1-1软件概念

软件是计算机系统中与硬件相互依存的另一部分，它是包括程序、数据及其相关文档的完整集合。

程序是按事先设计的功能和性能要求执行的指令序列。

数据是使程序能正常操作信息的数据结构。

文档是与程序开发、维护和使用有关的图文材料。

1-2软件的十大特性

形态特性:软件是无形的，不可见的逻辑实体。

智能特性：软件是复杂的智力产品，能够帮助我们解决复杂的计算、分析、判断和决策问题

生产特性：与硬件不同，软件一旦设计开发出来，如果需要提供多个用户，它的复制十分简单，其成本也极为有限。

管理特性：由于上述特性的存在，所以软件过程中的管理显得更为重要，相比传统行业，也更为独特。

环境特性：软件的开发和运行都离不开相关的计算机系统环境，包括支持它的开发和运行的相关硬件和软件。软件对于计算机的系统的环境有着不可摆脱的依赖性。

维护特性：软件投入使用以后需要进行维护，维护体现在升级、优化、功能更新等方面，甚至可以全盘重构。

废弃特性：与硬件不同，软件并不是由于被“用坏”而是被废弃的。

应用特性：软件的应用极为广泛。

1-3软件的分类

系统软件：是负责管理计算机系统中各种独立的硬件，使得它们可以协调工作。

* 服务性程序：如诊断程序、排错程序、练习程序等。
* 语言程序：如汇编程序、编译程序、解释程序。
* 操作系统
* 数据库管理系统

应用软件：是为了某种特定的用途而被开发的软件，它可以是一个特定的程序，如一个图像浏览器。也可以是一组功能联系紧密，可以互相协作的程序的集合。

2-1软件生命周期

基本概念：软件的生命周期，又称为软件的生存周期。它是按开发软件的规模和复杂程度，从时间上把软件开发的整个过程（从计划开发开始到软件报废为止的整个历史阶段）进行分解，形成相对独立的几个阶段。

每个阶段又分解成几个具体的任务，然后按规定顺序依次完成各阶段的任务并规定一套标准的文档作为各个阶段的开发成果，最后生产出高质量的软件。

软件的一生

问题定义----可行性研究----需求分析----概要设计----详细设计----编码和单元测试----综合测试----软件维护

实战介绍一个项目的生命周期

余额宝的诞生

问题定义：普惠金融与互联网金融的融合----可行性研究：T+0是否可以实现，是否合规----需求分析：细分story----概要设计----详细设计----编码和单元测试----综合测试----软件维护

2-2软件开发模型

瀑布模型：

计划----需求分析----设计----编码----测试----运行维护

特点：1.软件开发的各项活动严格按照线性方式进行。

2.当前活动接受上一项活动的工作结果。

3．当前活动的工作结果需要进行验证。

缺点：1.由于开发模型是线性的，增加了开发风险。

2.早期的错误可能要等到开发后期的阶段才能发现。

原型模型：

客户与开发公司紧密联系，开发周期长。开发会受到需求变更的影响。

特点：1.实现客户与系统的交互。

2.进一步细化待开发软件的需求。

3.开发人员可以确定客户的真正需求是什么。

螺旋模型：

制定计划---风险分析---实施工程（需求确认、软件需求、软件产品设计、设计确认与认证、设计确认与认证、详细设计、开发、测试）---客户评估

特点：1.螺旋模型是将瀑布模型与快速原型模型结合起来。

2.强调了其他模型所忽视的风险分析。

3.每一次螺旋包括4个步骤：制定计划、风险分析、实施工程、客户评估。

缺点：1.强调风险分析，但要求许多客户接受并相信这种分析，是不容易的。

敏捷模型：

敏捷开发是一种以人为核心、迭代、循序渐进的开发方法。

特点：1.短周期开发。

2.增量开发。

3.由程序员和测试人员编写的自动化测试来监控开发进度。

4.通过口头沟通、测试和源代码来交流系统的结构和意图。

5.编写代码之前先测试代码，也叫测试先行。

缺点：1.团队组建较难，人员素质要求较高。

2.对测试人员要求完全掌握各种脚本语言编程，能执行单元测试、自动化测试。

2-3软件开发文档

需求分析文档---概要设计文档---详细设计文档---测试设计文档---测试用例---测试报告

2-4阿里系开发模型的变迁史

最早期：边做边改---稳定期：瀑布式---发展期：敏捷---创新期：DEVOPS

2-5项目的一生---BAT测试

项目进程：

编程阶段：单元（白盒）--测试参与——编程完成-开发联调（集中测试）--开发为主——提测-冒烟测试（自动化为主，手工为辅）-测试执行——测试阶段-系统测试（黑盒功能测试为主，自动化/接口测试为辅，根据项目进行性能、安全测试）——验收阶段-验收测试-测试配合用户或需求

2-6软件测试方法

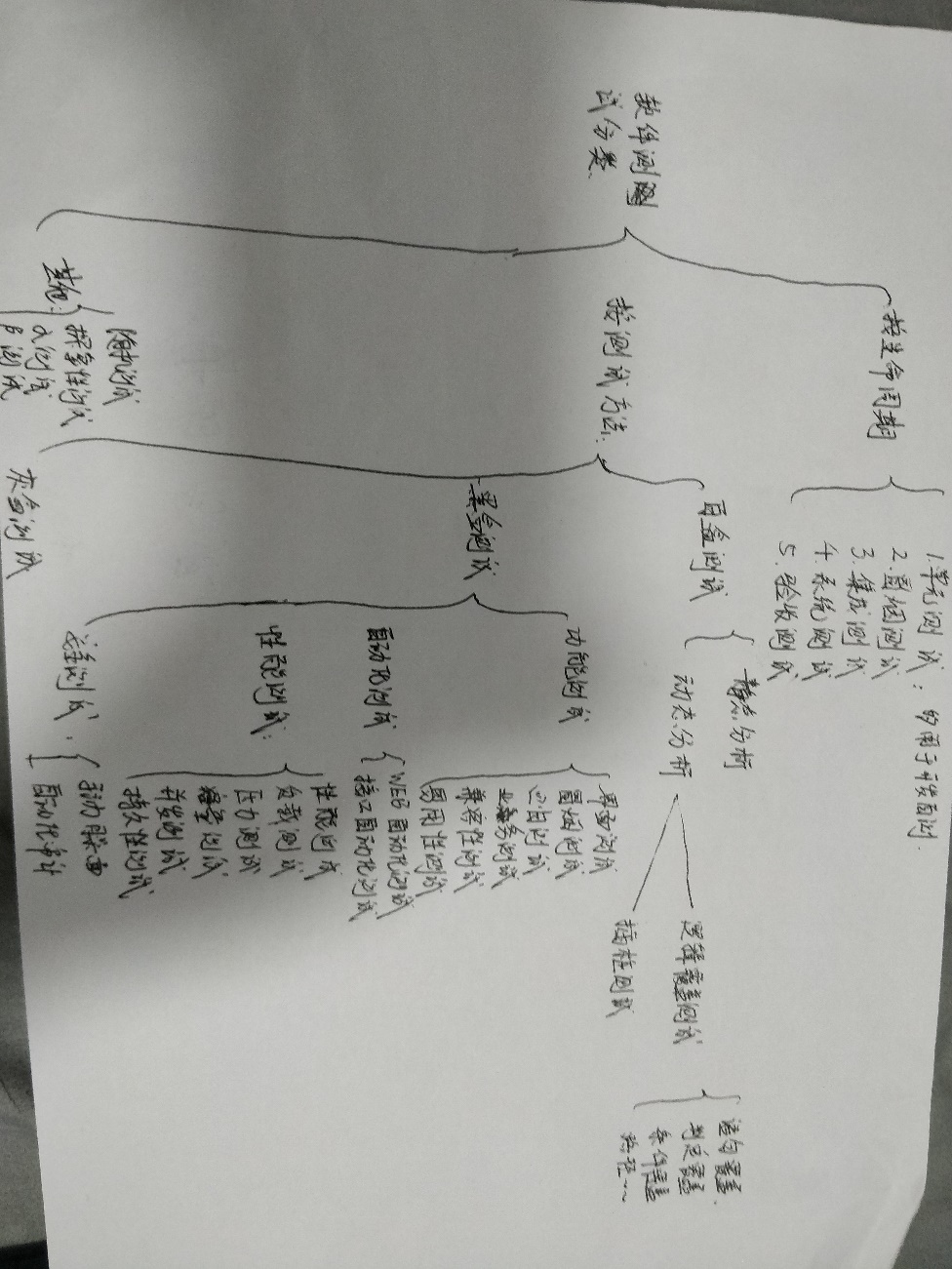
软件测试概念

经典定义：软件测试，在规定的条件下对程序进行操作，以发现程序错误，衡量软件质量，并对其是否满足设计要求进行评估的过程。

标准定义：软件测试是使用人工或自动的手段来运行火测定某个软件系统的过程，其目的在于检验它是否满足规定的需求或弄清预期结果与实际结果之间的差别。

软件测试的目的：在于发现问题，检查系统是否满足需求。

软件测试的分类：



生命周期各测试方法对比：

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 单元测试 | 集成测试 | 冒烟测试 | 系统测试 | 验收测试 |
| 测试阶段 | 编码后 | 单元测试完成后 | 提测后 | 冒烟测试通过后 | 发布前 |
| 测试对象 | 最小模块 | 模块间的接口 | 整个系统 | 整个系统 | 整个系统 |
| 测试人员 | 白盒测试或开发 | 白盒测试或开发 | 黑盒测试 | 黑盒测试 | 最终用户或需求方 |
| 测试依据 | 代码、注释、详细设计文档 | 单元测试模块、概要设计文档 | 冒烟测试用例 | 需求说明文档、测试方案、测试用例 | 用户需求、验收标准 |
| 测试方法 | 白盒测试 | 黑盒与白盒结合 | 黑盒测试（手工或与自动化结合） | 黑盒测试 | 黑盒测试 |

2-7软件测试的常用术语

C/S：C指的是客户端（Client）,S指的是服务器端（Server），这种软件是基于局域网或互联网的，需要一台服务器来安装服务器端软件，每台客户都需要安装客户端软件。如QQ和各种网络游戏就属于C/S结构的软件。

B/S：B指的是浏览器（Browser），S指的是服务器（Server），这种软件同样是基于局域网或互联网的，它与C/S结构软件的区别在于，不需要安装客户端（Client）,它只需要有浏览器，就可以直接使用。如搜狐，新浪等门户网站及163邮箱都属于B/S结构的软件，B/S结构软件是现在软件的主流，与C/S结构软件相比，便于升级和维护，是测试的重点。

缺陷（Bug/Defect）:软件的Bug指的是软件中（包括程序和文档）不符合用户需求的问题。

测试环境：软件测试环境就是软件运行平台，包括软件、硬件和网络的集合。用一个等式来表示：测试环境=软件+硬件+网络

测试用例（Test Case）:在测试执行之前设计的一套详细的测试方案，包括测试环境、测试步骤、测试数据和预期效果。

用一个等式来表示：测试用例=输入+输出+测试环境

其中，“输入”包括测试数据和操作步骤

“输出”指的是期望结果

“测试环境”指的是系统环境设置

冒烟测试（Smoke Testing）:在对一个新版本进行系统大规模的测试之前，先验证一个软件的基本功能能否实现，是否具备可测性。

α测试：验收测试的一种，指的是由用户、测试人员、开发人员等共同参与的内部测试。

β测试：验收测试的一种，指的是内测后的公测，即完全交给最终用户测试。

2-8软件测试常见模型

V模型

V模型是我们熟知的瀑布模型的一种改进，瀑布模型将软件生命周期划分为计划、分析、设计、编码、测试和维护六个阶段，由于早期的错误可能要等到开发后期的测试阶段才能发现，所以可能带来严重的后果。

V模型就是在这点上改进了瀑布模型，在软件开发的生存期，开发活动和测试活动几乎同时开始，这两个并行的动态的过程就会极大的减少bug和error出现的几率。

V模型

验收测试

用户需求

系统测试

需求分析

集成测试

概要设计

详细设计

单元测试

编码

W模型

一些高性能高风险的系统，互联网软件，或一个系统难以被具体模块化的时候，就比较难做成V模型所需的各种构件，需要更强调迭代的开发模型或者敏捷开发模型。

W模型从V模型演化过来，实际上开发是V，测试是并行的V；相对于V模型W模型增加了软件各开发阶段中应同步进行的验证和确认活动，W明确表示出了测试与开发的并行关系。测试与开发是同步进行的，有利于尽早地全面的发现问题。

W模型

验收测试

系统安装

需求测试

需求分析

系统测试

集成测试

单元测试

系统构建

编码实现

模块集成

详细设计测试

详细设计

概要设计测试

概要设计

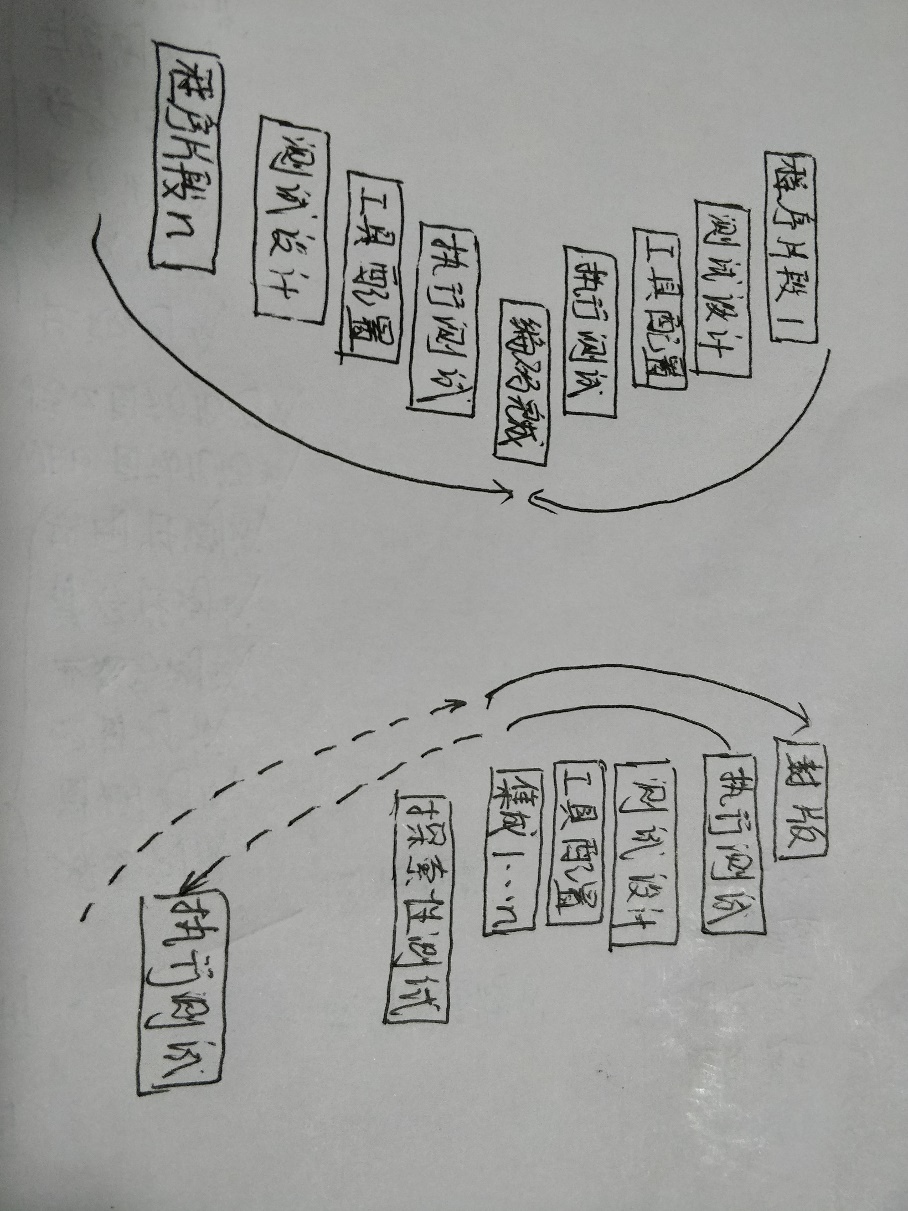
其他模型—H模型

真正的测试级别之间不存在严格的次序关系，各阶段间可以反复触发、迭代

增量。

为了解决V模型和W模型存在的问题，有专家提出了H模型。它将测试活动完全独立出来，形成一个完全独立的流程，将测试准备活动和测试执行活动清晰的体现出来。

其他模型—X模型



2-9软件测试覆盖率

测试覆盖率：

覆盖率是用来度量测试完整性的一个手段，同时也是测试技术有效性的一个度

量。

覆盖率=（至少被执行一次的item数）/item的总数

特点：1.通过覆盖率数据，可以检测我们的测试是否充分

2.分析出测试的弱点在哪方面

3.指导我们设计能够增加覆盖率的测试用例，有效提高测试质量，但是测试用例设计不能一味追求覆盖率，因为测试成本随覆盖率增加而增加。

测试覆盖率对于黑盒测试来说，主要指两个方面：需求覆盖和用例覆盖。

需求覆盖：

1. 定义：它表示在测试系统中，有哪些函数被测试到了，其被测试到的频率有多大，这些函数在系统所有函数中占的比例有多大通过设计一定的测试用例，要求每个需求点都被测试到。
2. 计算公式：需求覆盖=（被验证到的需求数量）/（总的需求总数）

用例覆盖：

1. 定义：主要体现在我们每轮测试验证通过的用例数在总用例中的比例
2. 计算公式：用例覆盖=（验证通过的用例数量）/（总的用例总数）

2-10测试覆盖率的实际运用

简单的测试覆盖率：本次测试执行的用例数/所有用例数

上述覆盖率统计建立在认为总用例数编写全面，一般对于大型系统测试要求覆盖率100%

覆盖率的审核：抽样验收

基于产品的测试覆盖率：已测试需求点/设计所有需求数

以产品、需求维度统计，无论大型项目或小需求迭代都要求覆盖率达到100%

覆盖率的审核：抽样验收

基于白盒的测试覆盖率：大多工具判断语句覆盖，即单元测试代码覆盖代码行/总代码行

更多考察研发人员；更多时候要求覆盖率达到80%+

缺陷：覆盖率数据只能代表测试哪些代码，不能代表是否测试好这些代码；容易遗漏逻辑、判断等场景

基于自动化的测试覆盖率：自动化覆盖的测试场景（测试用例）/所有测试场景（用例）

80/20原则。比如用户80%的时间在使用20%的功能，20%的功能就可以支撑起用户最关键的业务场景，自动化测试的用例选择更着重于这20%的核心功能。

用途：自动化测试更着重于回归验证，没必要追求过高的覆盖率，而要考虑用例设计。

测试覆盖率的最终意义

应用最多的地方在测试停止标准

单纯讨论测试覆盖率，在瀑布式开发模型中并不重要，但在螺旋式、敏捷开发模型中，由于不断迭代累加，很难确定哪些模块在开发过程中没有给予足够的测试。

在短迭代、DevOps中，更强调用单元测试覆盖率来评估不断增加的代码数量。

2-11测试团队组织架构

软件测试人员需要的知识体系

兼容性测试、易用性测试

测试用例设计方法

软件测试流程

软件测试基础知识

………..

测试文档编写

测试工具使用

缺陷管理

软件测试的原则

原则1：所有的测试都应追溯到用户需求

“产品缺陷的80%以上是在产品开发过程中需求定义阶段引入的，如果需求得到了准确的验证，则可以消除80%的返工问题，节省总项目投入费用的45%”

原则2：尽早启动测试工作

原则3：Pareto法则应用于软件测试

Pareto（帕累托）法则是由意大利经济学家帕累托提出的，又称为28效率法则。

测试中的Pareto法则是说一般情况下，在分析、设计、实现阶段的复审和测试工作能够发现和避免80%的缺陷，而系统测试又能找出其余缺陷中的80%，最后4%的缺陷可能只有在用户的大范围，长时间使用后才会暴露出来。

原则4：穷尽测试是不可能的

由于很少有机会对一个应用软件进行所有可能的测试，对大多数软件开发项目来说，利用风险分析是适当的。这需要判断技能、常识、感觉和经验。如果有正当理由，也可以采用正式的方法。

原则5：杀虫剂怪事

软件测试越多，其对测试的免疫力越强。

为了克服杀虫剂怪事，软件测试员必须不断编写不同的，新的测试程序，对程序的不同部分进行测试，以找出更多的软件缺陷。

原则6：前进两步，后退一步

测试中的一个基本问题是——缺陷修复总会以（20-50）%的机率引入新的缺陷。

每次修复后，必须重新运行先前所有的测试用例，从而确保系统不会以隐蔽的方式被破坏。

原则7：三心二意

细心、信心、耐心

团队合作的沟通意识、时刻保持怀疑的态度且有缺陷防范意识。

2-12软件工程标准

国内通用的软件工程标准主要有ISO9000及CMM

ISO9000系列标准是ISO国际标准化组织TC/176技术委员会制定的所有国际标准，其核心标准是质量保证标准（ISO9001/2/3）和质量管理标准（ISO9004）。

CMM（Capability Maturity Model）即软件能力成熟度模型，是向软件组织提供如何增加对其开发和维护软件过程的控制能力。

ISO9000：

ISO9000系列标准的基本思想，最主要的有两条：

一是控制的思想，即对产品形成的全过程——从采购原材料，加工制造到最终产品的销售，售后服务进行控制。

二是预防的思想，通过对产品形成的全过程进行控制以及建立并有效运行自我完善机制达到预防不合格，从根本上减少或消除不合格产品。

ISO900

产品数据

过程

产品

需求数据

资源（人员和材料）

技术

工具

CMM：

CMM准确来说不是标准，只是对过程能力的评估结果。

CMM对软件企业的评估从初始级开始，共分为5级，一级一级的改进，一级一级的向上提高。等级的上升过程是一个“动态渐进”的过程。

CMM是专为软件开发组织设计的，侧重于软件开发和改进过程，在产品的设计和开发的细节作了较多要求。

2-13软件测试规范

测试系统主要由下面6个相互关联、相互作用的过程组成:

测试规划

测试管理

测试设计

资产管理

测试实施

配置管理

软件测试过程中一般会从以下几个方面入手来规范过程，并在每个子过程明确角色、职责、活动描述及所需资料。

推出的 准则

验证与确认

输出项

活动过程

输入项

进入的准则

角色的确定

3-1软件测试环境搭建原则

搭建测试环境前

* 确定测试目的
* 功能测试，稳定性测试，还是性能测试，测试目的的不同，搭建测试环境时应注意的点也不同。

例：功能测试：不需要大量的数据，需要覆盖率高，测试数据要求尽量真实。

例：性能测试：可能需要大量存量数据或者与实际硬件环境尽可能相似的硬件配置。

* 测试的软件环境尽可能的模拟真实环境
* 尽可能的模拟用户使用环境，选用合适的操作系统和软件平台。
* 了解符合测试软件运行的最低要求及用户使用的硬件配置。
* 了解用户常用的软件，避免所有配置所有操作系统下都要进行测试，没有侧重点，浪费时间。
* 产品化的测试则需要考虑兼容性的方案。
* 营造独立的测试环境
* 不同的项目、不同的公司会对测试环境的独立性有不同的要求。
* 测试过程中尽量保证测试环境独立，不会受其他测试人员及项目研发人员的影响。
* 构建可复用的测试环境
* 通过备份或数据隔离的方式。
* 重复运用一套测试环境进行多版本多时间段的测试。

搭建测试环境过程分析

* 线下搭建
* 独立的测试服务器或虚拟机
* 测试环境配置
* 测试项目导入

测试环境配置

* 配置java环境（下载jdk并配置环境变量）
* 下载并安装中间件（tomcat、jetty或其他）
* 安装数据库并导入初始化脚本

Docker模式

* 构建属于自己的image
* 一键deploy

依赖第三方平台

* 依赖第三方平台（如蚂蚁金融云）

3-2浅谈测试环境的建设落地

环境建设思路

考虑点：用途、使用成本、维护成本

基本架构：

研发环境：用于研发自测、集成测试

测试环境：用于日常单系统或两两微服务之间测试，可同时集成自动化测试回归。

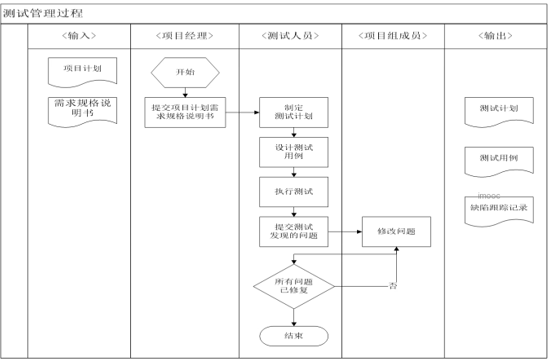
联测环境：完备环境，用于大型联测

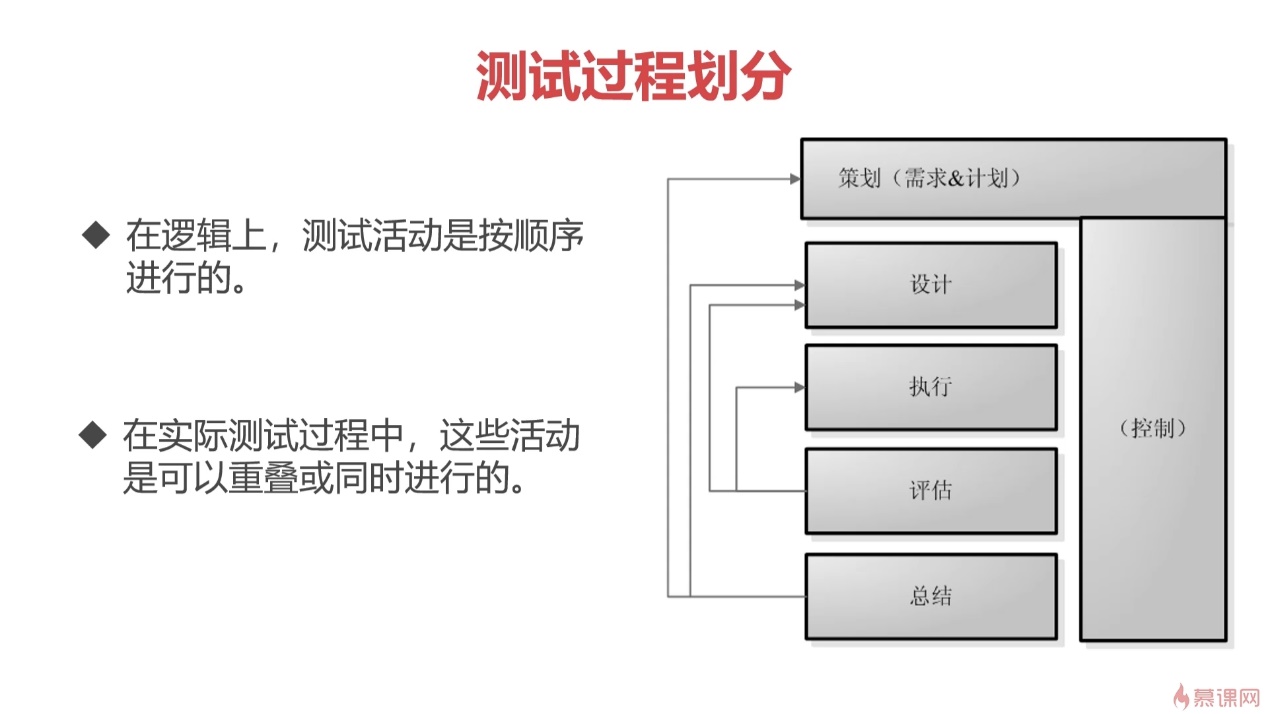
外联环境（如果有需求）：稳定版本环境，用于外部商户联调

灰度/沙箱环境：用于生产数据测试，仿真测试

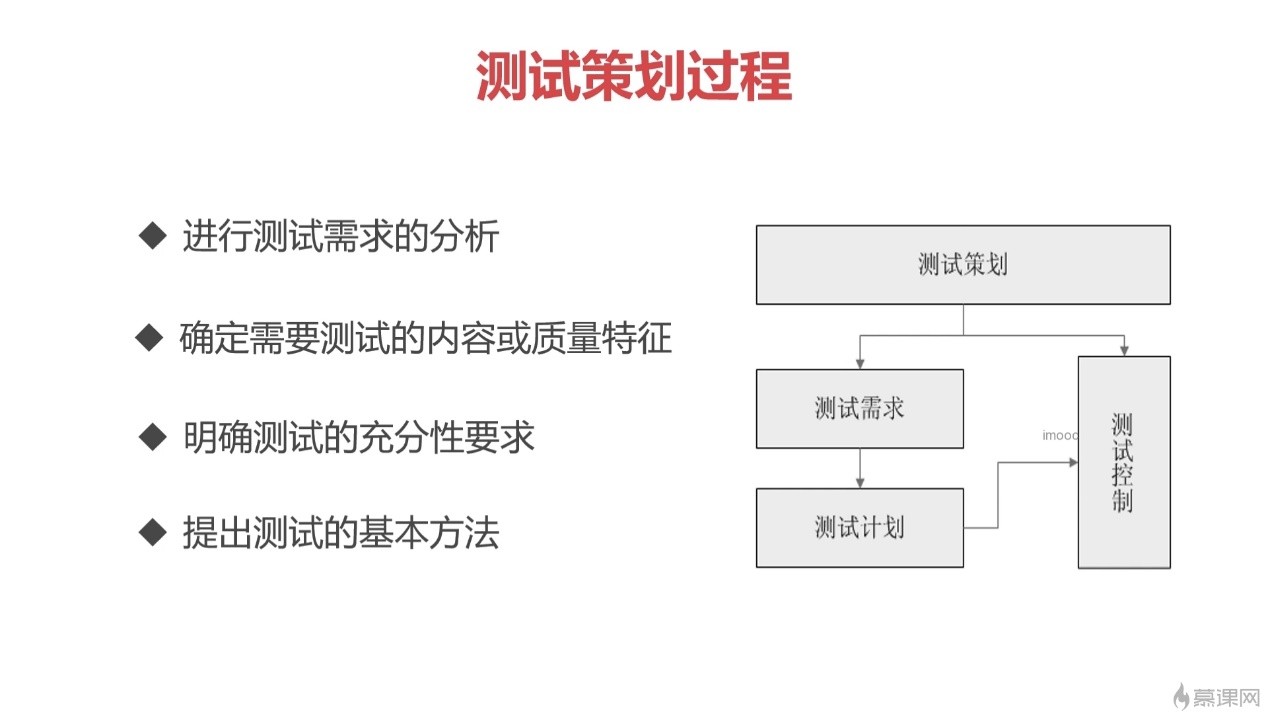
3-3测试过程

简单的测试过程





3-4测试策划

Hi 

* 确定测试的资源和技术需求
* 进行风险分析与评估
* 根据上述分析结果制定测试计划
* 根据测试计划开展相应的测试控制活动

3-4-1需求测试

需求分析

* 过往的软件呢生命周期中，需求分析阶段是测试人员参与的。
* 但随着软件过程的优化，测试人员的加入对需求的分析阶段有了更大的作用。

具体体现在

* 测试工程师参与需求分析，对需了解很深刻，减少开发人员的交互，节省时间。
* 早期确定测试用例的编写思路，为测试打好了基础。
* 可以获取一些测试数据，为测试用例设计提供帮助。
* 可以发现需求不合理的地方，降低测试成本。

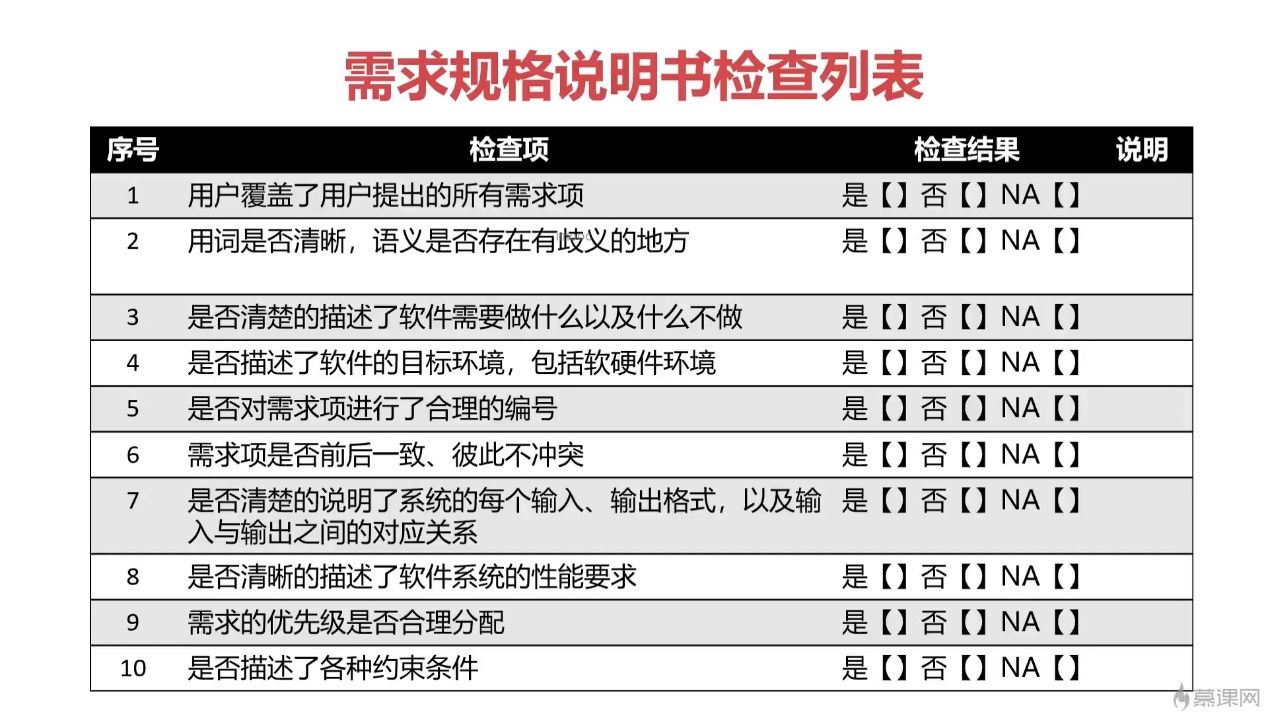
需求测试的作用

* 测试需求的分析用来确定整个测试工作，明确测试对象以及测试工作的范围和作用，并作为测试覆盖的基础。
* 被确定的测试需求项必须是可核实的，测试需求必须有一个可观察、可评测的结果。
* 如果无法核实的需求就不是测试需求。
* 测试需求的分析还包括与客户的交流以澄清某些混淆。
* 明确哪些需求更重要。
* 确保风险承担者尽早的对项目达成共识。
* 并对将来产品有个清晰的认识。
* 测试需求是制定测试计划的基本依据。
* 测试需求是设计测试用例的指导。
* 确定了要测什么，测哪些方面才能有效设计用例。

需求验证

* 审查需求文档
* 对需求文档及相关模型进行仔细检查
* 另外在需求开发期间所做的非正式评审也是有所裨益的
* 以需求为依据编写测试用例
* 编写用户手册
* 在需求开发早期即可起草一份浅显易懂的用户手册，用以描述所有对用户可见的功能并用它作为需求规格说明的参考并辅助需求分析。
* 确定合格标准
* 让用户描述什么样的产品才算满足他们的要求和适合他们的使用
* 将确认合格的测试建立在使用情景描述或使用实例的基础之上。

3-5余额宝需求测试实战



3-5-1测试需求实战

* 以支付宝上新增余额宝业务为例分析
* 原始需求
* 早在2012年左右，支付宝虽然很快被大众接受，但是却面临着一种比较普遍的现象：支付宝账户余额内总是有一笔闲置资金，虽然不同账户资金数额有多有少，但总体来说，这笔躺在账户什么做不了的闲置资金数额还是比较庞大的，对于支付宝的发展而言非常不利。
* 于是，产生了这样一个需求，与基金公司合作推出货币基金产品，同时用户购买货币基金后，可直接通过货币基金金额进行支付购买商品或服务。
* 货币基金可以视同余额，集分宝一样作为支付工具进行消费

3-5-2测试策略

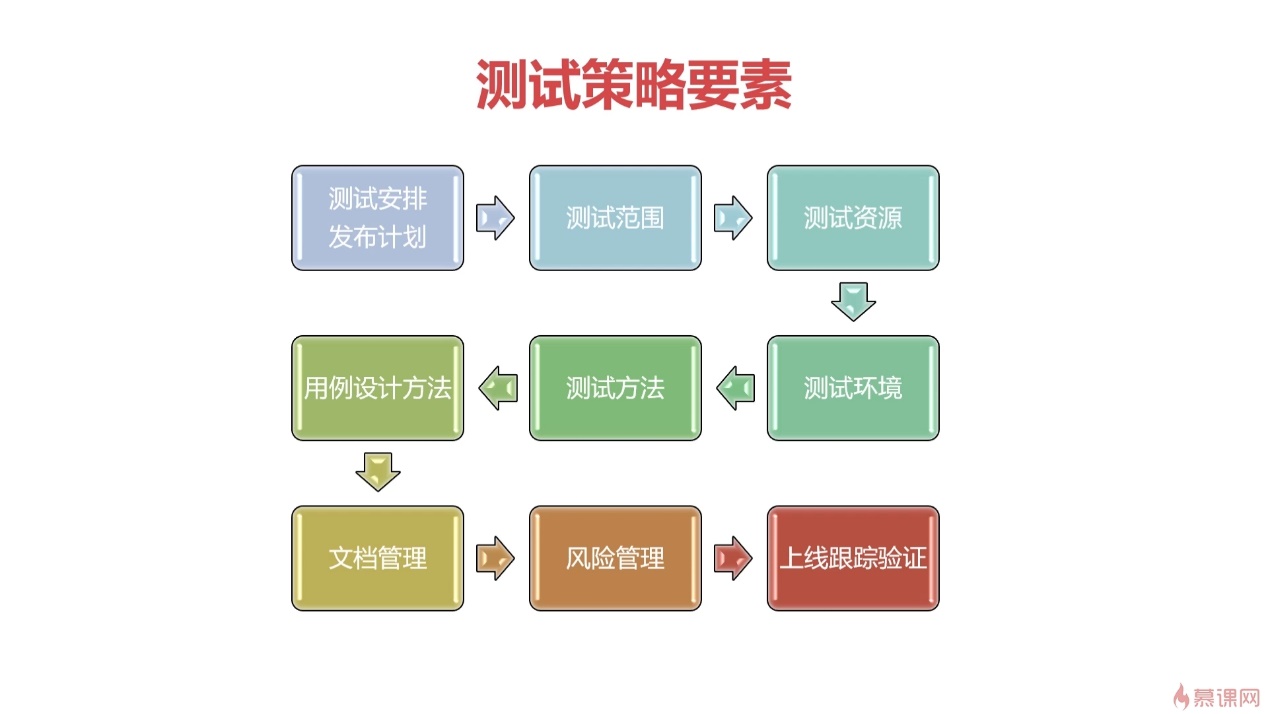
测试前的思考

* 你知道要测试的系统是干什么的吗？
* 你了解系统有些什么特点吗？
* 系统有些什么功能？
* 系统哪些部分需要测试？哪些不要测试？
* 系统对性能有什么要求？
* 系统对安全性有什么要求？
* ……

测试策略是什么

* 测试策略是描述测试项目和测试任务之间的关系。
* 它用来说明要测什么、如何测、如何协调测试资源和测试时间等
* 测试策略制定的是否合理高效会对测试项目的进度产生很大的影响。

如何制定一个好的测试策略并且能防止遗漏呢？



测试策略要素：

1.

* 测试安排、发布计划
* 罗列测试项目本身重要的里程碑
* 每个里程碑都需要有明确的结束时间
* 这个时间可以指导指导我们后续的测试

2.

* 如果测试时间安排不足，我们就可以在后续的测试范围中挑选优先级比较高的特性来执行测试
* 这样可以最大限度的保证产品的质量

3.

* 测试范围（按优先级排列）
* 分为In Scope（在测试范围内）和Out Of Scope
* 需要说明哪些模块是在测试范围中的，哪些是本阶段测试不考虑的
* 对于在测试范围中的模块，需要给出优先级
* 以便相应测试时间不足的情况
* 对于不在测试范围中的模块，需要给出原因
* 为什么在本测试阶段不考虑测

4.

测试资源

* 测试资源在测试策略中也是重要的一环，它分为人力和工具两部分
* 人力资源主要说明参与测试的人员，当然可以包括很多的角色，如专业的测试人员、客户、产品经理等
* 工具主要指可能用到的其他软件

5.

测试环境

* 测试环境主要包括推荐环境解决方案，操作系统要求，软硬件要求
* 对于推荐解决方案，需要陈述的是对测试项目对其他软件的依赖
* 比如测试项目是对JAVA有依赖，推荐版本可能就是1.7

6.

测试方法

* 测试方法的罗列主要是为了说明针对测试项目我们要开展哪些类型的测试
* 功能测试是必须的，非功能测试是可选的

7.

文档管理

* 对于一个完整的产品来说，文档是很重要的一环
* 它一般包括安装、升级文档、用户指南等
* 文档不单单是一个文件
* 它需要经过完整的测试才能发布给客户
* 差的文档很可能会误导用户，从而使他们对测试项目失去信心

8.

风险管理

* 风险管理模块需要罗列出来现在已知的可能出现的不确定性的因素
* 这些因素可能来自于技术，资源或其他方面的

3-6测试方案设计

3-6-1测试计划与测试方案

测试策略：侧重需求分析，评估风险，定义测试范围

（确定测试方法，制定测试启动、停止、完成标准和条件）

测试计划：制定项目测试过程中的测试重点

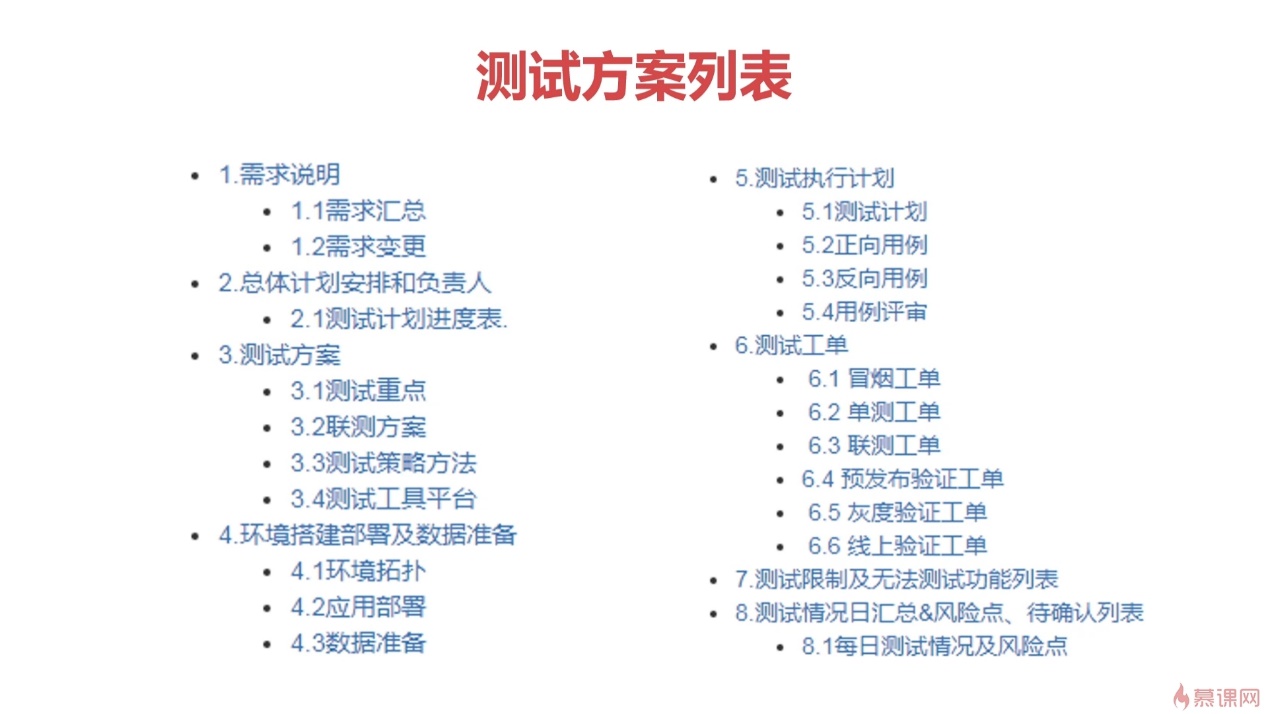
1. 各个阶段的任务分配以及时间进度安排，2.并提出对各项任务的评估，风险分析，可以包括测试策略）

测试方案：侧重测试的方法，测试环境的规划

（测试工具的设计和选择，测试用例的设计方法，测试代码的设计方案）

测试方案=测试计划+用例设计方案+工具选择+自动化/性能测试具体方案

测试计划=测试策略+测试任务分配+时间进度安排



货币基金消费测试方案分析过程

* 1.分析需求：当前测试包含需求项（需求文档或wiki链接等）
* 2.测试计划（里程碑）及负责人：整理当前项目各模块测试负责人，任务分配及测试时间安排
* 3.测试范围、测试重点：那些point需要测试，重点放在什么地方，优先级安排
* 4.测试策略及工具：是否需要进行自动化、性能、安全测试？使用哪些工具
* 5.测试用例设计方法：使用什么样的黑盒测试方法进行设计（等价类？边界值？因果图？等等）
* 6.测试环境：测试环境是什么？需要哪些服务器、数据库？配置如何等
* 7.联调测试：是否需要第三方或其他部门联调？何时开展？联调包括哪些功能？例如基金公司
* 8.测试限制：在测试环境中哪些内容无法测试？比如消费到账
* 9.测试风险：在测试或计划测试过程中由于时间安排、测试限制、优先级分布可能带来的测试风险考量

3-6-2测试方案的评审

测试方案评审

* 目前，开发有需求说明会、设计评审会、代码复审会等各种会议
* 但多是站在开发的角度，从需求和代码层面进行复审和风险规避
* 在测试环节和测试阶段缺少以测试为主的评审机制和沟通机制

容易造成以下几方面的问题：

* 仅从文档、沟通获取信息，可能会造成信息不对称，认识片面，理解错误或不深入等问题
* 缺少同行交叉评审和开发评审机制，无法充分发挥集体智慧，个人的思维难以突破，可能会出现测试遗漏的情况

评审的目的

* 呈现测试的工作
* 与开发达成共识
* 不同的思维方式碰撞出火花思维，借鉴别人的思考方式
* 培养这样的行为模式:愿意为团队或他人出谋划策
* 发挥团队协作，最大限度的发挥个人经验、特长，实现技能互补

评审重点

* 采用的测试方法
* 等价类划分的依据
* 测试数据的选取和准备方法
* 流程测试路径组合
* 数据比对选取的对象和数据检查点
* 是否需要模拟数据及模拟数据的方法
* 基于风险的测试取舍

4-1测试设计与测试用例

4-1-1软件测试设计

测试设计是将概括的测试目标转化为具体的测试条件和测试用例的一系列活动

测试分析和设计的主要任务

* 评审测试依据（需求，系统架构，设计和接口说明）
* 评估测试依据和测试对象的可靠性
* 通过对测试项、规格说明、测试对象行为和结构分析，识别测试条件并确定优先级
* 设计测试用例，并确定优先级
* 确定测试条件和测试用例所需的必要的测试数据

确定测试条件

* 依据在测试策略或测试计划中确定的测试技术
* 通过对测试依据和测试目标的分析，可以确定需要测试的内容，获得测试条件

4-1-2测试用例

概念：测试用例是通过使用在测试计划中确定的测试技术，对于已确定的测试条件进行逐步推敲，精炼而设计出来的重点说明如何具体操作产生何种结果的文档。

测试用例应该具有可重复性，可验证性和需求可追踪性

测试用例设计包括以下关键点

* 前提条件，如项目或局部测试环境的需求，及其交付计划
* 测试步骤
* 测试数据
* 预期结果



4-1-3测试用例常用设计方法

* 等价类划分法
* 边界值法
* 因果图设计法
* 判定表设计法
* 正交实验法

4-2等价类划分法

例：测试一个两位数的加法计算器

测试需求：

* 测试两个参数值相加后结果是否正确
* 其中：输入的数值在-99到99之间，大于99或小于-99的输入应被拒绝，并显示错误信息

等价类划分法

* 根据测试需求，我们开始测试
* 分别给第一个参数和第二个参数输入表中的值，得到的测试结果如表所示

|  |  |  |
| --- | --- | --- |
| 第一个参数的值 | 第二个参数的值 | 两数相加后的值 |
| 1 | 1 | 2 |
| 1 | 2 | 3 |
| 1 | -1 | 0 |
| 1 | -2 | -1 |
| …… | …… | …… |

* 很明显，如果我们对第一个参数的值分别取从-99到99的199个数，第二个参数的值分别取从-99到99的199个数，我们不可能对两位数相加的所有情况进行穷举测试。

如果我们不能进行穷举测试，我们将面临以下问题：

* 在测试了1+1,1+2,1+（-1）和1+（-2）之后，还是否有必要测试1+3,1+4呢？
* 如果不对加法计算器程序进行穷举测试，能否放心的认为所有的参数组合都是正确的呢？
* 对以上两个问题，我们是不是找不到解决的办法？

等价类划分法

* 等价类划分的办法是把程序的输入域划分成若干部分
* 然后从每个部分中选取少数代表性数据当做测试用例
* 每一类的代表性数据在测试中的作用等价于这一类中的其他值
* 也就是说，如果某一类中的一个例子发现了错误，这一等价类中的其他例子也能发现同样的错误
* 反之，如果某一类中的一个例子没有发现错误，则这一类中的其他例子也不会查出错误

等价类划分原则

* 如果输入条件规定了取值的范围或值的个数，则可以确定一个有效等价类和两个无效等价类
* 如果一个输入条件说明了一个“必须成立”的情况，则可划分一个有效等价类和一个无效等价类
* 如果输入条件规定了输入数据的一组可能的值，而且程序是用不同的方式处理每一种值，则可为每一种值划分一个有效等价类，并划分一个无效等价类
* 如果我们确知，已划分的某等价类中各元素（例子）在程序中的处理方式是不同的，则应据此将此等价类进一步划分成更小的等价类
* 在确立了等价类之后，建立等价类表，列出所有划分出的等价类

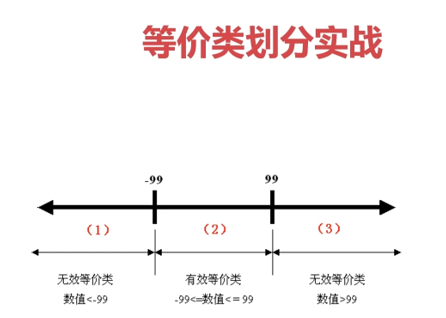
基于等价类划分的用例设计

* 明确测试对象，非测试对象保证正确
* 为每一个等价类规定一个唯一的编号
* 设计一个新的测试用例，使其尽可能多的覆盖尚未覆盖的有效等价类。重复这一步，最后使得所有有效等价类均被测试用例覆盖
* 设计一个新的测试用例，使其只覆盖一个无效等价类。重复这一步使所有无效等价类均被覆盖

等价类划分实战

STEP1:根据测试需求可以分为三个等价类：

* 一个有效数据的等价类，两个无效数据等价类
* 有效数据等价类就是：由那些对程序的规格说明有意义的，合理的输入数据所构成的集合
* 无效数据等价类就是：那些对程序的规格说明不合理的或无意义的输入数据所构成的集合



STEP2:建立等价类表

* 在实际工作中，我们通常在确定了等价类以后，把程序中所有的等价类建立等价类表，以便在编写测试用例的时候有所依据

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 功能项 | 有效等价类 | 编号 | 无效等价类 | 编号 |
| 1 | 两位数加法 | -99≤加数取值≤99 | 2 | 加数取值﹤-99  加数取值﹥99 | 1  3 |
| 2 | … | … | … | … | … |

STEP3:确定测试用例

* 为等价表中的每一个等价类分配一个唯一的编号
* 设计一个新的测试用例,使它能够尽量覆盖尚未覆盖的有效等价类
* 重复这一步骤,从而使得所有有效等价类均被测试用例所覆盖

与上步类似,设计一个新的测试用例,使它只覆盖一个无效等价类

重复这一步骤,从而使得所有无效等价类均被测试用例所覆盖

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例编号 | 输入数值 | 所属等价类 | 预期输出 |
| 1 | -50+24 | 2 | 正确输出:-26 |
| 2 | -130 | 1 | 错误信息 |
| 3 | 125 | 3 | 错误信息 |

STEP4:细化等价类划分

* 在测试”-99≤数值≤99”的等价类区间的时候
* 我们会发现如10+40,-20+30,-30+(-30)这类的正数相加,正数负数相加,负数相加也是不同的等价区间
* 因此我们可以使用更多的等价类划分

根据以上等价类划分的结果,得出下表的等价类表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 功能项 | 有效等价类 | 编号 | 无效等价类 | 编号 |
| 1 | 两位数加法 | -99≤加数取值≤0  0≤加数取值≤99 | 2  3 | 加数取值﹤-99  加数取值﹥99 | 1  4 |

SETP5:完善测试用例

* 根据上面划分的4个等价类,我们至少需要有5个测试用例

|  |  |  |  |
| --- | --- | --- | --- |
| 测试用例编号 | 输入数值 | 所属等价类 | 预期输出 |
| 1 | 50+2 | 3 | 正确输出:52 |
| 2 | -63+(-20) | 2 | 正确输出:-83 |
| 3 | -30+10 | 2,3 | 正确输出:-20 |
| 4 | -130 | 1 | 错误信息 |
| 5 | 125 | 4 | 错误信息 |

等价类的特点

* 测试相同的内容
* 如果等价类中的一个测试能够捕获一个缺陷,那么选择该等价类中的其他测试也能捕获该缺陷
* 如果等价类中一个测试不能捕获缺陷,那么选择该等价类中的其他测试也不会捕获缺陷
* 如果正确的划分等价类,可以大大降低测试用例的数量,测试会准确有效
* 如果错误的两个不同的等价类当作一个等价类,那就会遗漏一种测试情况

等价类划分要注意的问题

* 不但要考虑有效等价类,也要考虑无效等价类
* 仔细划分,审查划分
* 过于粗略可能会漏掉软件缺陷
* 组织评审

等价类用例设计练习

* 测试需求:余额宝提现到银行卡增加新规则:快速到账(2小时)日限额1W元
* 超过1W元只能选择普通到账

分析过程

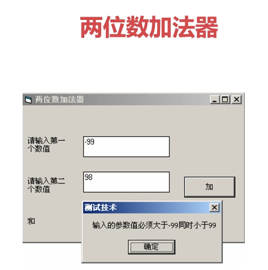
* 设计用例

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 功能项 | 有效等价类 | 编号 | 无效等价类 | 编号 |
| 1 | 余额宝快速提现 | 0＜取现金额≤10000 | 2 | 取现金额＜0 | 1 |
| 取现金额＞1W | 3 |
| 2 | 余额宝普通提现 | 0＜取现金额≤(余额宝总余额)1W | 5 | 取现金额＜0 | 4 |
| 取现金额＞10W | 6 |

* 细致分析需求,日限额1W,所以要区分两个场景

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 序号 | 功能项 | 有效等价类 | 编号 | 无效等价类 | 编号 |
| 1 | 余额宝快速提现(第一次) | 0＜取现金额≤10000 | 2 | 取现金额≤0 | 1 |
| 取现金额＞1W | 3 |
| 2 | 余额宝快速提现(第n次) | 0＜取现金额≤10000-已提现金额 | 8 | 取现金额≤0 | 7 |
| 取现金额＞1W-已提现金额 | 9 |
| 3 | 余额宝普通提现 | 0＜取现金额≤(余额宝总余额)1W | 5 | 取现金额≤0 | 4 |
| 取现金额＞1W | 6 |

4-3边界值法



为什么错了？

* 根据前面的等价类方法，我们按照测试用例表给出的测试用例进行了测试，没有发现问题
* 那么为什么现在输入参数-99和98后，程序提示输入的数据有误了呢？
* 而-99是合理的输入数据，以-99作为输入数据应该是有效等级类中的数据
* 是不是等价类划分方法有问题呢？

边界值分析法

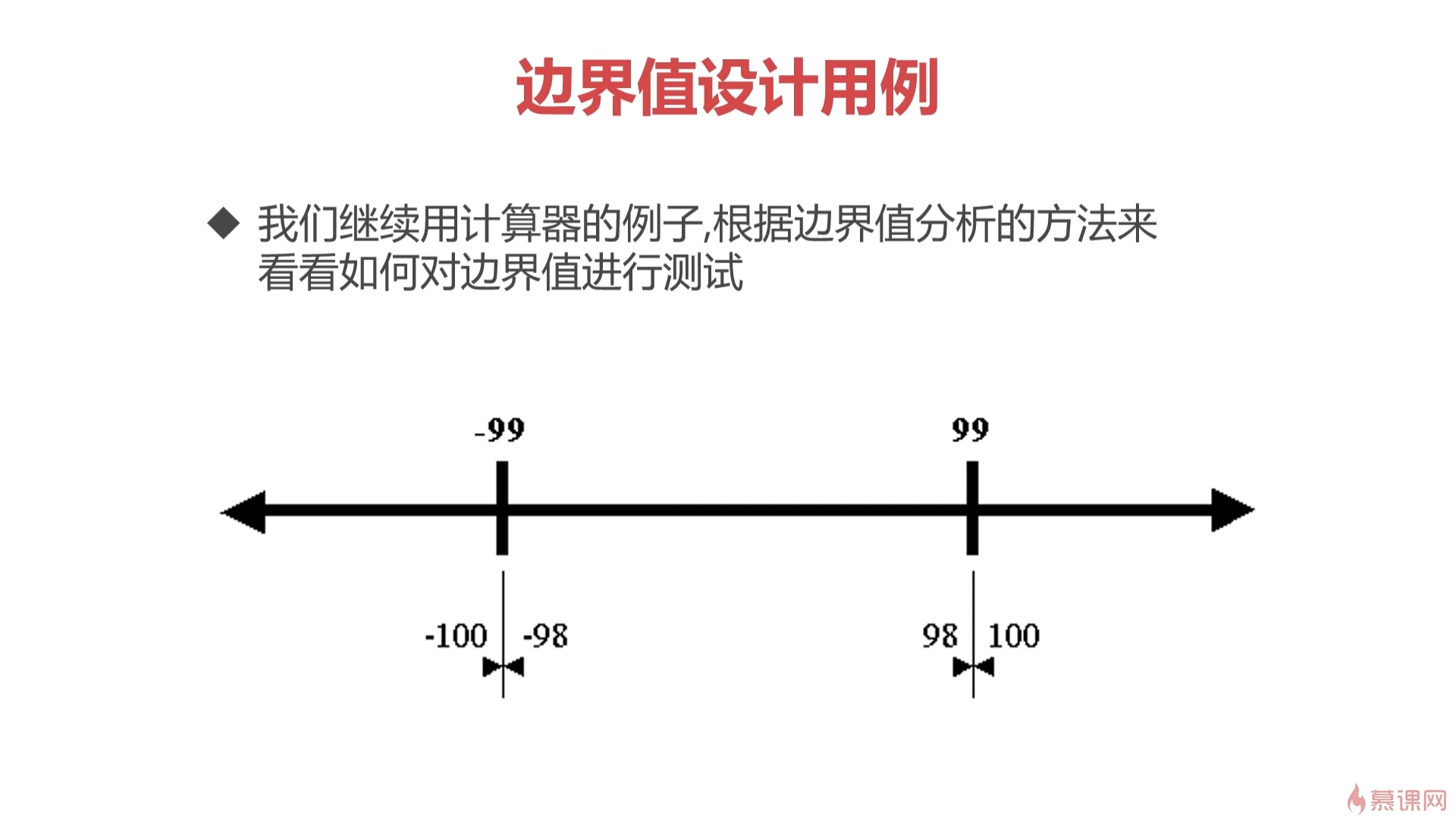
* 边界值分析法是一种补充等价类划分的测试用例技术，它不是选择等价类的任意元素，而是选择等价类边界值的测试用例。
* 实践证明，在设计测试用例时，对边界附近的处理必须给予足够的重视，为检验边界附近的处理专门设计的测试用例，常常取得良好的测试效果。
* 边界值分析法不仅重视输入条件边界，而且也从输出域导出测试用例。

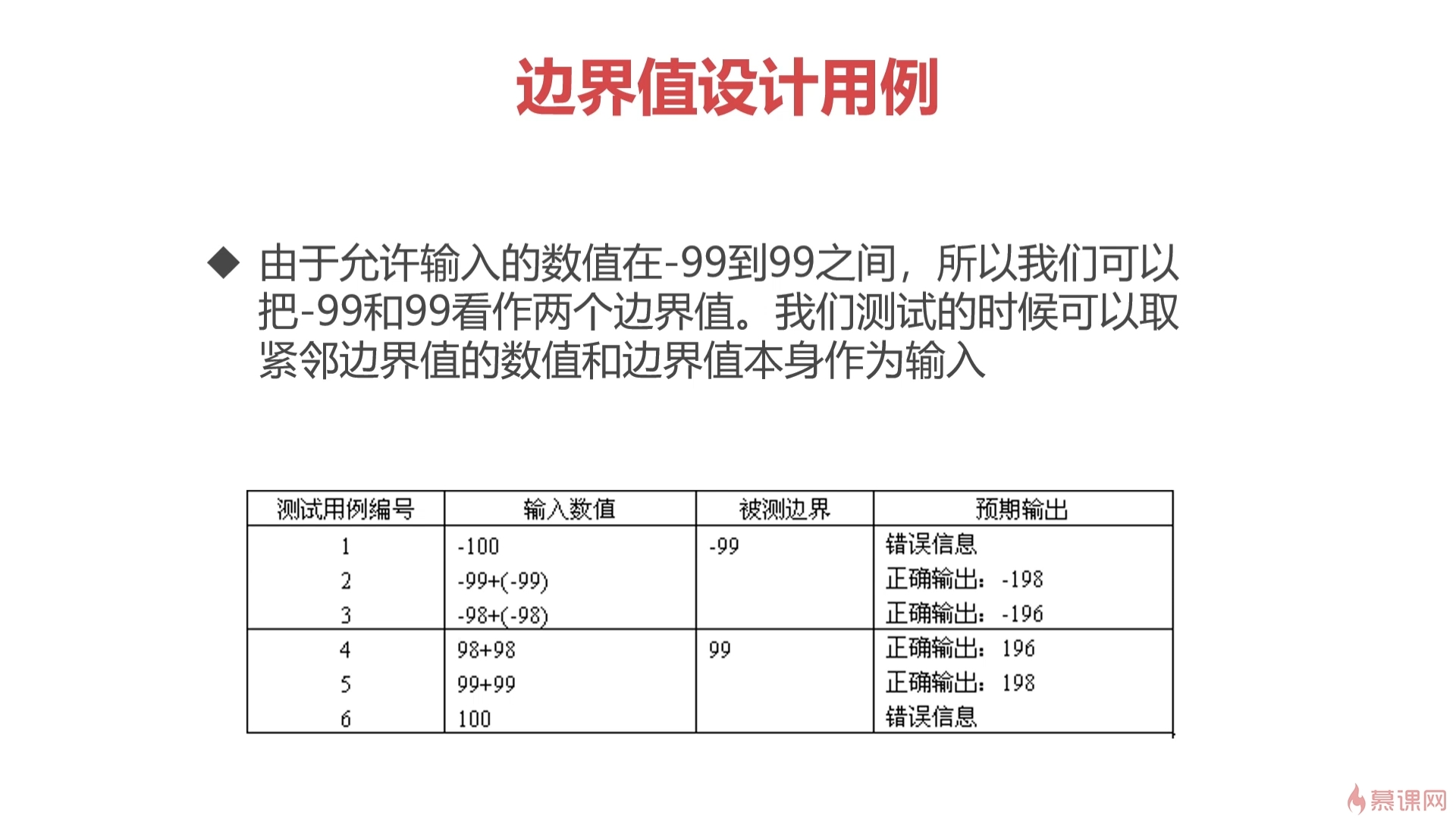
边界值设计的原则

* 如果输入条件规定了取值范围，应以该范围的边界内及刚刚超范围的边界外的值作为测试用例。
* 如以a和b为边界，测试用例应当包含a和b及略大于a和略小于b的值

边界值设计用例

* 我们继续用计算器的例子，根据边界值分析的方法来看看如何对边界值进行测试





边界值用例设计练习

* 同样的测试需求:余额宝提现到银行卡增加新规则：快速到账（2小时）日限额1W元



4-4因果图&判定表法

4-4-1因果图法

* 等价类划分法和边界值分析方法都是着重考虑输入条件
* 而不考虑输入条件的各种组合、输入条件之间的相互制约关系
* 如果在测试时必须考虑输入条件的各种组合，则可能的组合数目将是天文数字
* 因此必须考虑采用一种适合于描述多种条件的组合，产生多个相应动作的测试方法，这就需要利用因果图（逻辑模型）

因果图——判定表

* 因果图法基于这样的思想：一些程序功能可以用决策表的形式来表示，并根据输入条件的组合情况规定相应的操作。
* 因此，可以考虑为决策表中的每一列设计一个测试用例，以便测试程序在输入条件的某种组合下的输出是否正确
* 概况来说，因果图方法就是从测试程序规格说明书的描述中找出因（输入条件）和果（输出结果或程序状态的改变）
* 将因果图转换为判定表，为决策表中的每一列设计一个测试用例
* 这种方法考虑到了输入情况的各种组合以及各个输入情况之间的相互制约关系

判定表

* 判定表（Decision Table）是分析和表达多逻辑条件下执行不同操作的工具
* 在程序设计发展的初期，判定表就已被当做编写程序的辅助工具了
* 因为它可以把复杂的逻辑关系和多种条件组合的情况表达得既具体又明确



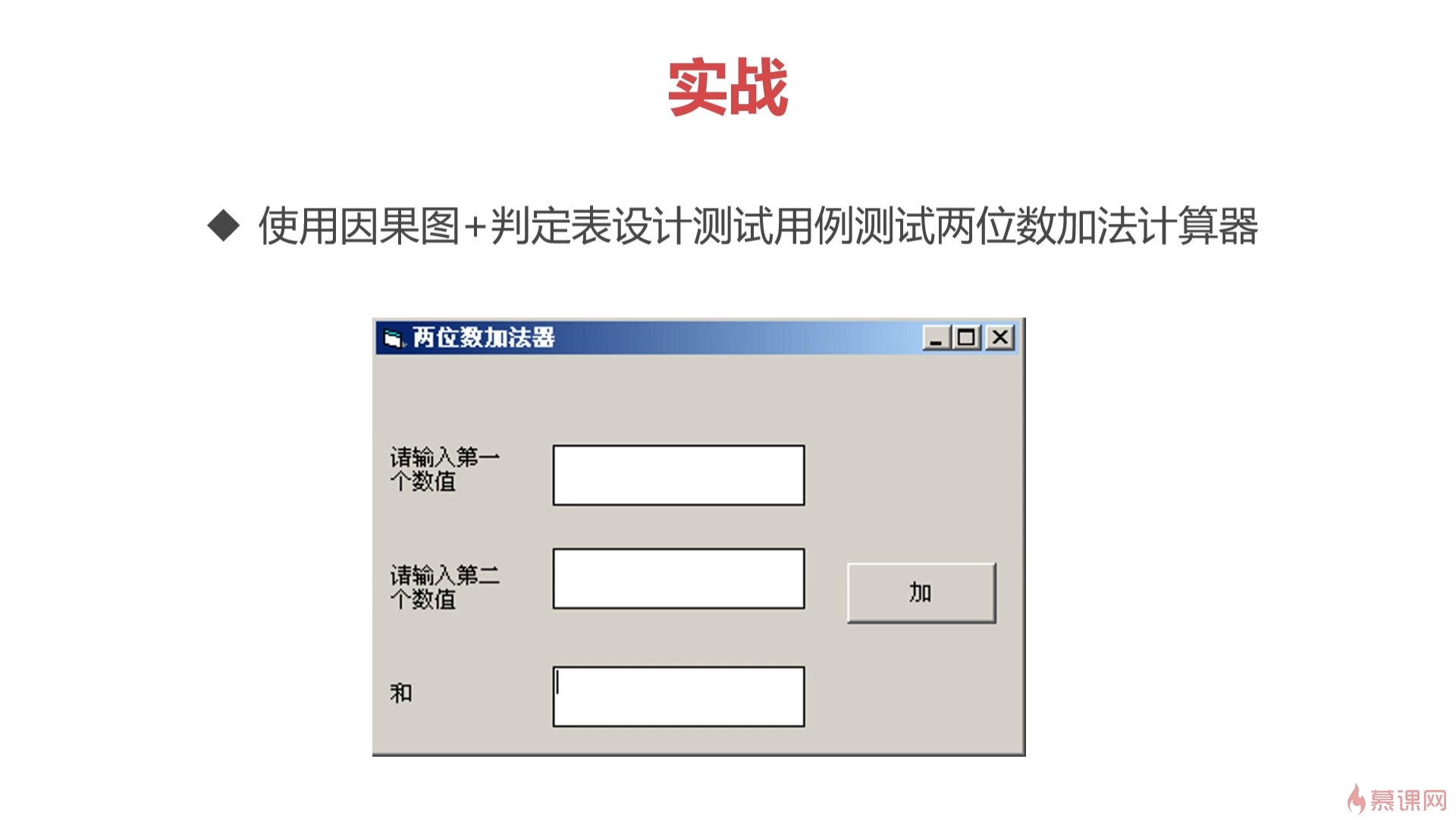
判定表通常由四个部分组成：

* 条件桩（Condition Stub）：列出了问题的所有条件，通常认为列出得条件次序无关紧要
* 动作桩（Action Stub）：列出了问题规定可能采取的操作，这些操作的排列顺序没有约束
* 条件项（Condition Entry）:列出针对它左列条件的取值，在所有可能情况下的真假值
* 动作项（Action Entry）：列出在条件项的各种取值情况下应该采取的动作

设计步骤

1. 分析软件规格说明中哪些是原因（即输入条件或输入条件的等价类），哪些是结果（即输出条件），并给每个原因和结果赋予一个标识符
2. 分析软件规格说明中语义的内容，找出原因与结果之间、原因与原因之间对应的关系，根据这些关系画出因果图
3. 由于语法或环境的限制，有些原因与原因之间，原因与结果之间的组合情况不可能出现。为表明这些特定的情况，在因果图上使用一些记号表明约束或限制条件
4. 把因果图转换为判定表
5. 根据判定表中的每一列设计测试用例

实战



分析输入条件和输出条件

* 输入1：

条件1:0≤X≤99

条件2：-99≤X＜0

条件3：X＜-99

条件4：X＞99

* 输入2：

条件1:0≤X≤99

条件2：-99≤X＜0

条件3：X＜-99

条件4：X＞99

* 输出：

正确计算

错误提示

* 输入

输入1:1,2,3,4互斥

输入2:1,2,3,4互斥

* 输出

输出结果正确与错误互斥





因果图法用例设计练习

* 同理我们将同一测试需求用因果图设计：
* 余额宝提现到银行卡增加新规则：快速到账（2小时）日限额1W元，超过1W元只能选择普通到账
* 按照因果图法设计测试用例

