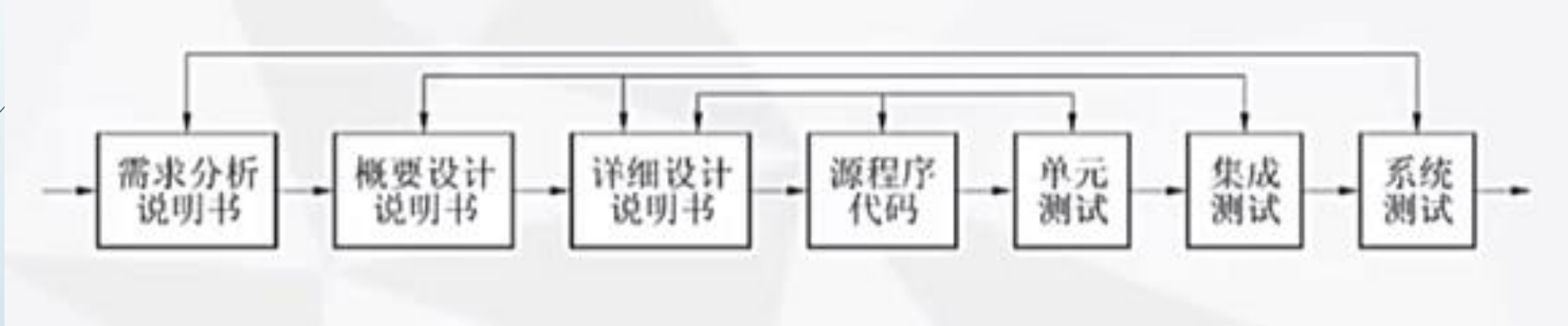
- 遵循敏捷原则：提倡简单、灵活与效率，符合敏捷价值。
- 交互胜于过程与工具；可运行的软件胜于完整文档；与客户合作胜于合同谈判；响应变化胜于教条遵循原计划。
- 迭代式增量开发过程（SCRUM）
- 特征驱动软件开发（FDD）
- 自适应软件开发（ASDI）
- 极限编程（XP）

2.2.4敏捷模式

- 敏捷模式，测试不完全依赖文档，需要自动寻找和挖掘更多关于程序的信息来指导测试。
- 敏捷中测试者需主动和开发者讨论软件需求和设计，研究缺陷出现的原因。
- 敏捷测试需实施持续测试、不断回归测试和快速测试。



2.3软件开发模式和测试关系



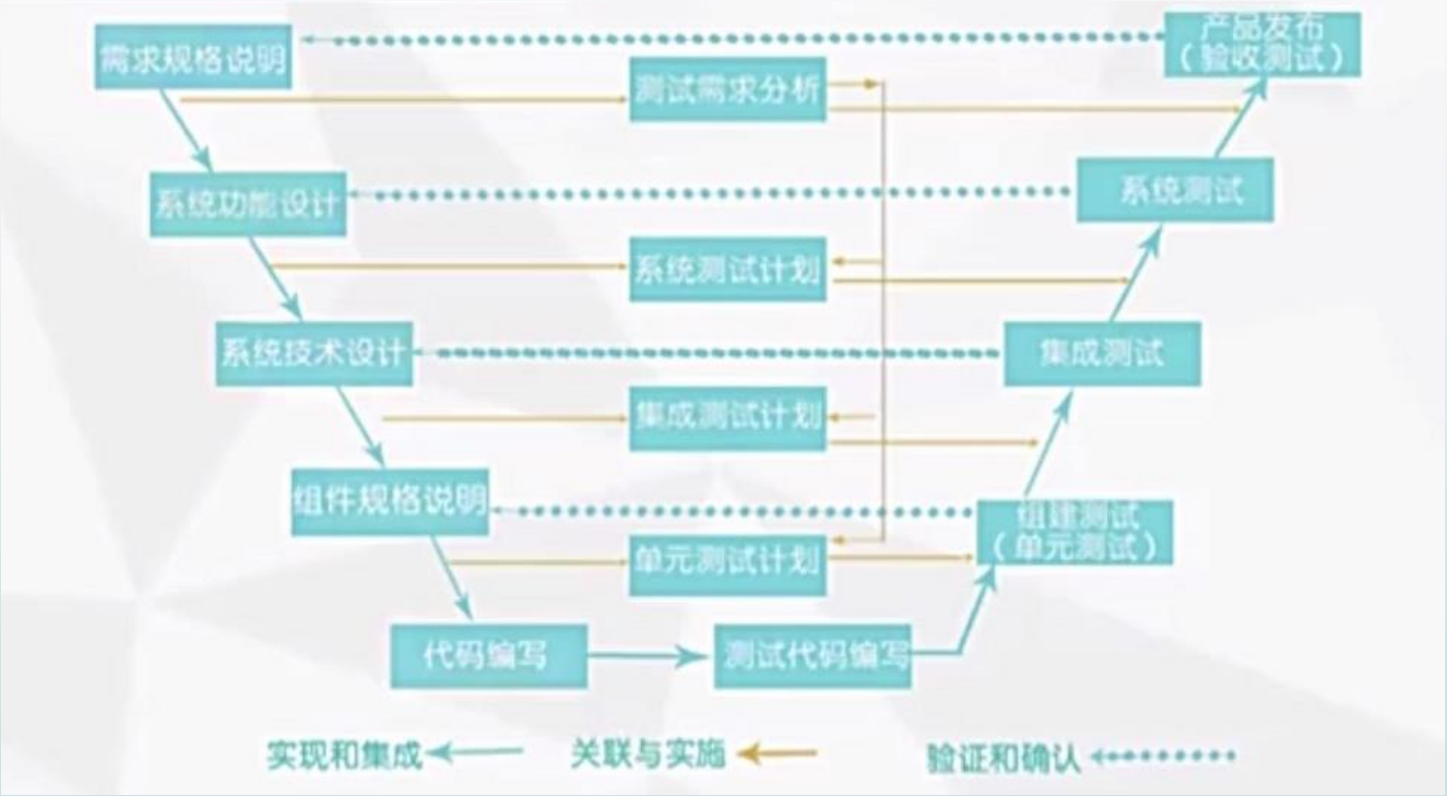
软件生命周期与测试

03

- 3.1软件生命周期
- 3.2软件测试通用流程
- 3.3组件测试、集成测试、系统测试、确认测试、验收测试、以及更新测试

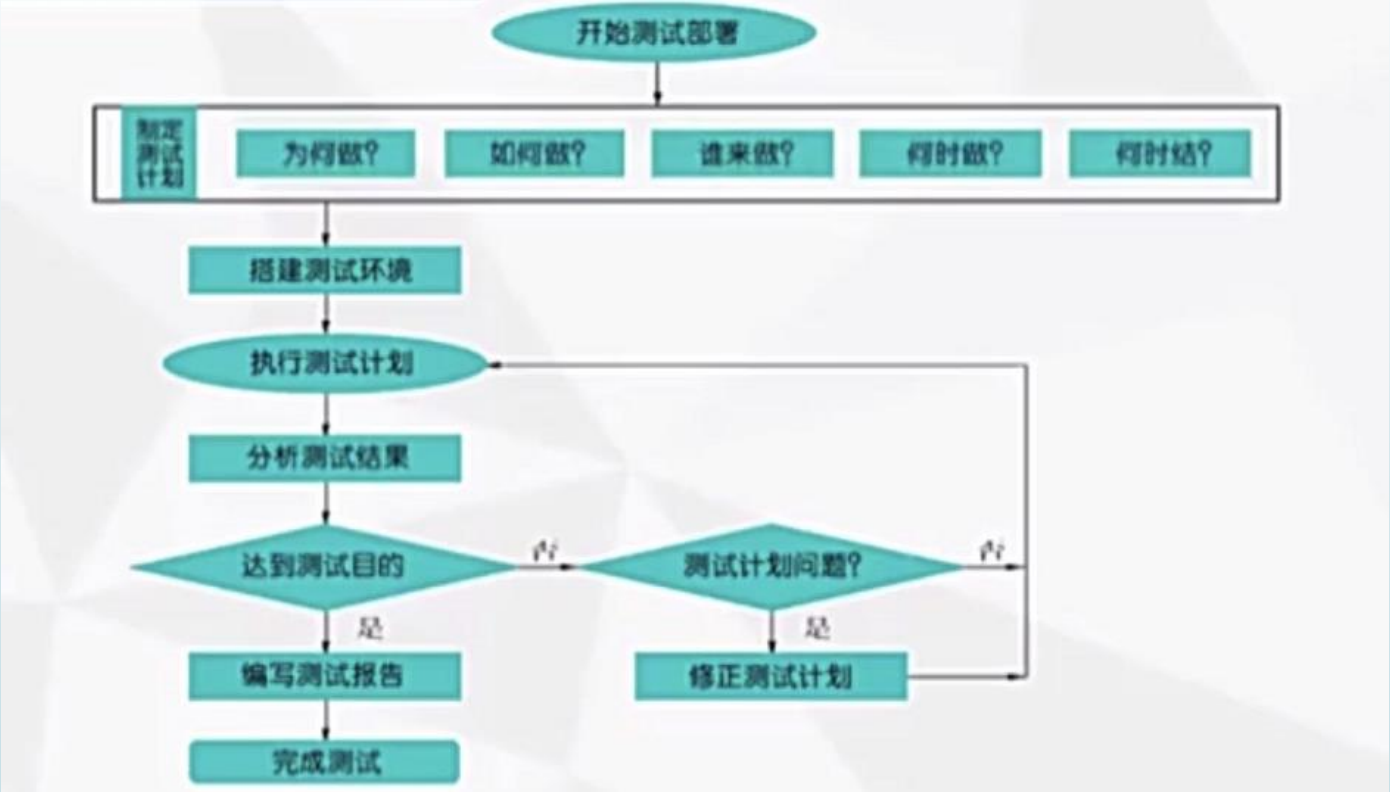
3.1软件生命周期

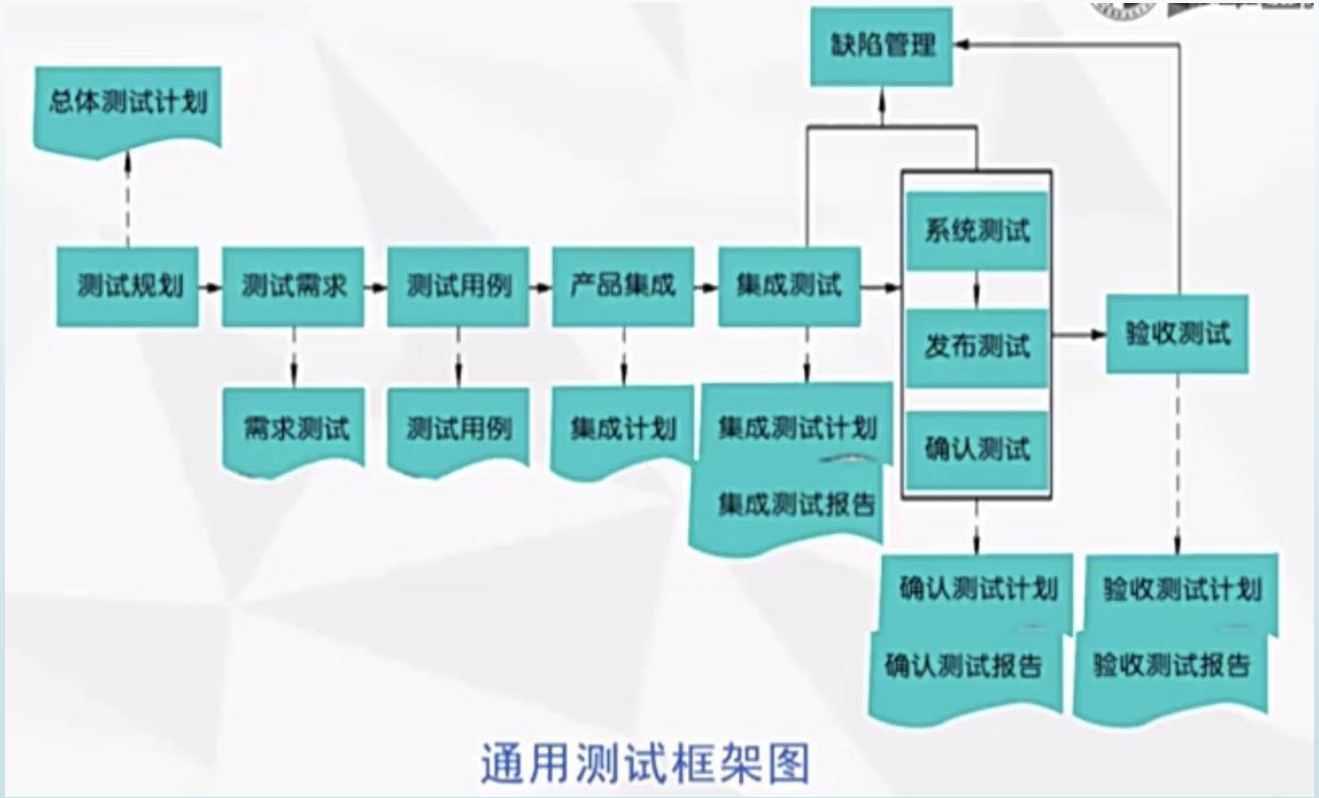
- V模型



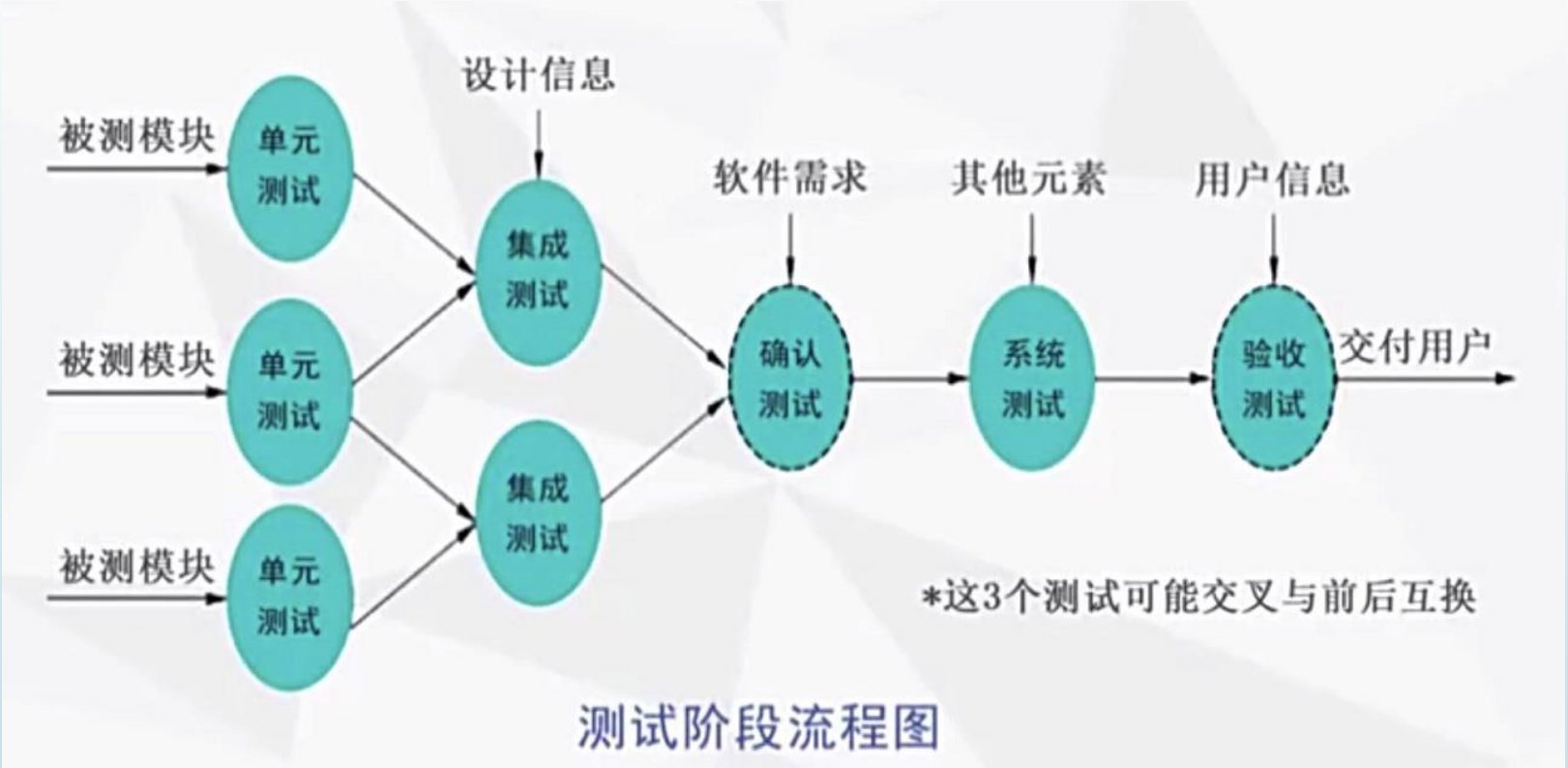
3.2软件测试通用流程

- 组织、规划、过程-部署管理
- 流程性框架、阶段性过程、阶段性流程




缺陷管理
总体测试计划
测试规划
测试需求
测试用例
产品集成
集成测试
系统测试
发布测试
确认测试
验收测试
需求测试
测试用例
集成计划
集成测试计划
集成测试报告
确认测试计划
确认测试报告
验收测试计划
验收测试报告

通用测试框架图


设计信息
被测模块
单元测试
被测模块
单元测试
被测模块
单元测试
集成测试
集成测试
软件需求
其他元素
用户信息
确认测试
系统测试
验收测试
交付用户
*这3个测试可能交叉与前后互换

测试阶段流程图

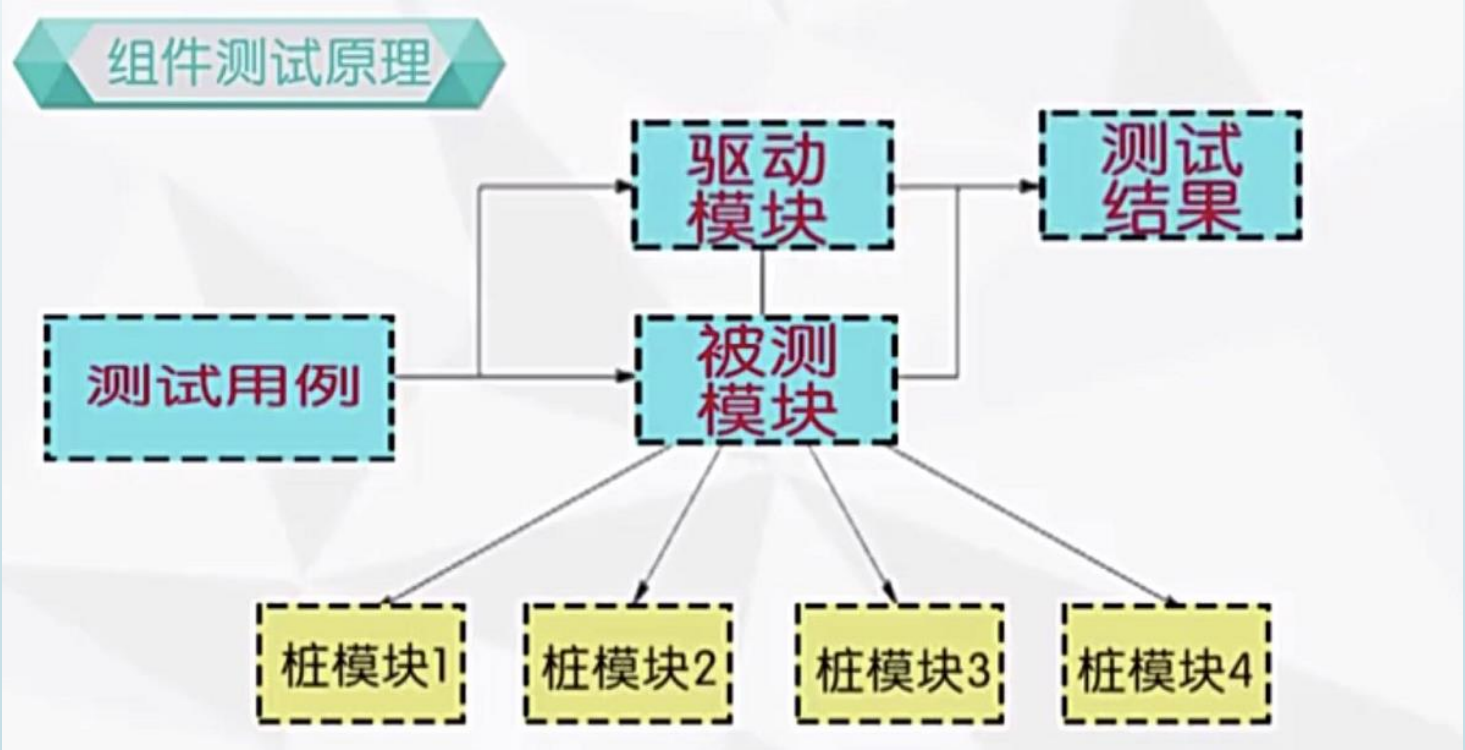
3.3.1组件测试

- 组件测试对象
  - V 第一层测试级别针对组件单元进行
  - 包含：模块、单元、程序或函数或类
  - 测试对象直接来自开发结果
- 组件测试特征
  - 组件测试通常由程序开发者完成（或在极限编程与测试驱动的开发中开发、测试人员结对完成）
  - 组件独立进行测试
  - 被测组件可为更小组件组成
  - 测试关注组件内部行为
  - 组件测试根据内容进行正确性检验
- 组件测试类别
  - 组件测试的广义和狭义
    - 狭义组件测试指编写测试用例进行验证被测代码正确性
    - 广义组件测试指编写测试用例进行测试，它小可到一行代码大可到功能模块验证。
  - 静态测试为代码走查或扫描而无需编译运行
  - 动态测试编写测试用例需编译运行并调用被测代码
  - 组件测试方式
  - 手工静态测试主要方式：代码评审审查
  - 动态测试通过执行测试用例的方式进行
  - 自动化测试主要运用测试工具，根据代码语法、词法、算法以及编码规范来识别潜在错误
  - 测试工具可动态生成组件测试用例

- 测试驱动模式
  - 测试提前到代码未产生之前进行
  - 测试模式将改变开发的过程与习惯
  - 开发对即将编写的代码程序需根据测试用例作细致考虑使得强迫性的在代码设计方案中实现这种策略
- 代码先行模式
  - 先编写代码后进行测试
  - 选择需要测试的重要代码进行测试
  - 这个方法对开发流程及习惯改变不大

- 模块接口与局部数据结构处理
  - 模块接口
    - 检查进出程序单元的数据流是否正确检查参数匹配与传送。
  - 局部数据结构
    - 检查内部数据是否保持完整性，检查类型、值、变量名及初始化。
- 路径测试与边界测试
  - 路径测试
    - 只要针对程序路径进行测试或覆盖。
  - 边界测试
    - 组件测试常需要进行边界测试，测试为限制的数据处理而设置的边界处理是否能够正常的工作。
- 检查程序关于出错处理
  - 出错处理测试重点是检查模块在工作中发生错误，而出错处理是否有效
  - 检验程序出错处理可能情况
- 组件测试的性能测试
  - 该测试进行与否，主要取决于组件对系统性能的影响程度
  - 性能测试使用代码效率测试工具进行
- 组件的测试与调试
  - 测试从已知条件开始，用预先定义的过程，可预知结果
  - 调试从未知条件开始，结束过程不可预计

- 组件测试管理
  - 测试若需合理、规范、高效进行，需制定一套测试管理规范
  - 管理规范通常由项目经理以上管理层发布，QA负责监督执行

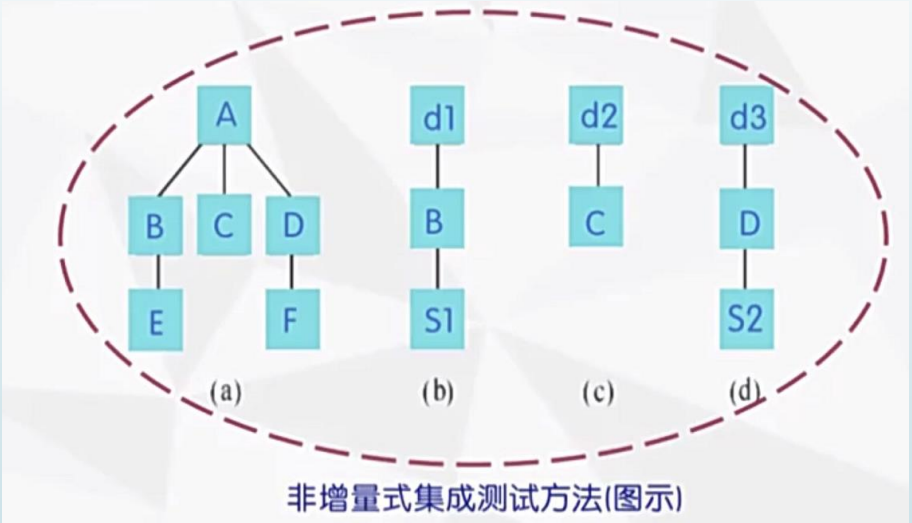


3.3.2集成测试

- 集成测试定义
  - 第二测试级别
  - 构成更大单元或子系统
  - 组件连接为集成
- 集成测试环境
  - 需要测试驱动器完成与“外部”接口
  - 驱动器向测试对象送测试数据
  - 接受数据并记录结果
  - 测试需从监视器读取和记录组件间的数据流
- 集成测试目标
  - 发现接口与相互之间协作问题及被集成部分之间的冲突
  - 检验数据交换和组件间通信存在问题
  - 当被测对象特性重要或风险较大时集成测试
  - 需进行非功能性测试

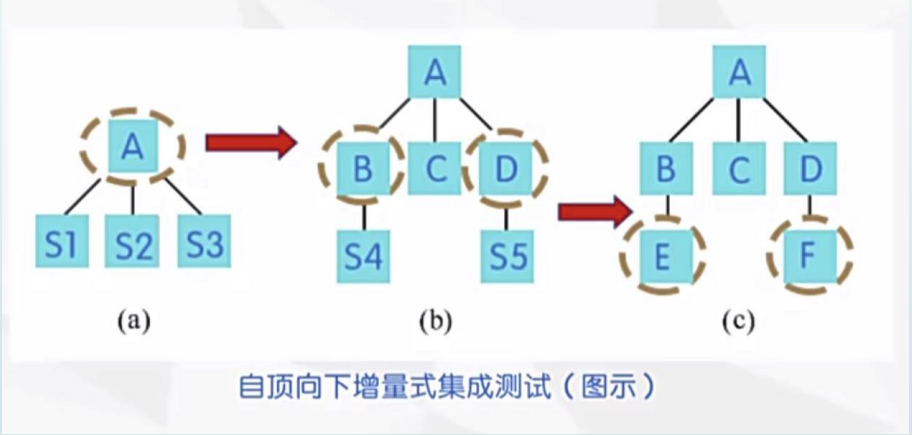
- 基本测试策略
  - 自顶向下或自底向上集成测试
  - 随意集成测试和中枢集成测试
  - 归纳为非增量方式和增量方式
- 无需使用桩模块

- 非增量式测试
  - 一步到位思想
  - 按程序结构连接
  - 整体测试完成



非增量式集成测试方法(图示)

- 增量式测试方式
  - 自顶向下增量：首集主控（主程序），依层次结构向下，按深度优先方式（纵向）或广度优先方式（横向）集成。
  - 自底向上增量：底层开始、自上而下、逐步集成、逐步测试、不断增量、直至完成



自顶向下增量式集成测试（图示）

集成测试方式比较

集成测试方式	优点	缺点
非增量式	先分散后集中，一次完成	存在错误则最后集成测试时暴露
增量式	逐步集成逐步测试，错误分散出现，便于找问题及修改，逐步集成测试多次检验，具优越性	
自顶向下式	自然做到逐步求精，开始即得系统框架	需提供桩模块，观察和分析测试输出较困难
自底向上式	驱动模块模拟所有调用参数，若关键模块在底部具优越性	

3.3.3 系统测试

- 系统测试对象
  - 软件生命周期第三级测试，集成测试完成后的产品，按系统结构测试
  - 目的检查集成后产品是否满足需求规格说明
- 系统测试环境
  - 需要独立测试环境
  - 对运行环境参数设置和配置的控制受限
  - 客户环境中其他系统影响测试条件
  - 系统测试执行需要重现
- 系统测试目标
  - 确认整个系统是否满足规格说明中的功能和非功能需求，以及满足的程度
  - 系统测试应发现需求不正确不完整或实现和需求之间不一致而引起的失效，并识别没有文档化或被遗失的需求

- 系统测试实践问题
  - 如没有系统需求，所有系统行为都正确，无法对系统进行评估
  - 不明确系统需求，系统测试只能时探测性测试
  - 错误的决策也将会导致系统测试失败，造成项目失败

- 功能性测试
  - 验证系统输入输出行为的各种测试
  - 测试基础为功能需求
  - 需求详细描述系统行为
  - 定义系统必须完成的功能
  - 功能性测试一般采用黑盒测试技术，采用人工/自动化测试的策略与手段
  - ISO/IEC 9126功能性测试：适应性、准确性、互操作性和安全性测试

- 功能性测试细分内容
  - 逻辑功能测试
  - UI测试
  - 易用性测试
  - 兼容性测试
  - 按照测试

- 非功能性测试（功能多好、产品满意度）
  - 性能测试
    - 性能测试可在测试过程中任一阶段执行，单独模块性能也可测试或评估。但整个系统所有成分集成后，才能测试出系统真正性能表现。
    - 定义：指软件系统的时间性能与空间性能的检测与验证。
      - 时间性能：软件的一个具体事务的响应时间
      - 空间性能：软件系统运行时所消耗的系统资源
  - 可靠性测试
  - 有效性测试
  - 梯度：
    - 一般性能测试
    - 可靠性测试
    - 负载测试和压力（或强度）测试
    - 安全性测试
    - 其他测试：恢复性测试、兼容性与数据转换测试、易用性测试、文档检查、可维护性检查

3.3.4确认测试、验收测试及更新测试

- 确认测试
  - 确认测试也称合格性测试
  - 常用黑盒测试方法
  - 确认测试实际可发生在各个测试阶段
  - 确认测试准则
    - 确认通过一系列证明软件功能和要求一致的黑盒测试过程
  - 已变更有关的回归测试
    - 当系统发生变化、缺陷被修正或软件增加新功能时，对变化部分须再测试，重新执行已有测试用例---程序被修改后重新测试的过程
  - 配置审查
    - 确认过程重要环节是配置审查
    - 配置审查确保已开发软件所有文档均已齐全并分类编目，足以支持运行软件维护
    - 文件资料：用户手册、操作手册、设计资料、源程序、产品发布版本及测试资料

- 验收测试
  - 检验软件产品质量的最后一项过程
  - 验收测试突出客户作用，并与开发人员参与的情况下开展
  - 验收测试可在多个级别进行
- 验收测试内容
  - 商业软件安装或集成时进行验收测试
  - 组件可用性验收测试在组件测试时进行
  - 系统测试前进行新功能验收测试（利用原型）
  - 进行功能测试
  - 特殊情况检查
  - 文档检查、强度检查、恢复测试、可维护性评价、用户操作测试、安全性测试

- 验收测试的形式
  - 根据合同进行的验收，验收判断合同是否满足
  - 用户（群）组织的验收，整个确认测试最后阶段
  - 验收测试包括备份/恢复（灾难恢复）、用户管理、维护任务和安全攻击检查
  - 现场测试（含α或β测试形式）
- 软件文档资料测试
  - 软件文档测试是测试工作规范化组成部分
  - 测试文档在软件开发初期需求分析阶段开始
  - 测试文档对测试阶段工作指导评价作用明显
  - 测试文档需遵循某种规范文件的编写标准
  - 文档检查：
    - 产品说明书的属性完整、准确、清晰
    - 描述是否精确、一致、贴切、合理
    - 文档的代码无关性检查
    - 文档可测试性检查

- 更新测试
  - 软件产品运行使用数年或更长时间
  - 软件维护：每个软件系统部署后，都需一定修正和改进，升级或功能扩展创建原产品的新版本，这种情况称软件维护或软件支持
  - 版本开发的测试：
    - 软件版本在不断计划和变更中出现
    - 软件版本的改进或版本发布，项目将重新启动，所有阶段重新进行，称迭代软件开发
    - 需对产品每个版本进行所有测试级别测试，所有改变都应回归测试
  - 增量开发测试
    - 增量开发项目不作为一整体完成，而由一系列较小开发或交付组成
    - 开发好的系统加入新增量，构成不断成长系统，通过尽早交付系统可用功能，可降低风险
    - 增量开发测试是持续集成测试和回归测试

测试用例

04

4.1黑盒测试用例设计方法

- 等价类划分法：将测试的范围划分成几个互不相交的子集
- 边界值分析法：选出的测试用例，应选取正好等于、刚刚大于、刚刚小于边界的值
- 错误推测法：在测试程序时，人们可以根据经验或直觉推测程序中可能存在的各种错误
- 判定表法：适合于逻辑判断复杂的场景，通过穷举条件获得结果，对结果再进行优化合并，会得到一个判断清晰的策略
- 正交实验法：在各因素互相独立的情况下，设计出一种特殊的表格，找出能以少数替代全面的测试用例
- 还有其它场景法和状态迁移法等

4.2白盒测试用例设计方法

- **（强度由低到高）语句覆盖、判定覆盖、条件覆盖、判定条件覆盖、条件组合覆盖、路径覆盖。**
- **语句覆盖：**设计若干个测试用例，运行被测程序，使得每一可执行语句至少执行一次。
- **判定覆盖：**使得设计的测试用例保证程序中每个判断的每个取值分支（t or f）至少经历一次。
- **条件覆盖：**是指选择足够的测试用例，使得运行这些测试用例时，判定中每个条件的所有可能结果至少出现一次，但未必能覆盖全部分支。
- **判定条件覆盖：**判定-条件覆盖就是设计足够的测试用例，使得判断中每个条件的所有可能取值至少执行一次，同时每个判断的所有可能判断结果至少执行，即要求各个判断的所有可能的条件取值组合至少执行一次。
- **条件组合覆盖：**在白盒测试法中，选择足够的测试用例，使所有判定中各条件判断结果的所有组合至少出现一次，满足这种覆盖标准成为条件组合覆盖。
- **路径覆盖：**是每条可能执行到的路径至少执行一次。

4.3测试用例八要素

- **用例编号：**规则是由字符和数字组成的字符串，具有唯一性，易识别性；
- **测试项目：**对应测试用例编号中的测试子项名系统测试；
- **测试标题：**体现测试出发点关注点以及测试用例预期的结果；
- **重要级别/优先级别：**重要级别一般分为高中低；
- **预置条件：**测试用例在执行时需要满足一些前提条件，环境的设置；
- **测试输入：**测试执行中需要加工的外部信息，避免用描述性语言，要具体，根据测试用例具体情况，有手工输入，文件，数据库记录；
- **操作步骤：**执行当前用例需要经过的操作步骤，需要明确的给出每一个步骤的描述；
- **预期输出：**需要判断测试对象是否正常工作。

5 Q&A


THANKS