COMP130015.02

软件工程

9. 软件测试

复旦大学计算机科学技术学院

沈立炜

shenliwei@fudan.edu.cn

软件缺陷与质量问题

- 人类依赖软件来实现越来越多的目标和任务
  - ✓ 最初局限于运行在计算设备中的各种计算任务
  - ✓ 如今已经深入渗透社会经济生活的各种信息处理、智能决策和自动控制任务
  - ✓ 支撑着交通、能源等传统基础设施以及在线支付等方面的新型信息化基础设施
- 不可避免的软件缺陷
  - ✓ 软件是一种复杂的人造物
  - ✓ 不可避免会因为人在认知和思维方式上的缺陷（例如理解和掌握复杂问题的局限性、偶然的疏忽等）而引入各种缺陷和质量问题
- 由此导致了各种事故甚至灾难性的后果

软件质量事故案例

1996年6月4日，**Ariane 5火箭**在发射37秒之后偏离飞行路线并发生爆炸。调查报告显示，事故是由于控制软件在将一个表示火箭水平速率的64位浮点数转换成16位有符号整数时发生溢出而导致的。发生这一问题的原因是开发人员在开发Ariane 4火箭控制软件时确定这个值不会超出16位整数，然而Ariane 5火箭的速度提高了近5倍。不幸的是，开发人员在开发Ariane 5火箭控制软件时复用了这段代码，并且没有对代码所基于的假设进行检查。

2007年10月30日，**北京奥运会第二阶段门票销售系统**刚启动就因为购票者太多而被迫暂停。经过评估，是由于开发团队低估了观众购票的热情，以至于当时的系统设计方案无法支持大规模的用户并发访问请求，从而导致售票系统瘫痪。

2012年1月9日，刚进入春运期间的**12306火车票网上订票系统**点击量超过14亿次，导致系统出现登录缓慢、支付失败、扣钱不出票甚至系统崩溃等严重问题。2012年9月20日，中秋和国庆黄金周出行人群集中购票导致系统日点击量达到14.9亿次，再次出现网络拥堵、重复排队等现象。这一系列故障的根本原因在于系统架构规划以及客票发放机制存在缺陷，无法支持如此大并发量的交易。

2018年3月18日，**美国优步公司的一辆沃尔沃XC90汽车**在测试时发生交通事故，导致一名49岁女性行人死亡。车祸发生时，涉事车辆正处于自动驾驶模式，车内配有一名安全操作员。调查报告认为，涉事车辆的自动驾驶软件存在几处设计缺陷，包括不能识别在人行横道以外出现的行人。

关于软件测试的观点

早期的软件测试主要依靠错误猜测和经验推断，并未形成系统化的方法与过程。随着软件工程研究与实践的积累，学术界和工业界逐渐形成了一些软件测试理论和方法。

20世纪70年代初Bill Hetzel博士提出的观点：软件测试是以评价一个程序或软件系统的质量或能力为目的的一项活动

这种观点认为软件测试的目的是验证软件的正确性。然而，由于软件自身的逻辑复杂性以及运行时近乎无穷的可能性，验证软件的正确性几乎是不可行的。

Glenford J. Myers在20世纪70年代末所提出的观点：软件测试是以发现错误为目的而执行一个程序或系统的过程

这种观点认为软件测试的目地是为了发现软件中的错误，只要发现的错误足够多那么开发人员就可以消除大部分软件质量问题。为此，软件测试人员应当采用各种“破坏性”的思维和手段（例如尝试极端的输入和操作方式）来暴露软件系统中潜在的问题。这种观点代表着软件测试理论和实践的主流。

软件测试的定义

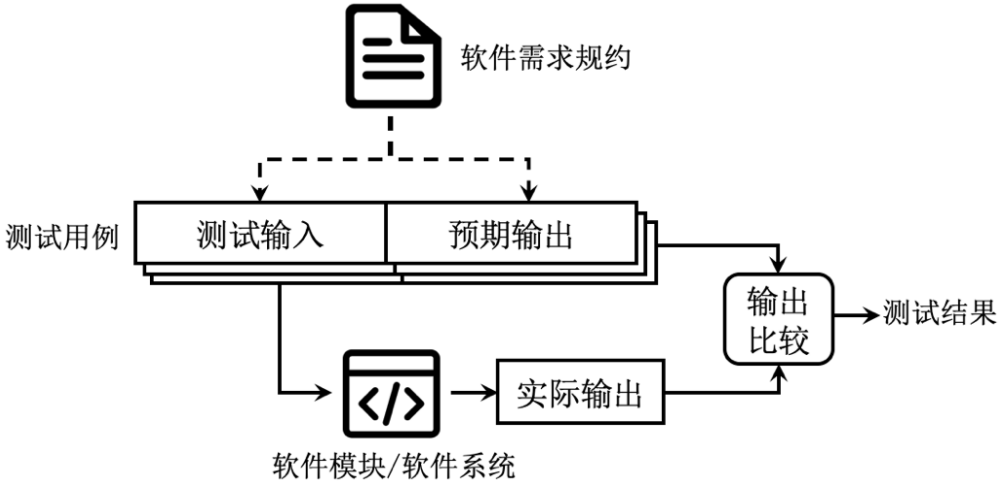
- ISO/IEC/IEEE系统和软件工程术语标准中的定义：将软件系统或组件在指定条件下执行，观察或记录执行结果，并对系统或组件的某些方面进行评估的活动
- ISTQB（International Software Testing Qualifications Board）的定义：由生命周期内所有静态和动态活动组成的过程，这些活动包括计划和控制、分析和设计、实现和执行、评估出口准则和编写报告、测试的收尾工作以及对软件产品和相关的工作产品的评估，目的是发现软件系统中的缺陷、提供涉众对软件系统质量的信心，以及预防软件系统中的缺陷
- IEEE SWEBOK（软件工程知识体系）给出的定义：一个动态的过程，它基于一组有限的测试用例执行待测程序，目的是验证程序是否提供了预期的行为

软件测试的几个重要方面

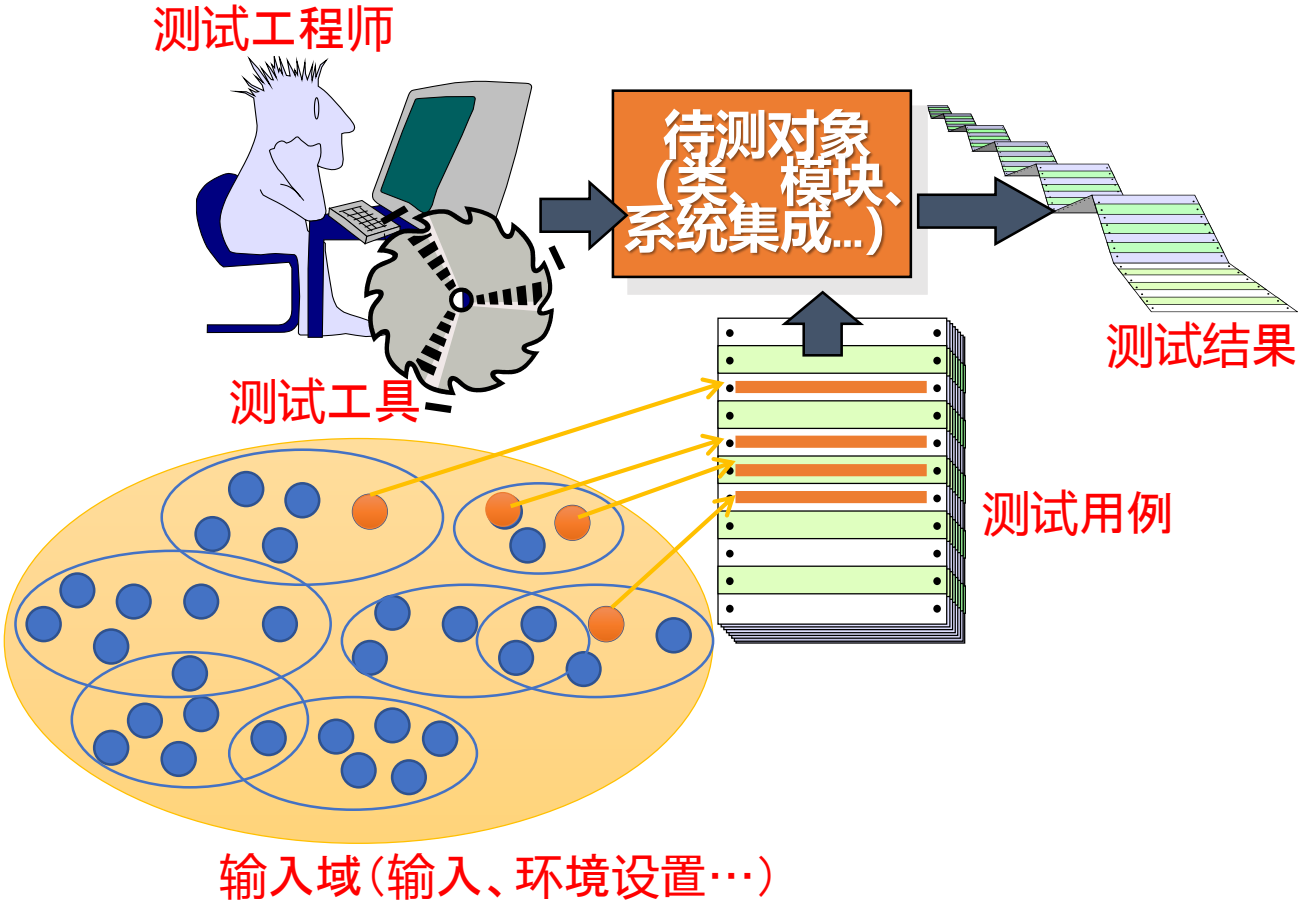
- 软件测试的几个关键理解
  - ✓ 对软件进行人工（人工测试）或自动（自动化测试）执行
  - ✓ 基于预期的软件行为进行的，通常来自于软件需求规格说明或其他要求
  - ✓ 需要对软件在测试中的实际行为和预期的行为进行比较，既包含输出结果比较又包括对于响应时间等指标的比较
  - ✓ 主要目的还是发现问题，找到差距
  - ✓ 充分的软件测试可以在一定程度上让我们建立起对于软件质量的信心
- 广义和狭义上的软件测试
  - ✓ 广义：涵盖了一切针对软件的质量检查和评价，除了动态的测试外还包括代码评审、静态扫描、形式化分析等
  - ✓ 狭义：通过人工或自动化的方式运行软件并对结果进行观察和分析，以此来发现潜在的软件功能性或非功能性缺陷

软件测试的基本过程

- 被测软件可以是完整的软件系统也可以是某个层次上的软件模块
  - ✓ 方法或函数：测试输入是传递给方法或函数的实参，预期结果是期望的方法或函数返回值
  - ✓ 类：测试输入可能包括对这个类的对象初始化和一系列后续方法调用以及各个方法调用的参数，预期结果是各方法调用返回值以及在此过程中的对象状态的变化
  - ✓ 完整软件系统：测试输入一般是一个用户操作序列，预期输出可以是系统通过界面展现的处理结果



测试：基于采样的过程



软件测试用例的数量

- 测试用例需要根据需求规格说明进行设计
  - ✓ 针对一个软件系统的测试用例中的操作步骤可能来自于需求规格说明中所定义的用户使用场景
  - ✓ 测试输入和期望输出则来自于需求规格说明中的功能定义
- 测试用例数量取决于多方面因素
  - ✓ 待测软件的复杂性：待测软件的功能越复杂，操作和输入组合的数量越多，所需要的测试用例数量就越多
  - ✓ 测试的严格性要求：待测软件的严格性要求越高，所需要的测试用例数量就越多
  - ✓ 所采用的测试技术：软件测试存在多种不同的测试方法和技术（例如黑盒测试、白盒测试等），利用不同方法和技术得到的测试用例数量也会有所区别

测试完备性与成本的权衡

根据质量准则和时间/成本约束进行仔细的权衡决策

覆盖度、完备性、充分性

成本、交付时间

课堂讨论：软件测试的充分性

课堂讨论：如何定义软件测试的充分性（覆盖度、完备性）？如何在控制成本的情况下提高软件测试的充分性?

软件测试的执行方

- 软件开发人员自身进行测试
  - ✓ 优势是熟悉软件功能和实现方式，容易进行针对性的深入测试
  - ✓ 劣势是不熟悉测试方法和技术，同时容易受自身思路的限制
  - ✓ 开发者测试的主要目的是确定软件开发任务已完成、软件具备可运行的条件，一般包括单元测试和集成测试，且可以与测试驱动的开发相结合
- 企业内专门的测试人员进行测试
  - ✓ 优势是熟悉测试方法和技术，同时可以用一种“严苛”的眼光进行测试
  - ✓ 劣势是不熟悉软件的功能逻辑和实现方式
  - ✓ 一般负责系统测试和验收测试等更高级别的测试，其测试目的主要是为了发现软件缺陷
- 外部第三方测试机构进行测试
  - ✓ 针对需要权威认证和确认的软件进行测试并提供相关证明材料

软件测试的基本原则

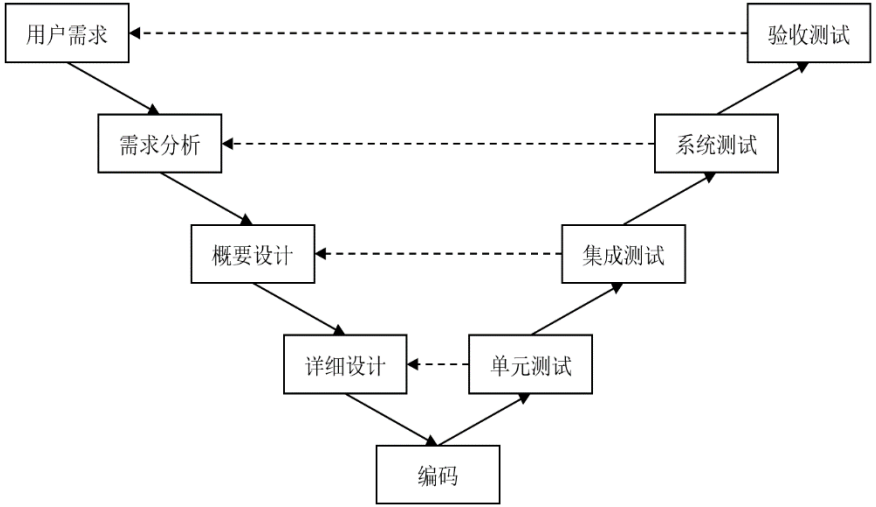
- 测试只能揭示软件中的缺陷，不能证明软件的正确性
- 穷举测试是不现实的
- 尽早开始测试
- 缺陷经常是聚集分布的（80%的缺陷集中在20%的软件模块中）
- 测试中的杀虫剂悖论（软件经受的测试越多，对于测试人员的测试就具有越高的免疫力）
- 根据不同软件的特点开展测试活动
- “不存在缺陷的系统就一定是有用的系统”是一个谬论

软件测试过程模型

- 软件测试并不仅仅是一个活动，而是包含一系列不同类型测试活动的过程，例如单元测试、集成测试、系统测试等
- 软件测试与软件开发过程紧密结合，各种测试活动穿插在整个开发过程中，从而衍生出了各种软件测试过程模型
  - ✓ V模型
  - ✓ W模型
  - ✓ 敏捷测试模型

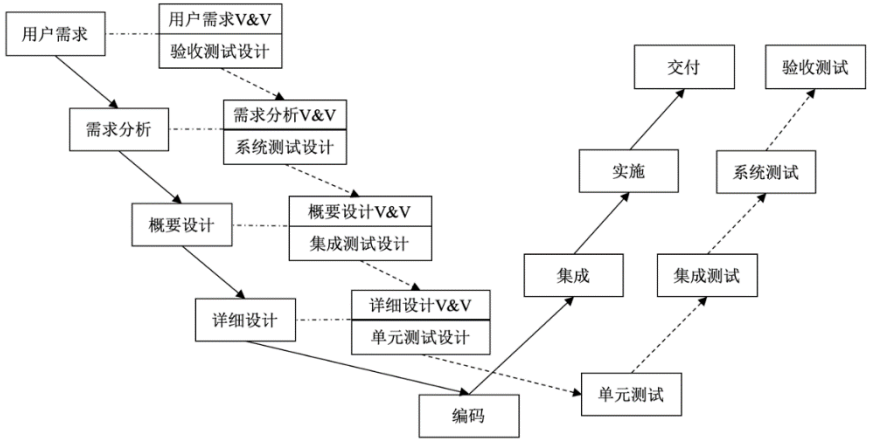
V模型

- V模型是与瀑布模型相配套的测试过程模型，因此不可避免带有很多与瀑布模型类似的局限性
  - ✓ 阶段性的测试过程使得V模型主要适合于瀑布模型等串行的开发过程，不适合需求变化频繁的软件系统
  - ✓ 测试活动在编码之后才会进行，这使得早期需求和设计阶段引入的缺陷发现时间较晚，从而导致对应的缺陷修复成本较高



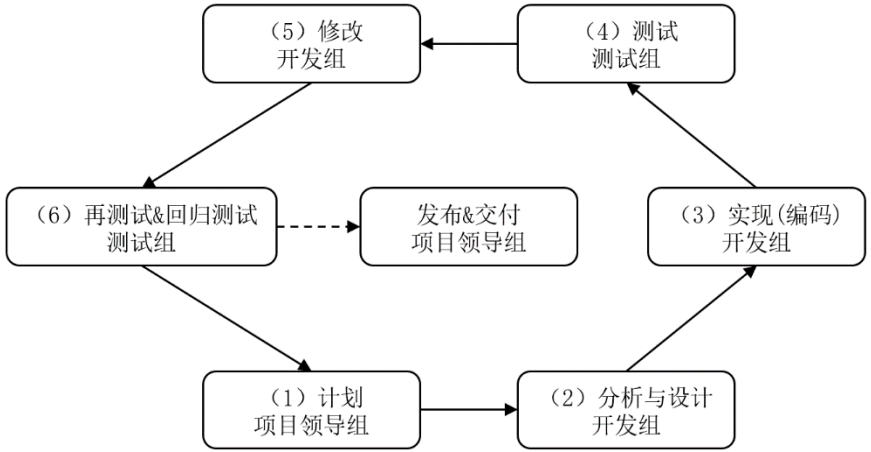
W模型

- 在V模型基础上增加了各个开发阶段中应同步进行的验证（verification）和确认（validation）活动
  - ✓ 强调测试是伴随整个软件开发周期的持续过程，其中不仅包括对于软件实现的测试，而且还包括对于需求和设计规格说明等早期软件开发制品的验证和确认活动
  - ✓ 与V模型类似，W模型中的测试仍旧保持严格的顺序关系，只有当前一阶段完成后才能进入下一阶段，这种串行的测试过程难以支持频繁的需求变更和迭代化的开发过程



敏捷测试模型

- 与传统的v模型和w模型不同，敏捷软件测试过程中的测试活动不再是一个个独立的阶段，而是随着迭代化的过程持续进行的活动
- 每一次迭代往往交付较少的新功能或特性，测试人员要在有限时间内对新功能或特性进行测试从而确保能频繁交付可运行软件



软件测试类型

- 单元测试：针对类、模块等代码单元的测试，往往属于开发者测试
- 集成测试：针对不同类或模块的逐步集成及相应的测试，往往属于开发者测试
- 系统测试：将整个软件系统作为一个整体并考虑具体的系统运行环境的测试
- 验收测试：确认软件系统是否完成了用户所提出的需求

单元测试

- 单元是软件系统中不同层次上的组成部分
  - ✓ 最小的单元可以是方法或函数
  - ✓ 稍大一些的单元可以是类或文件
  - ✓ 更大的单元可以是模块或组件
- 软件开发中的各种软件单元也需要先进行单元测试，然后再逐层逐级组合得到更大的软件单元直至完整的软件系统
- 随着敏捷和测试驱动开发（TDD）的兴起，单元测试已被广泛认为是开发活动的一部分
  - ✓ 作为开发者测试，由开发人员进行测试用例设计和执行
  - ✓ 需要一整套框架的支持，包括测试用例的执行和测试桩等，其中涉及单元测试工具（例如JUnit）以及Mock工具（例如JMockit）

单元测试的测试目标-1

- 代码单元接口
  - ✓ 接口是否可以像预期那样接收正确的输入数据并返回正确输出数据
  - ✓ 考虑接口方法或函数的各种输入组合以及相应的期望输出
  - ✓ 考虑不同接口方法或函数的典型调用序列（对于类或文件）
  - ✓ 观察测试结果时，除了考虑方法或函数返回结果外，还需要注意代码单元调用其他代码单元或外部API时的参数传递是否正确、文件访问和网络传输等外部IO操作结果是否符合预期、资源使用是否正确
- 局部数据结构
  - ✓ 局部数据结构在程序执行过程中的使用和更新方式是否符合预期，数据内容是否保持正确、完整
  - ✓ 例如，类属性在使用前是否进行了初始化，类中的堆栈结构是否发生溢出、覆盖或丢失

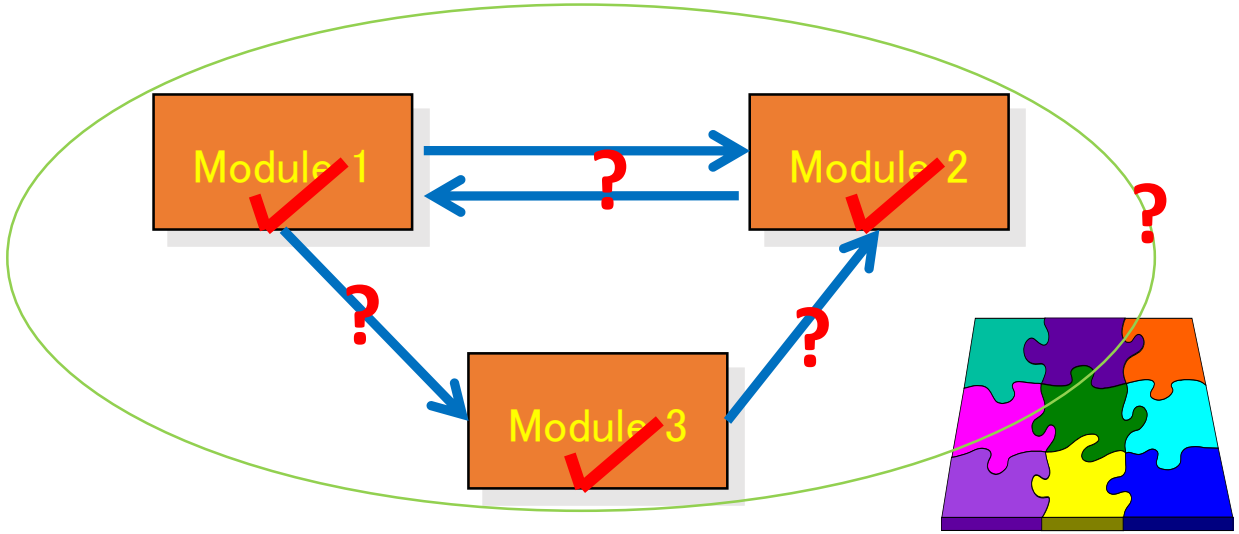
单元测试的测试目标-2

- 代码中的边界条件
  - ✓ 代码单元中的各种边界条件附近可能经常会隐藏缺陷，因此需要在测试中特别加以注意
  - ✓ 这里的边界条件包括方法或函数输入参数取值范围的边界以及程序执行控制流的边界等
- 代码执行路径
  - ✓ 一些缺陷仅当某些特定程序路径被执行时才会暴露出来
  - ✓ 应当尽量将各个程序路径都执行到从而发现更多的缺陷
- 错误及异常处理
  - ✓ 代码在遇到非法输入或外部异常时应当以正确的方式进行处理，因此需要对错误及异常处理方式进行测试
  - ✓ 需要考虑出现错误或异常时是否进行了正确的内部处理，是否正确给出了错误信息，是否按照预期抛出了异常等

集成测试

- 多个通过测试的代码单元组合在一起并不能保证没有问题，例如
  - ✓ 将销售模块与发货模块进行组合时，发货模块可能因为无法解析销售模块提供的地址格式而无法正确执行发货业务
- 这种缺陷是由于代码单元对于相互间的接口理解不一致导致的，需要通过集成测试发现
- 集成测试，也称为组装测试或联合测试
  - ✓ 将已分别通过测试的软件单元按照设计方案进行集成同时进行测试
  - ✓ 其主要目的在于检查参与集成的软件单元之间的接口是否存在问题
- 代码单元的集成方式以软件设计方案为基础

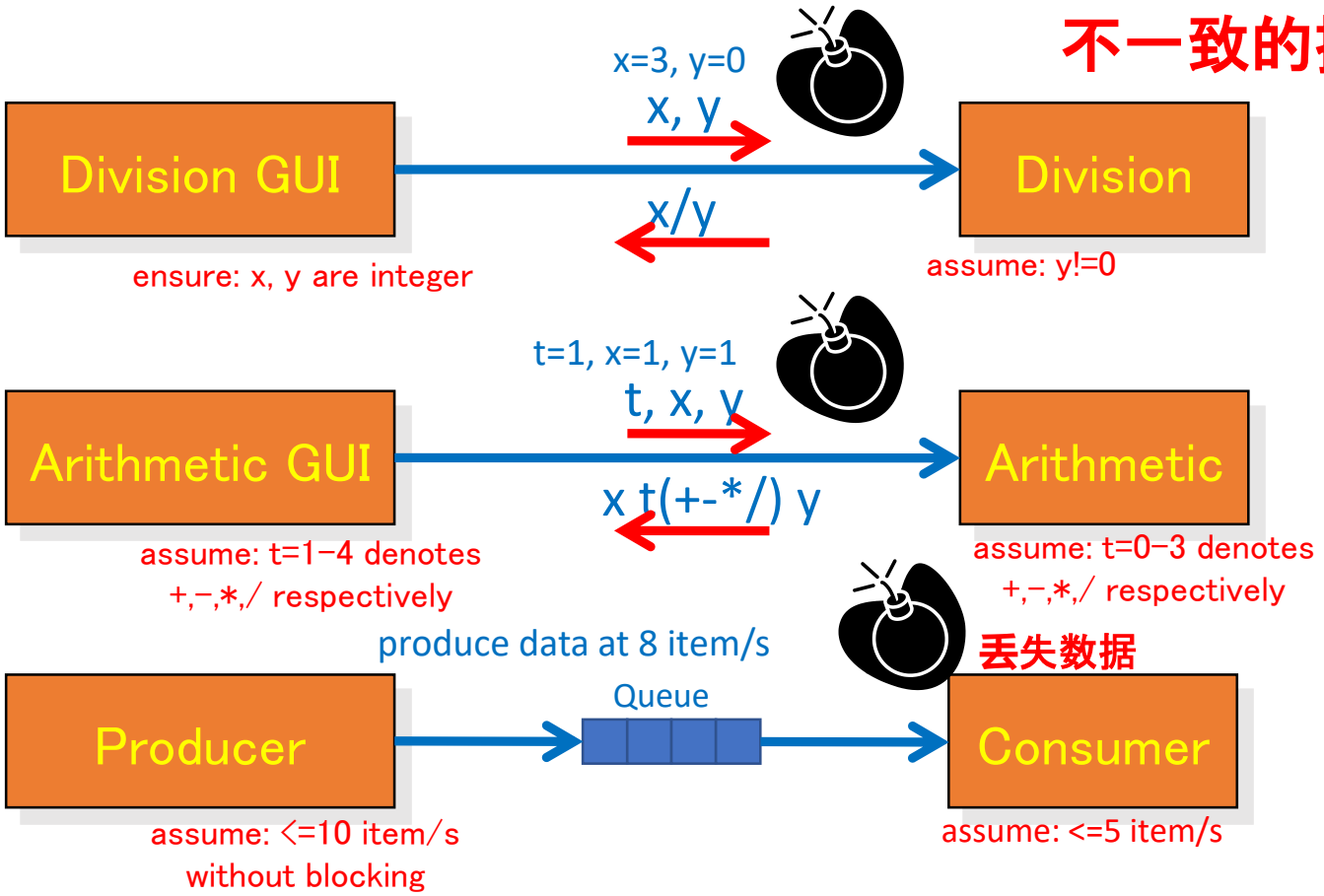
从单元测试到集成测试



集成测试:根据软件体系结构设计构造完整系统并在此过程中进行测试，以发现与接口和交互相关的问题

目标:在通过单元测试的软件组件基础上根据软件体系结构设计构造完整的软件系统

集成错误示例


不一致的接口假设
x=3, y=0
x, y
x/y
Division GUI
Division
ensure: x, y are integer
assume: y!=0
t=1, x=1, y=1
t, x, y
x t(+-*/) y
Arithmetic GUI
Arithmetic
assume: t=1-4 denotes
+,-,*,/ respectively
assume: t=0-3 denotes
+,-,*,/ respectively
produce data at 8 item/s
丢失数据
Queue
Producer
Consumer
assume: <=10 item/s
without blocking
assume: <=5 item/s

集成测试针对的主要错误来源

- 数据在穿越软件单元之间的接口传递时发生丢失或明显延迟（例如通过网络传输）
- 不同软件单元对参数或值理解不一致，这种问题不会导致软件单元之间调用或通信失败，但可能导致不一致的处理逻辑
- 软件单元由于共享资源或其他原因而存在相互影响和副作用
- 软件单元交互后进行计算的误差累计达到了不能接受的程度，或者接口参数取值超出取值范围或者容量
- 全局数据结构出现错误，使得不同软件单元之间无法按照统一的标准进行计算
- 软件单元使用未在接口中明确声明的资源时，参数或资源造成边界效应

集成测试策略-1

- 大爆炸式（Big Bang）的集成策略
  - ✓ 将所有通过单元测试的软件单元一次性按照设计方案集成起来进行测试
  - ✓ 优点：简单、易行，所用的测试用例数量较少，因此可以缩短测试时间
  - ✓ 缺点：参与集成的软件单元数量众多，无法对各种组合情况进行充分测试，发现错误后难以进行定位
- 自顶向下的增量集成策略
  - ✓ 以主控单元为起点，自上而下按照深度优先或者广度优先的次序逐个组合各个软件单元
  - ✓ 每次组合一个新的软件单元时就执行一次集成测试，因此能够逐一检查软件单元之间的接口交互是否符合预期
  - ✓ 优点：便于测试人员进行故障隔离和错误定位
  - ✓ 缺点：需要为每个单元的下层单元构造测试桩，要求主控单元易于测试，可能导致对底层单元的测试不够充分

集成测试策略-2

- 自底向上的增量集成策略
  - ✓ 从系统概要设计所定义的软件层次结构的最底层单元开始进行组装和集成，每组合一个新的软件单元就进行一次集成测试
  - ✓ 优点：可以尽早地验证底层软件单元的行为，提高测试效率，减少了开发测试桩的工作量，便于测试人员对错误进行定位
  - ✓ 缺点：最后看到整个系统的框架，在测试过程的后期才能发现与时序和资源竞争相关的问题，增加了开发测试用例的工作量和成本，不能及时发现高层单元设计上的错误
- 三明治集成策略
  - ✓ 一种混合的渐增式策略，综合了自顶向下和自底向上两种集成策略的优点
  - ✓ 优点：较早地测试高层单元，也可以将低层单元组合成具有特定功能的单元簇并加以测试，如果运用一定的技巧就有可能减少测试桩和测试用例的开发成本
  - ✓ 缺点：是中间层可能无法尽早得到充分的测试

系统测试

- 系统测试在集成测试完成后进行
- 将整个软件系统作为一个整体并考虑具体的系统运行环境，在此基础上确定软件系统是否符合需求规格说明中各个方面的要求
- 目标和任务
  - ✓ 一方面，系统测试针对特定的质量属性开展，例如性能测试、兼容性测试、安全性测试等
  - ✓ 另一方面，系统测试强调软件运行所处的整个系统环境（包括硬件、网络、物理环境和设备、用户等）
- 包括一系列测试活动，分别针对不同的软件质量特性采用相应的测试技术和手段

不同类型的系统测试-1

- 功能测试
  - ✓ 根据软件系统的需求规格说明和测试需求列表，验证软件的功能实现是否符合要求
  - ✓ 系统级的功能测试需要将软件系统置于一个实际的应用环境中，模拟用户的操作实现端到端的完整测试，以确保软件系统能够正确地提供服务
  - ✓ 主要的测试目标涵盖功能操作、输出数据、处理逻辑、交互接口等多个方面
- 性能测试
  - ✓ 目的是在真实环境下评估系统性能以及服务等级的满足情况，同时分析系统性能瓶颈以支持系统优化
  - ✓ 关注于响应时间、吞吐量、负载能力等性能指标
  - ✓ 系统性能测试环境应当尽量与产品的真实运行环境保持一致，并模拟一些可能出现的特殊情况

不同类型的系统测试-2

- 兼容性测试
  - ✓ 旨在验证软件系统与其所处的上下文环境的兼容情况，即系统在不同环境下，其功能和非功能质量都能够符合要求
  - ✓ 主要针对硬件兼容性、浏览器兼容性、数据库兼容性、操作系统兼容性等方面展开测试工作
- 易用性测试
  - ✓ 针对软件产品易理解性、易学习性、易操作性等方面的测试
  - ✓ 与人机交互及用户的主观感受相关，因此一般需要模拟用户对系统进行学习和使用并对参加测试的人员的主观感受和客观学习和使用情况（例如学习时间、界面操作等）进行分析

不同类型的系统测试-3

- 可靠性测试
  - ✓ 可靠性是指软件系统在特定条件下及特定时间内正常完成特定功能或提供特定服务的能力，一般可以用概率来度量
  - ✓ 软件可靠性不仅与内部的实现方式及缺陷相关，而且也与系统环境、使用方式及系统输入相关
  - ✓ 一般需要模拟高强度以及持续的系统访问和使用，并分析系统正常运行并提供服务的概率
- 信息安全测试
  - ✓ 验证软件系统的信息安全等级并识别潜在信息安全问题的过程，其目的是发现软件自身设计和实现中存在的安全隐患和漏洞，并检查系统对外部攻击和非法访问的防范能力
  - ✓ 包括物理环境的安全性（物理层安全）、操作系统的安全性（系统层安全）、网络的安全性（网络层安全）、应用的安全性（应用层安全）及管理的安全性（管理层安全）

验收测试

- 在系统测试完成之后进行，其目的是确认软件系统是否完成了用户所提出的需求
- 站在用户的角度对软件进行检验，因此一切的判别标准是由用户决定的
- 通常包含α测试与β测试两个阶段
  - ✓ α测试：软件开发企业组织用户或内部人员模拟各类用户对即将面市的软件产品（称为α版本）进行测试，试图发现问题并对其进行修复，通常在企业内搭建的与实际应用相类似的软件运行环境中执行
  - ✓ β测试：软件开发公司组织各方面的典型用户在日常工作中实际使用β版本，并要求用户报告异常情况，例如一些互联网企业会将软件产品的β版本发布于网络进行公测

课堂讨论：从小到大的软件测试

课堂讨论：软件测试为什么要从小（单元测试）到大（系统测试和验收测试）逐步进行?

黑盒软件测试方法

- 测试人员将被测软件整体视为一个仅能通过外部接口交互的封闭黑盒，无法查看同时也无需了解其中的实现细节（如代码控制结构）
- 依据被测软件的规格说明设计测试用例
  - ✓ 适用于单元测试、集成测试、系统测试、验收测试等不同层次的测试，对应的外部接口的含义也有所不同
  - ✓ 例如，针对类测试时面对的外部接口是类接口，针对系统测试时面对的外部接口是用户界面和其他系统接口
- 黑盒测试面临的主要问题
  - ✓ 被测软件的输入空间非常大，因此需要选取一定数量的输入数据（包括多个输入参数的组合）作为测试用例
  - ✓ 需要权衡测试充分性及成本和时间约束，同时注意到一些经常容易出错的地方

常用的黑盒软件测试方法

- 等价类划分法
- 边界值分析法
- 判定表
- 错误推断法

等价类划分

- 将程序的输入划分为一组等价类，针对同一个等价类中任何一个输入数据的测试都等同于针对该等价类中其他输入数据的测试
- 关键在于确定输入空间中的等价类划分，一般可以按照有效等价类和无效等价类来分
  - ✓ 例如，软件界面或方法的规格说明中明确一个表示年龄的整型输入参数取值范围为1到120，那么在此范围内的整数都是合法输入，而其他输入都是非法输入（例如0或负数）
  - ✓ 有效等价类是被测软件的合法输入数据集合：验证软件是否能够实现预定义的功能以及达到期望的性能
  - ✓ 无效等价类则是被测软件的非法输入数据集合：检验软件是否能够对非法输入进行判断和适当的处理，从而测试软件的健壮性和容错性

等价类划分的基本原则-1

- 如果输入条件规定了输入数据的取值范围
  - ✓ 可以确定一个有效等价类和两个无效等价类
  - ✓ 例如，如果输入参数X的取值范围是1到20，那么有效等价类为[1，20]的区间，两个无效等价类为X<1和X>20的区间
- 如果输入条件规定了输入数据值的集合，或者是规定了“必须如何”的条件
  - ✓ 可以确定一个有效等价类和一个无效等价类
  - ✓ 例如，如果两个输入参数X和Y的取值条件为X<Y，那么有效等价类为X<Y，无效等价类为X≥Y
- 如果规定了输入数据的一组可枚举的值，并且软件对每一个输入值都进行不同的处理
  - ✓ 可以确定n个有效等价类（假定有n个值）和一个无效等价类
  - ✓ 例如，如果输入X来自于一个集合{1,2,3}，那么有效等价类是X=1、X=2、X=3这三个，而无效等价类是X∉{1,2,3}（例如X=4）

等价类划分的基本原则-2

- 如果规定了某个输入数据必须满足的规则
  - ✓ 可以确定一个有效等价类（符合规则）和若干个无效等价类（从不同角度违反规则）
  - ✓ 例如，一个系统的合法用户名由字母或数字组成且不包含特殊字符，同时必须由字母开头，那么针对该用户名的一个有效等价类是“符合规则的用户名”，无效等价类包括“包含特殊字符的用户名”、“不以字母开头的用户名”等
- 如果确定一个已知等价类中不同的取值在软件内部会按照不同的方式进行处理
  - ✓ 可以针对该等价类进一步进行划分，从而形成更小的等价类
  - ✓ 例如，对于一个日期类型的输入数据，如果确定软件对于合法日期中的节假日、周末和平时三种情况会进行不同的处理，那么需要进一步将合法日期细分为这三种更小的等价类

从等价类到测试用例

- 建立等价类表：其中每一行代表一个输入条件（例如方法的一个参数）以及针对该条件的有效等价类和无效等价类
- 测试用例设计
  - ✓ 寻找一个能尽可能多地覆盖尚未被覆盖的有效等价类的测试用例，重复该步骤直至所有的有效等价类都被覆盖为止
  - ✓ 寻找一个只覆盖一个尚未被覆盖的无效等价类的测试用例，重复该步骤直至所有的无效等价类都被覆盖为止
- 注意，一个测试用例不应覆盖多个无效等价类，否则当出现测试用例执行失败的情况时无法确定是由于哪个无效等价类所引发的

基于等价类的货品登记程序测试用例设计

输入货品信息，最后给出货品存放指示

- ✓ 编号必须为英文字母与数字的组合，由字母开头，包含6个字符，且不能有特殊字符
- ✓ 货品的登记数量在10到500之间（包含10和500）
- ✓ 货品的类型是设备、零件、耗材中的一种
- ✓ 货品的尺寸是大型、中型、小型中的一种，大型货品存放在室外堆场，中型货品存放在专用仓库，小型货品存放在室内货架
- ✓ 违反以上要求的登记信息被视为无效输入

输入数据	有效等价类	无效等价类
货品编号	（1）符合规则的编号	（7）编号长度不为6 （8）编号有特殊字符 （9）编号不以字母开头
登记数量	（2）10≤数量≤500	（10）数量＜10 （11）数量＞500
货品类型	（3）{设备，零件、耗材}	（12）非设备、零件、耗材中的一种
货品尺寸	（4）大型 （5）中型 （6）小型	（13）非大型、中型、小型中的一种

输入数据	预期输出	覆盖的等价类
A=EQ0101， B=30， C=设备， D=大型	合法登记信息，存放地为室外堆场	（1）（2） （3）（4）
A=CM0202， B=100， C=零件， D=中型	合法登记信息，存放地为专用仓库	（1）（2） （3）（5）
A=MT0303， B=400， C=耗材， D=小型	合法登记信息，存放地为室内货架	（1）（2） （3）（6）
A=EQ01023， B=30， C=设备， D=大型	非法登记信息	（7）
A=EQ0102#， B=30， C=设备， D=大型	非法登记信息	（8）
A=0102EQ， B=30， C=设备， D=大型	非法登记信息	（9）
A=EQ0101， B=0， C=设备， D=大型	非法登记信息	（10）
A=EQ0101， B=600， C=设备， D=大型	非法登记信息	（11）
A=EQ0101， B=30， C=装置， D=大型	非法登记信息	（12）
A=MT0202， B=100， C=耗材， D=中小型	非法登记信息	（13）

边界值分析

- 大量历史经验表明，开发人员经常会在软件的输入或输出数据的边界附近犯错
- 可作为等价类划分法的一种补充，即在等价类划分基础上围绕它们的边界设计测试用例
- 所选择的测试数据一般位于输入的边界条件或临界值附近，可参考以下原则
  - ✓ 如果输入条件规定了一个取值范围，那么选择这个范围的上界和下界以及刚刚超出上界和下界的取值作为测试输入
  - ✓ 如果输入条件规定了传入值的个数，那么选择最大个数、最小个数、比最大个数多1个、比最小个数少1个的传入值作为测试输入
- 对于参数取值范围明确的测试很有效，但不能很好测试布尔值、逻辑变量及不同输入的组合

货品登记程序登记数量的边界值测试用例

针对货品登记程序示例

下表列出了与登记数量相关的边界值测试用例

输入数据	预期输出
A=MT0303， B=10， C=耗材， D=小型	合法登记信息，存放地为室内货架
A=MT0303， B=9， C=耗材， D=小型	非法登记信息
A=MT0303， B=500， C=耗材， D=小型	合法登记信息，存放地为室内货架
A=MT0303， B=501， C=耗材， D=小型	非法登记信息

判定表

- 被测软件包含的多个输入参数的取值存在多种有意义的组合，其中的一些组合可能会导致运行出错
- 需要使用一种基于参数组合的测试技术用于发现导致这种错误的软件缺陷
- 一个判定表由条件和动作两部分组成，表明在每种条件下应该采取的动作（预期的输出结果和行为），而相应的测试要覆盖判定表列出的所有可能的参数取值组合

判定表方法的基本概念

- 条件桩：问题的所有判断条件，即针对待测软件的参数且对问题处理有影响的所有条件
- 动作桩：针对问题可能采取的所有操作，即待测软件的所有预期输出或可能执行的行为
- 条件项：针对所有条件桩的具体取值组合，其中每个条件桩可以取true或false
- 动作项：针对每一个条件项应该采取的动作桩的组合
- 规则：条件项和动作项的每一个组合形成一条规则，即判定表中贯穿条件项和动作项的一列，每条规则可以对应产生一个测试用例

商品配送程序的判定表

需求规格说明：当客户采购特定商品时，如果该商品不在可经营范围内，则向客户发送短信通知。如果商品在经营范围内并且此商品可发售，则告知发运部门发送此商品。当货物送达时，如果客户以往的付款历史情况正常，则允许客户货到后二周内转账，否则（客户付款历史存在有不良记录，例如拖欠付款）货到后要求客户立即付款。如果该商品无法发售，则通知采购部门重新进货，并告知客户。如果客户的付款历史情况正常，则采用电话通知的形式，否则采用短信通知的形式。

条件桩	条件项							
	R1	R2	R3	R4	R5	R6	R7	R8
此商品在可经营范围内	false	false	false	false	true	true	true	true
此商品可发售	true	true	false	false	true	true	false	false
客户付款历史情况正常	true	false	true	false	true	false	true	false
动作桩	动作项							
发送商品					√	√		
货到后允许客户转账					√			
货到客户必须立即付款						√		
重新进货							√	√
电话通知							√	
短信通知	√	√	√	√				√

错误推断

- 又称为探索性测试方法，是指测试人员根据经验、知识和直觉来推测程序中可能存在的各种错误，从而开展有针对性测试
- 主要依赖于测试人员的直觉和经验
- 优缺点
  - ✓ 优点：测试者能够快速且容易地切入，并能够体会到程序易用与否
  - ✓ 缺点：难以知道测试的覆盖率，可能遗漏大量未知的软件部分，并且这种测试行为带有主观性且难以复制
- 一般作为辅助手段，在没有其他方法可用的情况下，用于补充一些额外的测试用例

白盒软件测试方法

- 也被称为结构测试或逻辑驱动的测试
- 意指测试人员将被测软件看作一个透明的白盒，能够基于软件内部的代码实现和逻辑结构进行针对性的测试用例设计
  - ✓ 方法级单元测试：可以利用方法内的程序执行路径来设计针对性的测试用例，从而覆盖不同执行路径
  - ✓ 模块间集成测试：可以利用模块间的交互结构以及交互行为路径来设计针对性的测试用例，从而覆盖不同行为路径
- 经常被用于单元测试，特别是核心关键模块
- 接下来主要介绍方法和函数级白盒测试，其他级别的白盒测试方法可采取类似的思想来实现

白盒测试覆盖目标：从理想到现实

- 方法和函数的内部实现逻辑可以用程序流程图来描述，在此基础上考虑测试用例设计
  - ✓ 理想情况下，白盒测试可以以程序流程图为依据，设计出一组测试用例覆盖所有可能的执行路径，从而确保所有可能出错的情况都被暴露出来
  - ✓ 然而，稍微复杂一点的程序可能就拥有非常大量的执行路径，从而使得完全的路径覆盖变得不现实：一个具有n个判定条件的程序理论上有2^n条执行路径；如果这段程序处于一个循环之中且最大可能循环次数是m，那么执行路径的数量级就会上升到$2^{m\times n}$
- 现实的测试目标
  - ✓ 完全的路径覆盖一般都是不现实的
  - ✓ 一般会按照某种可接受的准则产生测试用例，如语句覆盖、分支覆盖、条件覆盖、分支-条件覆盖、条件组合覆盖等

白盒测试分析案例

商场促销积分计算：若购物满200元且用户拥有VIP卡，则获取本单10%的积分；若购物满400元或者购物品种大于10件，则无论用户是否拥有VIP卡都可以另外获赠5个积分

```
    //a是本单金额;b是VIP卡标志(0代表普通卡，1代表VIP卡)；c是购物品种数量
1       public int getPoints(double a, int b, int c) {
2           int point = 0;
3           if (a>=200 && b==1) {
4               point = (int) (a * 0.1);
5           }
6           if (a>=400 || c>10) {
7               point += 5;
8           }
9           return point;
10      }
```

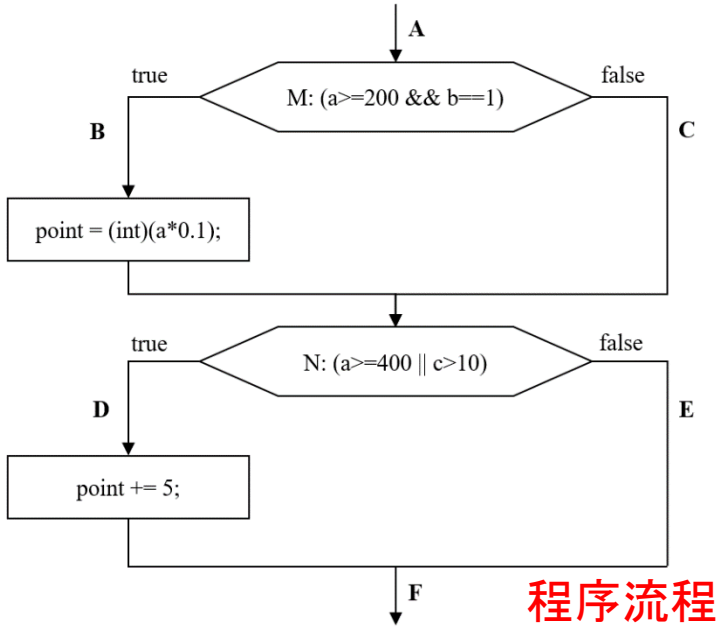
4条执行路径

P1：(ABDF)，当M=true且N=true

P2：(ABEF)，当M=true且N=false

P3：(ACDF)，当M=false且N=true

P4：(ACEF)，当M=false且N=false

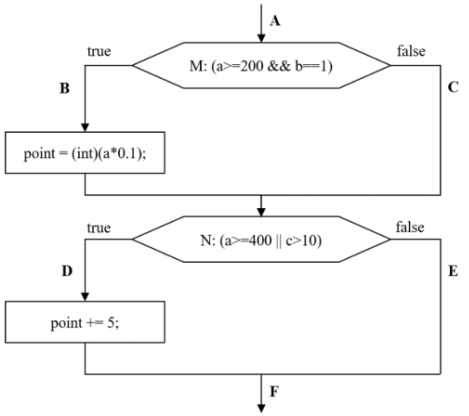


程序流程图

语句覆盖

- 要求所设计的测试用例能够使得被测程序中的每条可执行语句都能被执行至少一次
- 很低的覆盖度准则，很多出错情况都无法捕捉
  - ✓ 例如，如果代码示例中第3行的“&&”被误写为“||”，那么当前测试用例虽然覆盖所有语句但无法发现这一缺陷
  - ✓ 这是由于语句覆盖未考虑判定条件中的各种逻辑组合

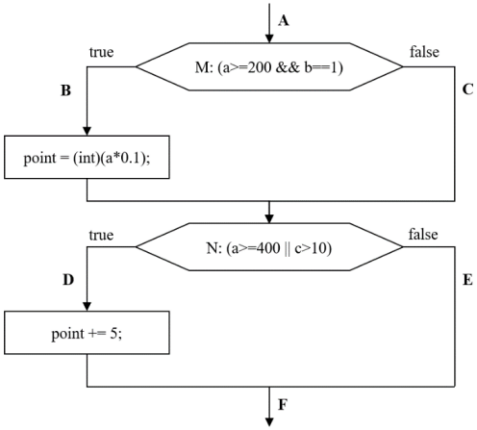
输入数据	预期输出	判定结果	通过路径
a=350, b=1, c=12	40	M=true, N=true	P1



分支覆盖

- 也被称为判定覆盖，要求所设计的一组测试用例能够使得程序中的每个判定的true分支和false分支都能被执行至少一次
- 对于很多出错情况仍然无法捕捉到
  - ✓ 例如，如果代码示例中第6行的条件“c>10”被误写为“c>8”，那么以上测试用例虽然能实现分支覆盖但不能发现这一缺陷

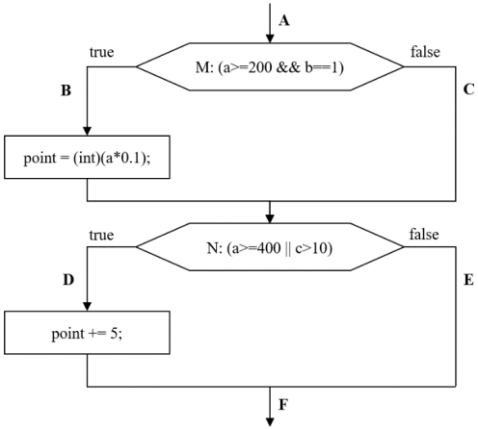
输入数据	预期输出	判定结果	通过路径
a=350, b=1, c=12	40	M=true, N=true	P1
a=150, b=0, c=7	0	M=false, N=false	P4



条件覆盖

- 要求所设计的一组测试用例能够使得程序中每个判定中的每个原子条件的可能取值至少满足一次
- 这两个测试用例的输入与输出不同，但是都经过同一个路径，因此不满足分支覆盖准则甚至还不满足语句覆盖，这也说明了条件覆盖本身也存在不足之处

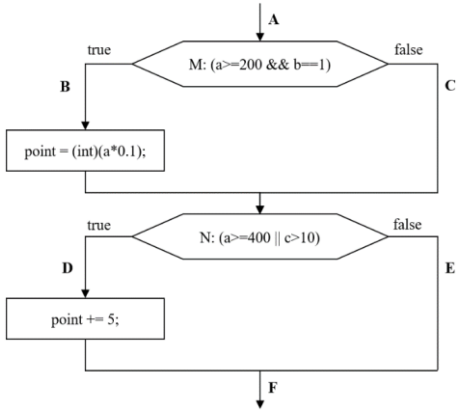
输入数据	预期输出	原子条件取值结果	判定结果	通过路径
a=450, b=0, c=7	5	a≥200, b≠1, a≥400, c≤10	M=false, N=true	P3
a=150, b=1, c=12	5	a<200, b=1, a<400, c>10	M=false, N=true	P3



分支-条件覆盖

- 要求所设计的一组测试用例能够使得程序中的每个判定的true分支和false分支都能被执行至少一次，且每个判定中的每个原子条件的可能取值至少被满足一次
- 是分支覆盖与条件覆盖的结合，但是仍有不足
  - ✓ 如果将代码示例中第6行的“||”误写为“&&”条件，这两条测试用例也能够获得相同的测试结果，这是由于判定表达式中的多个“与”、“或”条件存在相同的组合结果

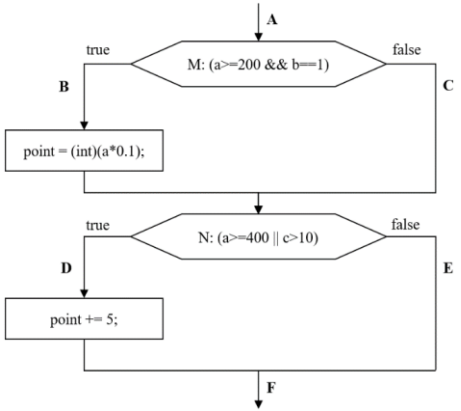
输入数据	预期输出	原则条件取值结果	判定结果	通过路径
a=450, b=1, c=12	50	a≥200, b=1, a≥400, c>10	M=true, N=true	P1
a=150, b=0, c=7	0	a<200, b≠1, a<400, c≤10	M=false, N=false	P4



条件组合覆盖

- 要求所设计的一组测试用例能够使得程序中每个判定中的所有条件组合至少被满足一次
- 这意味着判定中所有原子条件及整个判定的取真和取假都能够被满足
- 是一个很高的覆盖度准则，但并不能使所有程序路径都被执行

输入数据	预期输出	条件组合	判定结果	通过路径
a=450, b=1, c=12	50	① ⑤	M=true, N=true	P1
a=450, b=0, c=7	5	② ⑥	M=false, N=true	P3
a=150, b=1, c=12	5	③ ⑦	M=false, N=true	P3
a=150, b=0, c=7	0	④ ⑧	M=false, N=false	P4

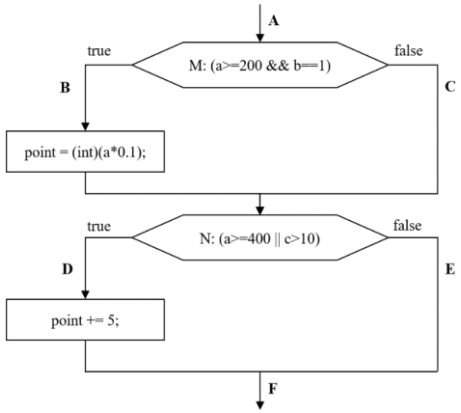


上面四条测试用例满足条件组合覆盖要求，但未能覆盖路径P2（ABEF ）

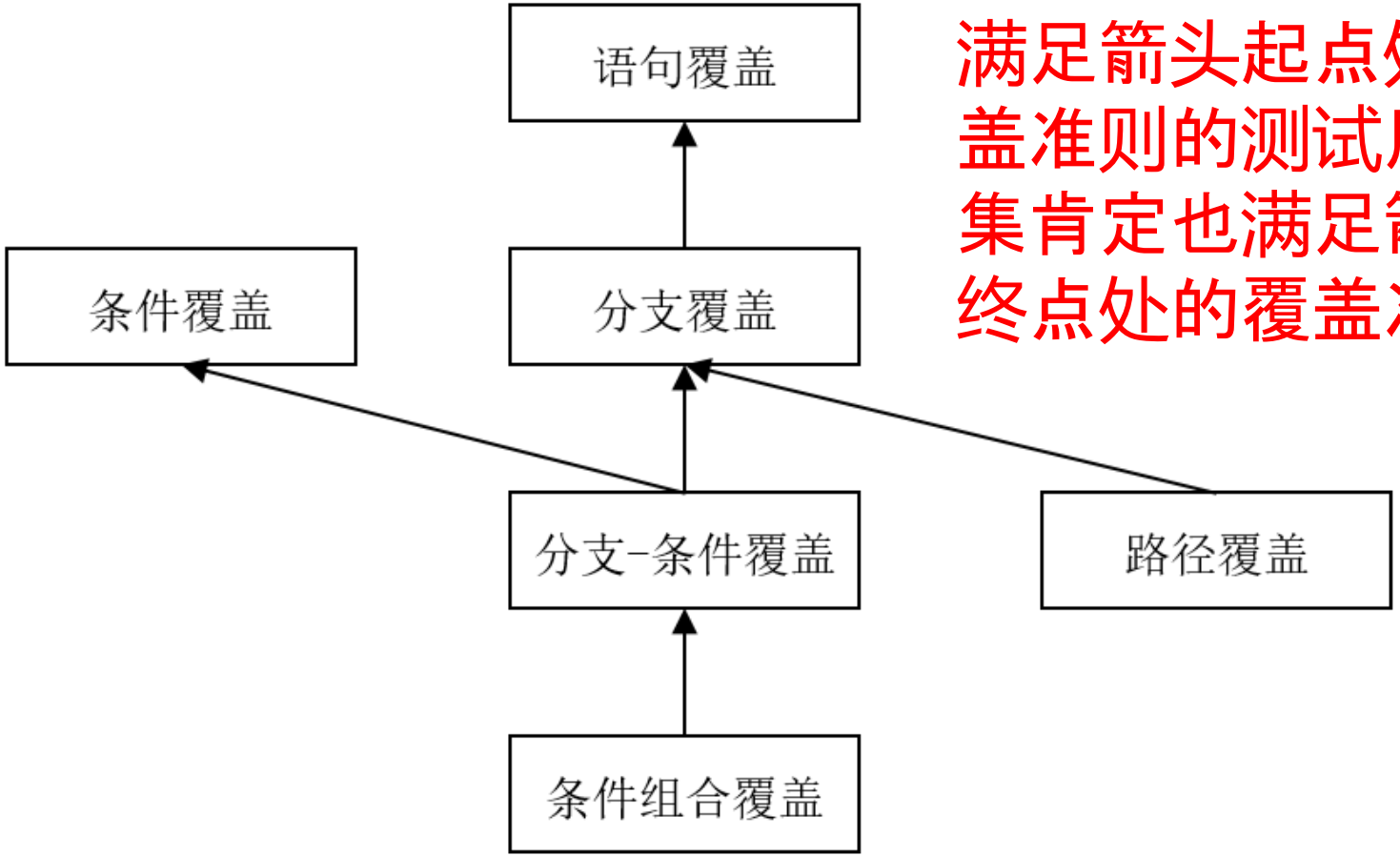
路径覆盖

- 要求所设计的一组测试用例能够使得程序中所有可能的执行路径都被至少执行一次
- 虽然覆盖所有路径，但却未能覆盖所有的条件组合
- 可以将多种白盒测试方法综合运用，从而对全部路径、条件组合、分支都进行覆盖

输入数据	预期输出	条件组合	判定结果	通过路径
a=450, b=1, c=12	50	① ⑤	M=true, N=true	P1
a=300, b=1, c=7	30	① ⑧	M=true, N=false	P2
a=150, b=1, c=12	5	③ ⑦	M=false, N=true	P3
a=150, b=0, c=7	0	④ ⑧	M=false, N=false	P4



各种覆盖准则之间的包含关系



满足箭头起点处覆盖准则的测试用例集肯定也满足箭头终点处的覆盖准则

课堂讨论：黑盒测试和白盒测试

课堂讨论：黑盒测试和白盒测试在发现软件缺陷的能力上有什么互补性？如何将二者有效结合从而提高软件测试质量？

系统测试技术与工具

- 系统测试将整个软件系统作为一个整体并考虑具体的系统运行环境进行测试
  - ✓ 考虑软件运行所处系统环境（包括硬件、网络、物理环境、用户等）
  - ✓ 针对特定质量属性开展测试，如性能测试、兼容性测试、安全性测试等
- 不同类型的系统测试都有一些相应的自动化测试技术以及测试工具支持
  - ✓ 功能测试
  - ✓ 性能测试
  - ✓ 兼容性测试
  - ✓ 易用性测试
  - ✓ 可靠性测试
  - ✓ 信息安全测试

功能测试技术与工具-1

- 线性脚本：简单的录制回放
  - ✓ 使用录制工具并站在最终用户的角度录制被测软件的使用过程，一般是针对交互界面上元素的操作序列
  - ✓ 通过回放执行所录制的使用过程来进行自动化测试
  - ✓ 缺点：测试用例开发成本较高（需要不断录制不同的软件使用过程）；测试脚本之间可能会存在重复的操作序列，维护成本较高
- 结构化脚本：在线性脚本基础上加入控制结构
  - ✓ 在线性脚本基础上加入结构化控制，例如“if-else”，“switch”，“for”，“while”等条件和循环语句
  - ✓ 一条测试脚本可以包含多个执行流（经过不同的条件或循环分支）
  - ✓ 缺点：脚本间仍然是相互独立的，即仍然存在脚本间重复的问题

功能测试技术与工具-2

- 模块化脚本：允许测试脚本的模块化调用
  - ✓ 将被测软件中公共功能的测试脚本封装成模块，其他测试脚本可按需对其进行调用
  - ✓ 实现了测试脚本的标准化和组件化，避免了重复编写相同的操作脚本，提高了测试脚本开发效率、降低了维护成本
- 数据驱动测试：将数据从脚本中分离出来
  - ✓ 将脚本要用到的数据进行单独存储，支持使用不同的数据对同一功能进行测试，进一步提高了脚本的可复用性
  - ✓ 测试脚本包含读取数据存储单元的代码，数据存储可以是脚本内的数组、字典，或外部文件（csv、excel等格式）

功能测试技术与工具-3

- 关键字驱动测试：将“数据”转化为“关键字”，通过关键字的改变引起测试结果的变化
  - ✓ 将软件的操作对象、所执行的操作、满足条件、传输值、检查点断言，甚至是所需要读取的数据文件等都转换为具有可识别语义的关键字词组，例如：使用“login with (user name and password)”代表以特定的用户名和密码进行登录操作
  - ✓ 一个测试脚本就是由这些关键字词组序列组合而成的，测试执行引擎读取脚本并将这些关键字翻译成针对被测软件的具体操作流程，从而驱动被测软件的执行
  - ✓ 这种方法能够显著降低测试脚本编制难度，使得不熟悉低层脚本编写语言的人员（例如领域专家等）也可以使用填写关键字的形式来编制测试脚本

功能测试技术与工具-Selenium

- 最知名的自动化测试框架之一
- 可以直接驱动浏览器来模拟用户的web应用操作（打开浏览器、获取网页内容、点击网页控件等），从而满足不同类型网站的自动化测试需求
- Selenium的构成
  - ✓ Selenium IDE是一个用于创建测试脚本的工具，能够记录用户的操作，并把它们导出到一个可重用的脚本中用于重复执行，一般以插件的形式发布在浏览器中
  - ✓ Selenium-Grid是一种自动化测试辅助工具，通过使用多台机器并行地运行测试来加速测试的执行过程
  - ✓ Selenium WebDriver是Selenium的核心工具，提供了一套编程框架用于创建和执行测试用例

Selenium WebDriver

- 针对各个浏览器进行开发，取代了嵌入到被测Web应用中的JavaScript，即每个浏览器都有单独的驱动程序，可以直接调用浏览器和本地方法，这种与浏览器紧密集成的方式避免了浏览器对JavaScript的安全限制
- 可以通过调用操作系统的本地方法与浏览器进行交互，例如调用操作系统的本地方法模拟用户在界面上的数据输入操作
- 支持通过大多数常用编程语言（Java、C# 、PHP、Ruby、JavaScript、Pearl、Python等）编写的测试脚本实现与浏览器的交互和数据传输

Selenium WebDriver的测试脚本

一组对界面控件元素的基本操作构成的序列

常用的用于操控浏览器和控件的API方法如下

API方法	作用	示例
get	打开网页	webDriver.get(“http://www.baidu.com”)
findElement	获取元素	webDriver.findElement(By.id(“kw”))
findElements	获取元素集合	webDriver.findElements(By.tagName(“input”))
sendKeys	输入内容	webElement.sendKeys(“software testing”)
clear	清空内容	webElement.clear()
click	执行点击事件	webElement.click()
getText	获取元素内容	webElement.getText()
navigate().to	打开网页	webDriver.navigate().to(“http://www.baidu.com”)
navigate().back	回退上一个页面	webDriver.navigate().back()
navigate().forward	前进到下一页面	webDriver.navigate().forward()
navigate().refresh	刷新当前页面	webDriver.navigate().refresh()
close	关闭浏览器	webDriver.close()

Selenium WebDriver中的控件元素定位

- 为了编写测试脚本，最关键的是要正确地定位到web应用界面上的控件元素
- Selenium WebDriver采用8种策略实现定位
  - ✓ 使用ID进行定位
  - ✓ 使用name进行定位
  - ✓ 使用xpath进行定位
  - ✓ 使用tagName进行定位
  - ✓ 使用className进行定位
  - ✓ 使用cssSelector进行定位
  - ✓ 使用linkText进行定位
  - ✓ 使用partialLinkText进行定位

软件系统的性能评价

- 性能评价的几个方面
  - ✓ 响应时间：软件系统从请求发出开始到客户端接收到最后一个字节数据所消耗的时间，是用户能够直接感受到的性能指标之一
  - ✓ 并发用户数：在同一时刻在线并同时使用软件系统的用户的数量
  - ✓ 吞吐量：单位时间内软件系统所能处理的客户请求的数量
  - ✓ 系统占用资源：软件系统运行时的服务器CPU使用率、内存使用率、硬盘的I/O数据、数据库服务器缓存命中率、网络带宽数据等
- 根据测试目标的不同，软件系统性能效率的测试主要分为性能测试、负载测试和压力测试

性能测试

- 通常在功能测试基本完成，并且软件已经趋于稳定（改动越来越少）的情况下进行
- 目的是在真实环境下检测软件系统性能
  - ✓ 模拟多种正常、峰值及异常负载条件来对系统的各项性能指标进行测试
  - ✓ 评估系统性能以及服务等级的满足情况
  - ✓ 分析系统瓶颈并优化系统
- 在执行性能测试时，应当使测试环境与真实执行环境尽量保持一致，同时确保软件系统单独执行（即避免与其他软件一起运行），从而保证测试有效性

负载测试

- 在系统负荷（例如并行用户数）不断增加的情况下对一个软件系统行为的测量，以探索并确定软件系统所能够支持的负载量级
- 当不清楚软件系统所能支持的负载时，可使用负载测试方法（例如每隔1秒增加5个并发用户数）来找到该系统的性能极限（即容量）
- 当确定了系统的容量后，如果该容量不满足用户要求时，就需要寻找解决方案以扩大容量，否则就要在软件产品说明书上明确标识该容量的限制

压力测试

- 对一个软件系统在其需求所定义的边界内或超过边界的情况下的性能指标进行评估
- 试图发掘出系统在负载临界条件下（在标定的容量限制之上的一定范围内，例如在超过标定的最大并发数量10%的情况）的功能或性能隐患，从而评估在异常情况下的软件系统的健壮性和可生存性
- 可在为系统加载反常数量（例如长时间的峰值）、频率或资源负载的情况下执行可重复的测试，以检查程序对异常情况的抵抗能力

性能测试技术与工具

- 性能测试、负载测试和压力测试均采用不断加载系统负载的方式，例如逐渐增加模拟用户的数量或其它对系统资源的使用负荷
- 相应的测试工具模拟用户的并发行为
  - ✓ Flat：一次性地加载所有的用户，然后在预定的时间段内保持这些用户的持续运行
  - ✓ Ramp-up：用户是交错上升的，即工具每隔几秒增加一些新的用户
- 性能测试工具依赖于测试人员编制的测试脚本，即所模拟出的并发用户按照脚本对系统进行访问或使用

性能测试工具JMeter

- 最知名的性能测试工具之一
- Apache基金会旗下的一款基于Java的开源软件，主要针对服务器或网络，通过模拟并发负载来测试并分析被测对象的性能
- 可以支持多种类型的应用、服务或协议，包括HTTP/HTTPS、SOAP、REST、FTP、TCP、LDAP等
- 允许测试人员通过可视化界面配置性能测试的执行策略

JMeter的配置选项

配置项	配置内容
添加线程组(Thread Group)	线程组可被理解为独立的测试任务。测试人员可以指定测试任务的执行次数、是否自动停止等配置内容
配置线程组	对线程组内的三个关键属性进行配置：线程个数、启动全部线程所花费的时间、线程循环执行次数
添加取样器(Sampler)	取样器用来模拟用户操作，即模拟用户向被测对象发送请求并接收被测对象的响应数据，例如“HTTP请求”取样器
添加配置元件(Config Element)	常用的配置元件（用户定义的变量和CSV数据文件）可用于实现测试用例的参数化，使得所模拟出的用户请求能够附带不同的参数，从而提高测试的覆盖度
添加监视器(Listener)	配置监视器来观察样本发送的消息状态，比较常用的是View Results Tree和Summary Report
添加断言元件(Assertion)	断言元件来判断响应数据是否符合预期

兼容性测试

- 面向以下三个兼容性维度
  - ✓ 系统内部兼容：系统内部各部件之间的兼容性，包括软件和软件、软件和硬件、硬件和硬件之间的兼容性
  - ✓ 系统间兼容：系统与其他系统存在接口互连、功能交互等情况下的配合能力
  - ✓ 系统自身兼容：系统的新老版本间需要保证的功能、操作体验等方面的一致性，包括前向兼容和后向兼容
- 首先需要识别出一个对象，即现场可替换单元（LRU：Line Replaceable Unit），然后检验该对象与周边环境的配合、适应关系及程度

一个安卓APP的兼容性测试示例

步骤	分析结果
应用场景分析	**应用场景**：验证APP可以在不同手机上正常安装和卸载，功能正常使用，并且不影响其他APP的使用 **LRU识别**：被测对象是安卓手机上的APP，LRU操作是APP安装、卸载和运行 **内部兼容接口分析**： 1）软件间兼容：APP安装后与其他APP存在共享相同系统及资源的情况，所以需要验证兼容性。另外，因为APP可能运行在不同的安卓版本上，所以需要验证与安卓系统的兼容 2）软硬件兼容：APP可能安装在不同的手机上，手机硬件配置存在差异，需要验证APP与不同类型手机间的兼容 3）硬件间兼容：因为场景只涉及软件安装和卸载，所以不涉及硬件间兼容 **前后向兼容分析**：因为没有版本变更操作，所以不涉及系统自身的前后向兼容 **周边兼容场景分析**：因为APP无外部接口，所以不涉及系统间兼容
LRU分析	影响对象：1）APP自身；2）其他APP；3）安卓系统；4）不同硬件类型手机。
影响指标分析	1 APP自身：基本功能正常，安装和卸载的启动时间满足要求； 2 其他APP：基本功能正常，加载和卸载正常； 3 安卓系统：基本功能正常，可以正常使用，并且没有出现使用卡顿情况； 4 不同硬件类型手机：指标同前3条。

可根据指标采用自动化或人工的方式对测试场景中所涉及的内容进行验证

易用性（Usability）测试

- 易用性测试需要从用户的角度来验证软件系统是否易于理解、易于学习、易于操作
- 参与易用性测试的人员可以包含软件企业中的各类角色，从不同的角度去验证系统的易用性
- 可以将软件系统的十大交互原则作为易用性测试的评价准则
- 测试者应当熟悉以上十原则，具备基础的易用性测试能力，从而在使用软件系统的过程中发现其中的问题

软件系统的十大交互原则

原则	解释
状态可见	针对用户的操作，系统能及时反馈操作是否生效
环境贴切	用户常用操作和大部分系统设计保持一致，不应出现“反人类”的设计
用户可控	为了避免用户的误操作，系统应支持撤销的功能，并以方便的形式允许用户使用，从而使得用户能够方便地回退到之前的状态
一致性	系统中同一用语、功能、操作保持一致，同样的语言、同样的情景、操作应该出现同样的结果
防错	系统应防止用户的误操作
易取	系统应减少用户记忆负担，把需要记忆的内容放在可见界面上
灵活高效	系统应提供特定能力以使得用户在使用某些功能时更加灵活、操作更加高效
优美简约	系统界面上多余的信息会分散用户对有用或者相关信息的注意力，因此界面应贴合实际场景，突出重点，弱化和剔除无关信息
容错	系统应帮助用户从错误中恢复并将损失降到最低。如果无法自动挽回，则应提供详尽的说明文字和指示方向，而不应该使用代码
人性化帮助	系统应提供帮助性提示，包括一次性提示、常驻提示、帮助文档等

可靠性测试

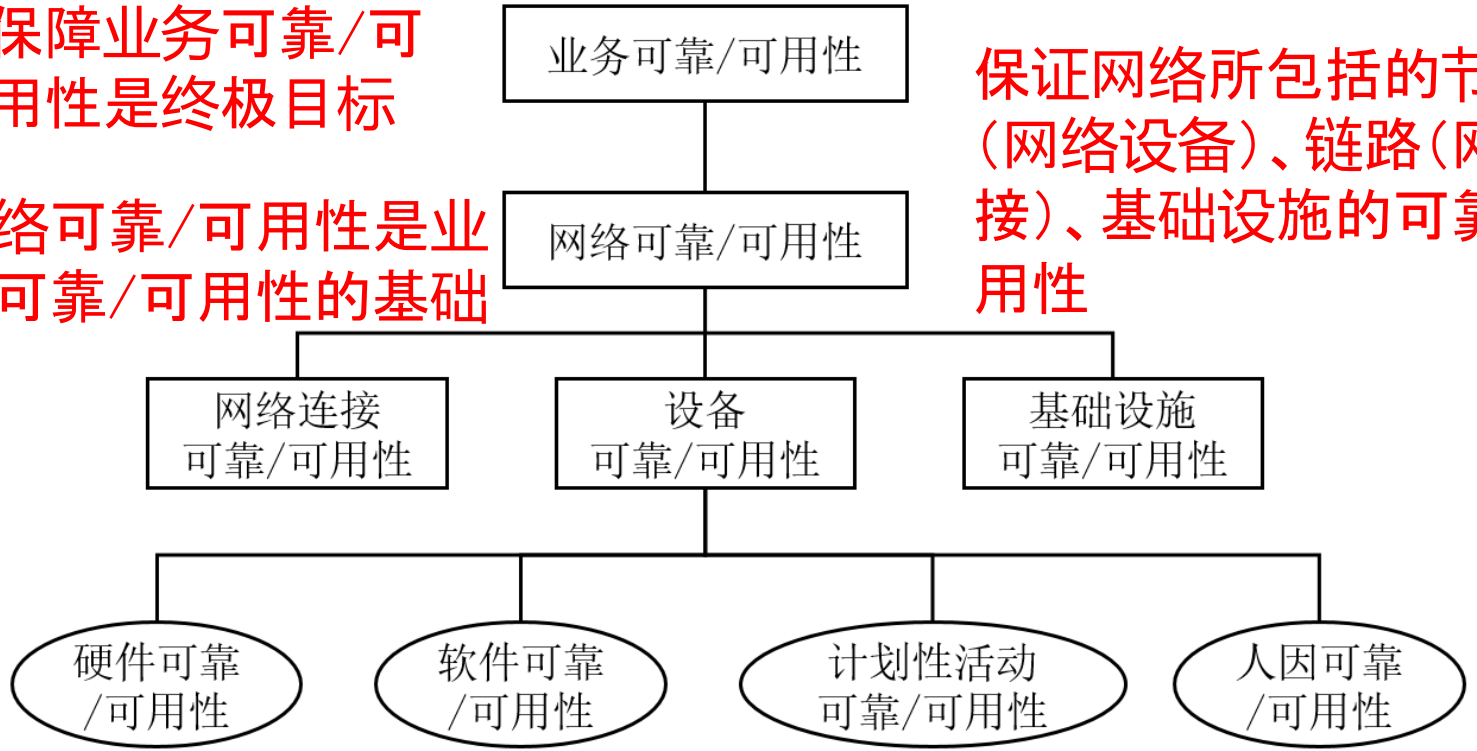
- 广义的可靠性包括可靠性和可用性
  - ✓ 可靠性：软件系统在规定的条件下和规定的时间内完成规定功能的能力，通常使用概率性的度量（可靠度）
  - ✓ 可用性：软件系统在任意随机时刻需要开始执行任务时，都可处于可工作或可使用状态的程度，通常使用概率性的度量（可用度）
- 也可以被理解为可靠/可用性测试
- 可靠/可用性测试的目标是通过增强故障验证的能力来提高产品的可靠性和可用性

电信领域软件系统的可靠性/可用性

保障业务可靠/可用性是终极目标

网络可靠/可用性是业务可靠/可用性的基础

保证网络所包括的节点（网络设备）、链路（网络连接）、基础设施的可靠/可用性



对于设备的可靠/可用性，除了基础的软硬件可靠/可用性外，还必须考虑计划性活动的可靠/可用性和人为因素的可靠/可用性

可靠性和可用性度量

- 可靠性使用MTBF度量
  - ✓ MTBF（Mean Time Between Failure）是平均故障间隔时间，或称为平均无故障工作时间
  - ✓ 具体指相邻两次故障之间的平均工作时长
- 可维修性使用MTTR度量
  - ✓ MTTR（Mean Time To Recover）是平均故障修复时间
  - ✓ 越短的MTTR意味着在规定条件和规定时间内，按规定的程序和方法维修时，系统保持和恢复到规定状态的能力越强
- 可用性计算： MTBF/(MTBF+MTTR)
  - ✓ 可用性不仅取决于可靠性，还取决于可维修性（可恢复性），即出现故障后快速恢复的能力

高可靠与高可用

- 高可靠不一定等于高可用
  - ✓ 若一个系统是高可靠的，即MTBF非常大，表明该系统很少出故障
  - ✓ 若该系统的可维修性很差，出一次故障需要很长时间才能恢复（即MTTR很大），那么该系统仍然不是高可用的（处于可用状态的时间比例小）
  - ✓ 高可用性一般等于高可靠性加上高可维修性
- 最终用户更加关心可用性
- 电信领域中不同类型系统的可用性要求

可用度	9的个数	年停机时间	使用产品或系统
0.999	三个9	500分钟	电脑或服务器
0.9999	四个9	50分钟	企业级设备
0.99999	五个9	5分钟	一般电信级设备
0.999999	六个9	0.5分钟	更高要求电信级设备

可靠/可用性测试的实施手段

- 可靠性测试触发和激活系统中的故障，观察系统能否不发生错误或失效，关注故障的避免和预防
- 可用性测试触发和激活系统中的故障，在系统出错后观察业务功能是否正常，从业务的角度验证系统如何不受或仅受尽量少的影响
- 触发或激活故障的方式
  - ✓ 自然方式：通过长时间运行系统使其出现超常规负荷情况或异常
  - ✓ 故障注入方式：向系统注入在实际应用中可能发生的故障
  - ✓ 常见的故障注入方式包括网络级故障注入、系统级故障注入、资源类故障注入、数据类故障注入、硬件类故障注入

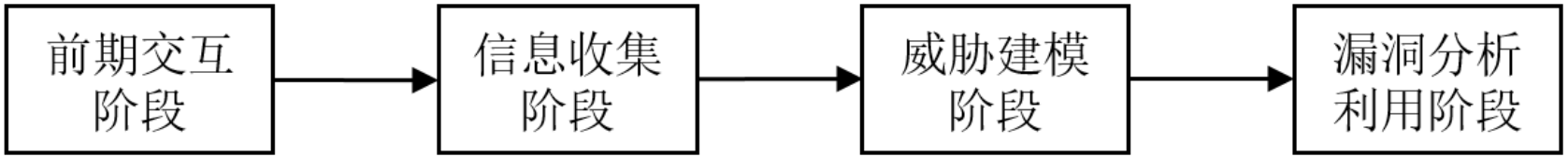
信息安全测试

常见的面向Web应用的信息安全测试的测试内容

类型	测试内容
服务器信息测试	运行账号测试；web服务器端口版本测试；HTTP方法测试
文件目录测试	目录遍历测试；文件归档测试；目录列表测试
认证测试	验证码测试；认证错误测试；找回、修改密码测试
会话管理测试	会话超时测试；会话固定测试；会话标识随机性测试
授权管理测试	横向越权测试；纵向越权测试；跨站伪造请求测试
文件上传下载测试	文件上传测试；文件下载测试
信息泄露测试	数据库账号密码测试；客户端源代码测试；异常处理测试
输入数据测试	SQL注入测试；XML注入测试；LDAP注入测试
跨站脚本攻击测试	反射型测试；存储型跨站测试；DOM型跨站测试
逻辑测试	上下文逻辑测试；算术逻辑测试
Webservice测试	Webservice接口测试；Webservice完整性、机密性测试
HTML5测试	CORS测试；Web客户端存储测试；WebWorker安全测试
FLASH安全配置测试	全局配置文件安全测试；浏览器端安全测试
其他测试	Struts2测试；Web部署管理测试；日志审计测试

Web应用信息安全测试的主要步骤

- 前期交互阶段：组织产品架构师进行业务串讲，确定测试范围，识别具有高危风险的模块
- 信息收集阶段：收集尽可能多的受测产品的材料文档，并在运行环境上使用扫描工具收集运行环境信息
- 威胁建模阶段：针对收集到的产品信息进行威胁建模，进一步细化高风险模块
- 漏洞分析利用阶段：综合分析获取和汇总的信息，找出可以实施渗透攻击的攻击点，并在环境中进行验证



Web应用信息安全测试的技术关注点

- 识别高危模块，从而有策略、有优先级地安排对每个模块信息安全方面的测试
- 掌握常见攻击模式库，以便能快速验证疑似安全问题是否真实存在
- 掌握跟踪调测技术，通过调测技术可以加快对产品业务的理解，特别是加快对复杂代码块的理解
- 了解基础的白盒测试方法，从而更有深度、更大范围地覆盖产品中的逻辑代码

本章小结-1

- 软件测试是一种重要的软件质量保障手段
- 软件工程领域已经形成了系统性的软件测试方法和技术体系，能有效保障软件产品质量
  - ✓ 软件测试与软件开发过程紧密结合，衍生出了V模型、W模型和敏捷测试模型等软件测试过程模型
  - ✓ 大多数系统的测试都遵循由小到大的原则，按照单元测试、集成测试、系统测试、验收测试等不同类型逐层进行
- 指导软件测试实践的方法主要包括黑盒测试方法和白盒测试方法两大类
  - ✓ 黑盒测试将被测软件整体视为一个仅能通过外部接口交互的封闭黑盒，根据被测软件的外部规格说明设计测试用例
  - ✓ 白盒测试将被测软件看作一个透明的白盒，基于软件内部的代码实现和逻辑结构进行针对性的测试用例设计

本章小结-2

- 系统测试将整个软件系统作为一个整体并考虑具体的系统运行环境进行测试
- 不同类型的系统测试都有相应的自动化测试技术以及测试工具支持
  - ✓ 功能测试
  - ✓ 性能测试
  - ✓ 兼容性测试
  - ✓ 易用性测试
  - ✓ 可靠性测试
  - ✓ 信息安全测试

COMP130015.02
软件工程

End

9. 软件测试