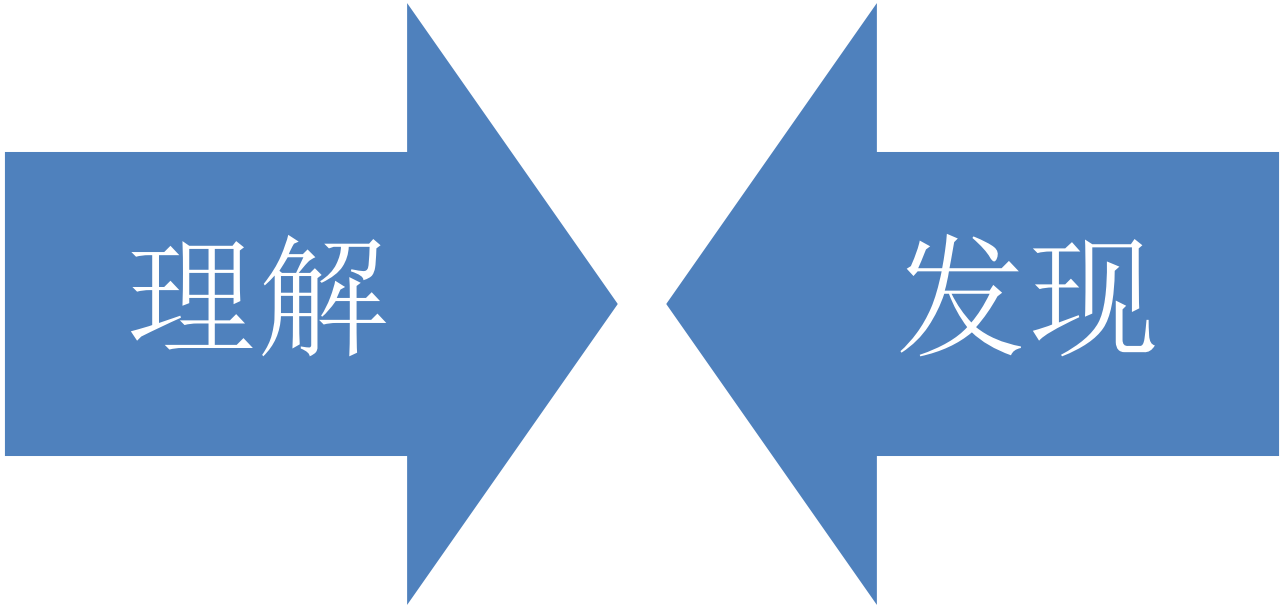
中国软件测试认证委员会

Chinese Software Testing Qualifications Board

软件测试中
模式（Pattern）的应用

陈晟　　软通动力

Key words

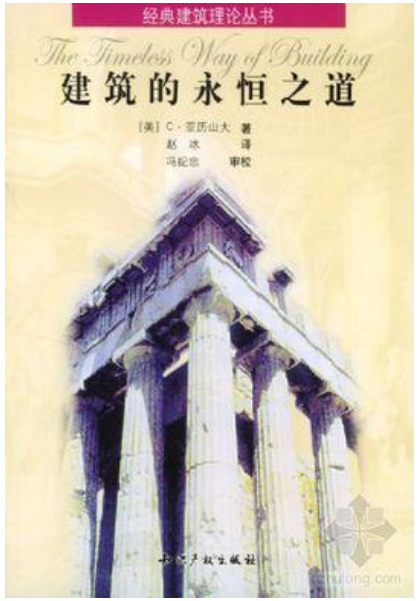


目录

- 模式的概念、层次与测试
- 代码层测试模式分析
- 集成与接口层测试模式分析
- 系统层测试模式分析–以金融行业为例
- 可积累的测试模式框架与模式表达

模式

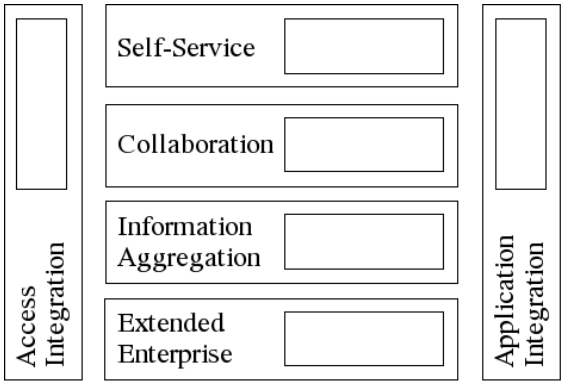
- 每一个模式描述了一个在我们周围不断重复发生的*问题*，以及该问题的*解决方案*的核心。这样，你就能一次又一次地使用该方案而不必做重复劳动
- Problem-Solution pair



-Christopher Alexander

软件生命周期中的模式

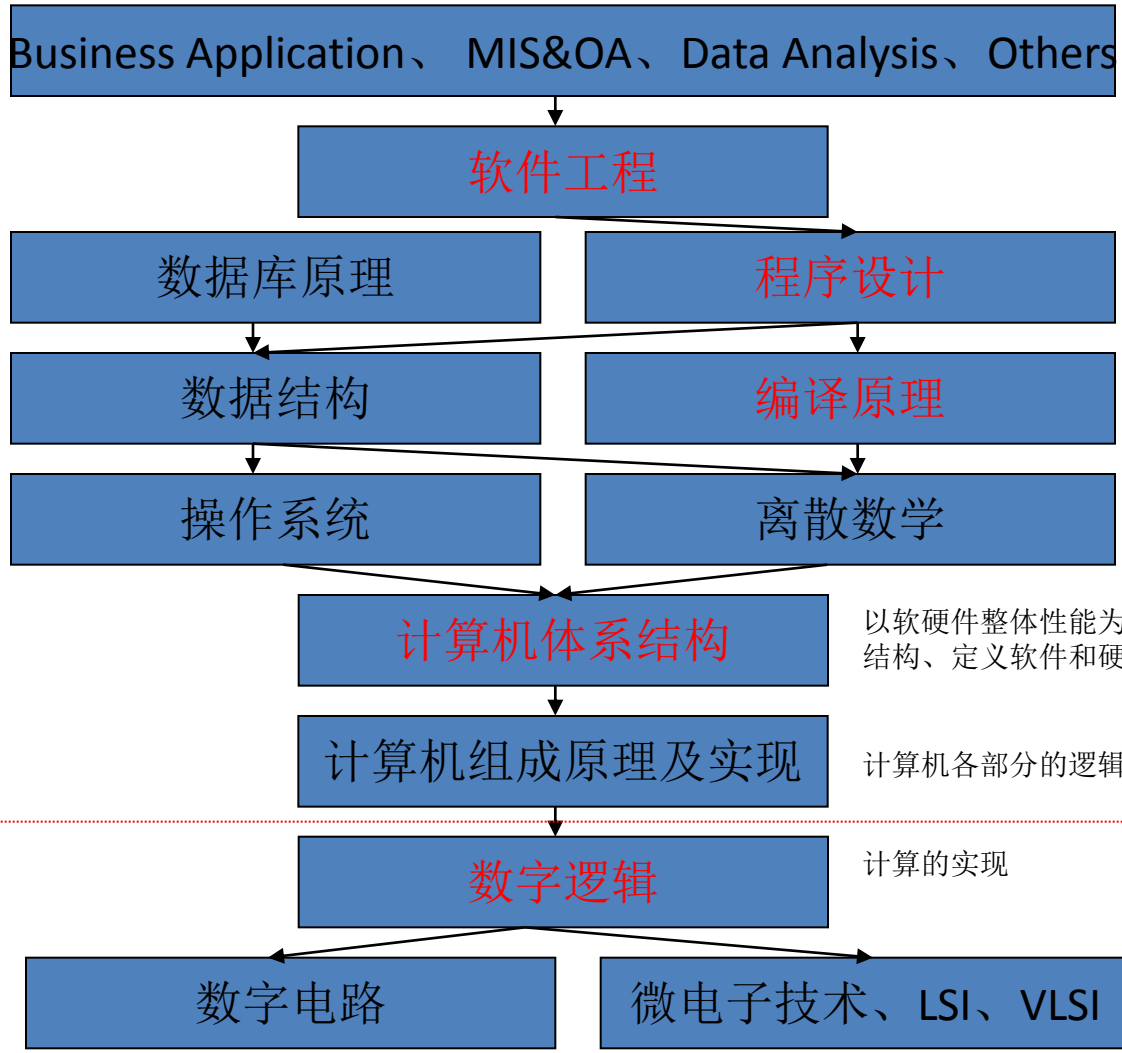
- 业务层面：（缺）
- 分析和规格层面：Martin Fowler
- 设计层面：GOF
- 测试层面：（部分）我们今天探讨的话题

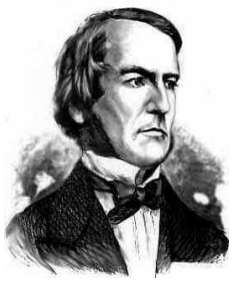


缺陷模式

- 问题
  - 软件测试发现的重大缺陷
  - 可能重复出现的某类缺陷
- 方法
  - 检出该问题的测试方法

基础知识1–软件系统的抽象层次





George Boole, 1815-1864

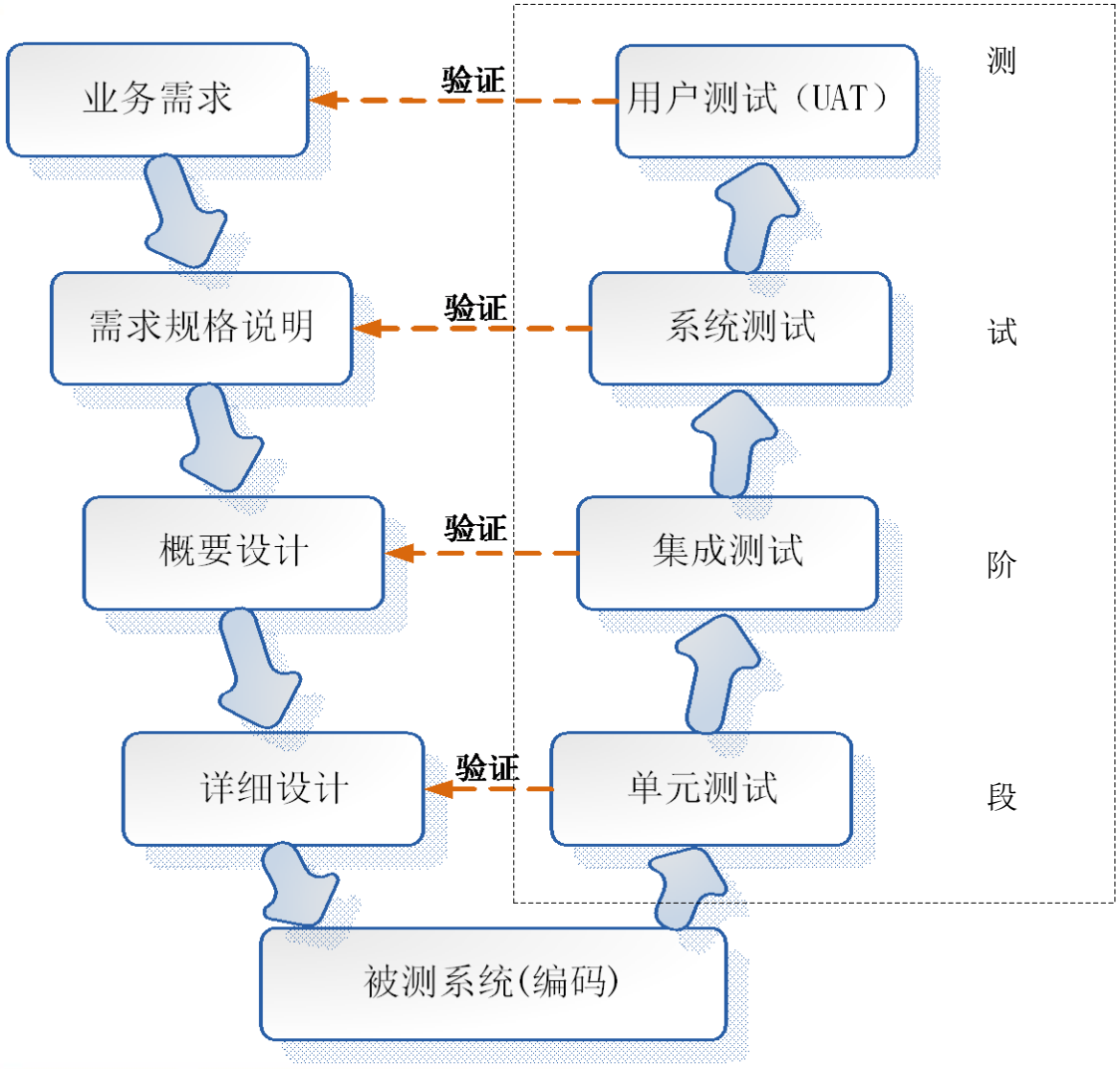
("*A Calculus of Logic*")

Claude Shannon, 1916-

("Symbolic Analysis of Relay and Su

)

基础知识2-分层测试



在各个测试阶段（Test Stage)中，测试对象、测试目的、测试依据、测试关注点在不断的变化，也在采用不同的测试方法。从而，以不同的维度对软件进行全方位的测试。

测试级别（Test Level）与测试阶段为同一概念.

目录

- 模式的概念、层次与测试
- 代码层缺陷模式分析
- 集成与接口层缺陷模式分析
- 系统层测试模式分析–以金融行业为例
- 可积累的缺陷模式框架与模式表达

代码层缺陷模式-案例

Ariane 5 阿里亚娜5号火箭案例

- **情形：1996年6月4日，阿里亚娜火箭在升空40秒后偏离飞行轨道，解体并爆炸。火箭价值1亿美元，同时载有价值5亿美元的通信卫星；**

- 这是一个著名的案例，几乎所有的的软件工程书上都会提到这个案例－ “一行代码错误带来了6亿美元的直接损失”；
- 让我们来看看这是一行什么样的代码？

- Ada 语言代码

```
begin
double d_bh; short s_bh;
sense_horizontal_velocity(&d_bh);
s_bh = d_bh; // OPERAND ERROR
end;
```

- 在飞行过程中，水平速度产生了一个很大的值，该值存贮在一个浮点型的变量中，在向一个短整型变量赋值的过程中，产生了溢出，该溢出导致程序异常，而该异常并没有被捕获和进行保护处理；
- 调查报告的原文如下：

during execution of a data conversion from 64-bit floating point to 16-bit signed integer value. The floating point number which was converted had a value greater than what could be represented by a 16-bit signed integer. This resulted in an Operand Error.

- 测试情况
  - 该程序是经过严格测试的，但测试中使用的数据是Ariane4号的实际飞行轨道数据，而Ariane5的指标与“4”大为不同，因此在测试中没有出现此类数据；
  - 根据事后的调查报告，此类数据转换代码在程序中共有7处，其中4处进行了异常捕获和保护，3处未进行保护，出错即是其中的一处。未保护的原因是程序员认为该数据不可能达到溢出的情况；
- 思考：
  - 此类缺陷应是在单元测试中重点解决的；
  - 国外的程序在90年代即已开始进行严格的测试，但测试并不能完全保证程序不出错；
  - 我们经常能听到程序员说：“这种情况是不可能出现的....”。但是，今天的代码明天就可能复用在更一个地方，程序员提出的假设前提在复用时可能早就被忘记了；
  - 代码扫描型的白盒测试工具，甚至是编译器都会对此类代码报出警告，做为开发人员和单元测试人员，一定要重视这些潜在的问题。

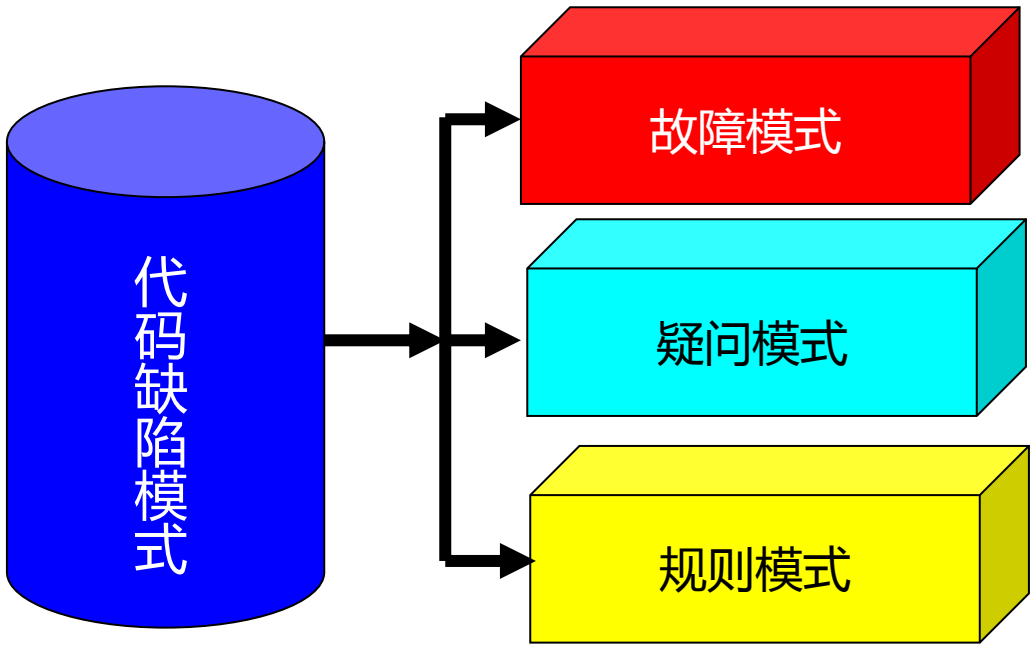
代码层缺陷模式-案例二

- 这是一个简单的案例，C语言代码如下：

```
void someFunction()
{
    char someString[10];
    ……//using of someString
    free someString;    //一个在堆上静态分配的空间被强制释放掉了！
}
```

- 这个案例发生在上个世纪九十年代早期，做为一个工程编程经验不够充分的新手，随手写下了这样的代码；
- 带着这个代码的程序（一个公安系统的指挥调度程序）在运行过程中经常莫名的异常退出；
- 为了查这个问题，两个小硕士在公安局的顶楼一遍一遍地走读代码，他们的老板，一个有经验的硕士甚至是认为碰到了Windows的一个BUG…. 最终问题还是找到了……☺

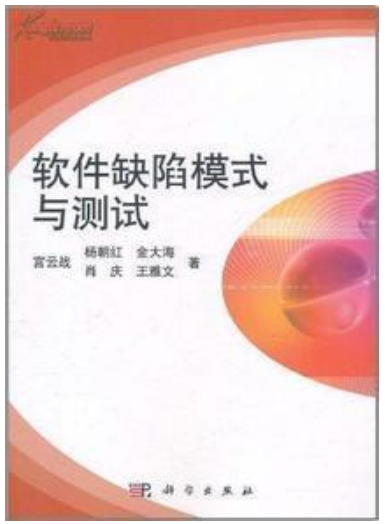
代码-缺陷模式



系统运行过程中产生的异常或错误会造成严重后果

容易引起质疑的不当逻辑及低效操作

针对既定的开发标准所制定的规则



北京邮电大学宫云战教授团队研究成果

代码-缺陷模式

- 1故障模式
  - 1.1内存泄露（MLF）
  - 1.2空指针引用（NPD）
  - 1.3数组越界（OOB）
  - 1.4使用未初始化变量故障模式（UVF）
  - 1.5非法计算类故障（ILCF）
  - 1.6死循环故障（DLF）
  - 1.7资源泄漏故障（RLF）

代码-缺陷模式

- 2疑问代码模式
  - 2.1强制类型转换（FDT）
  - 2.2变量赋值未使用（ANU）
  - 2.3return、break语句后面冗余语句（RS）等
- 3规则模式
  - 3.1在switch语句中必须有default语句（NDS）等

1.1内存泄漏的故障模式（Memory Leak Faults）

定义：内存泄漏故障：设在程序的某处申请了大小为M的空间，凡在程序结束时M或者M的一部分没被释放、或者多次释放M或M的一部分都是内存泄漏故障。

MLF有三种形式：

- 遗漏故障：是指申请的内存没有被释放。
- 不匹配故障：是指申请函数和释放函数不匹配。
- 不相等的释放错误：是指释放的空间和申请的空间大小不一样。

内存泄漏示例：本地函数内存泄漏

- 1 void f(){
- 2int *memleak_error;
- 3memleak_error=(int*) malloc(sizeof(int)*100);
- 4}

- 在第4行处报告一个，例子中的malloc函数可以被其它的内存分配函数替换；

19

内存泄漏-示例代码

- 1　listrec *add_list_entry (listrec *entry, int value) {
- 2 listrec *new_entry = (listrec*) malloc (sizeof (listrec));
- 3 if (!new_entry) {
- 4return NULL;
- 5　}
- 6 new_entry->value = value;
- 7 if (!entry) {
- 8return NULL;
- 9 }
- 10new_entry->next = entry->next;
- 11entry->next = new_entry;
- 12return new_entry;
- 13 }
- 在程序的第2行，给变量new_entry分配了内存，当程序在第8行返回时并没有释放该内存，即是第I类MLF。

20

内存泄漏代码举例：指针运算引起的内存泄漏

- 1 void *malloc(int nmemb);
- 2 void free(void* p);
- 3 void f8(){
- 4 int* melleak_error9=NULL;
- 5 melleak_error9=(int*)malloc(100);
- 6 melleak_error9++;//DEFECT, MLF, melleak_error9
- 7 free(melleak_error9)
- 8 }
- 对指针melleak_error9进行了++操作，所以再对原变量进行释放内存操作则是不正确的

21

MLF模式的严重性

- MLF故障是在C++中是非常危险的；
- 若在某个函数中有MLF故障，则当多次运行该函数时，由于申请的内存没有释放，可能会造成内存空间不足而造成系统死机或异常退出；
- 不当的内存释放也将造成程序异常。

22

代码层缺陷及测试的思考

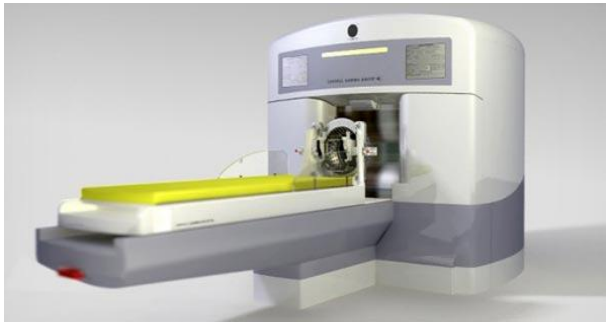
- 资源型的缺陷（模式）通过代码扫描都是可以发现的，可以避免，但更重要的是提高开发人员对于程序设计、对于程序运行时机制的理解；
- “空指针调用”、“资源未释放”等问题都最好是在单元测试中去检查、去排除，白盒测试工具能提供很好的帮助；
- 此类缺陷在运行时表现出来的问题就是程序莫名地报错退出，出错原因极难定位，甚至是在当前消息驱动的程序中不可能从应用层定位。而系统测试人员也只能将此类缺陷粗略地归结到“可靠性”方面的问题；
- 单元测试很重要，如果不做好，后续的测试成本很高，系统故障的修复成本更高！

目录

- 模式的概念、层次与测试
- 代码层缺陷模式分析
- 集成与接口层缺陷模式分析
- 系统层测试模式分析–以金融行业为例
- 可积累的缺陷模式框架与模式表达

接口-集成层面的缺陷模式

■Therac-25辐射治疗仪案例



- Therac-25是Atomic Energy of Canada Limited所生产的一种辐射治疗的机器。由于其软件设计时的瑕疵，致命地超过剂量设定导致在1985年6月到1987年1月之间产生多起医疗事故，5名患者死亡，多名患者严重辐射灼伤。这些事故是操作失误和软件缺陷共同造成的。
- Therac-25有两种电子束设置：低能量模式，可以直接照射病人；高能量模式，需要屏蔽一个X射线过滤镜。

- 用户界面和射线控制器之间的设计存在竞争。一旦操作者选择了一种模式，机器就开始自我配置。如果操作者在8秒之内，撤销了前面的操作并选择其他不同的模式，系统的其他部分并不会接收到新的设置。因为，该机器需要8秒钟才能使磁针摇摆到位。因此，某些操作熟练、动作敏捷的操作者会不经意地增加病人的药量，而这些致命的药量造成了几个病人的死亡。
- 出错的情况是：
  - 1.操作人员首先错误选择了高能量模式（此时，机器将开始配置，而且机器配置是高级别任务，在配置完成的8秒钟内将不接受新命令）；
  - 2. 操作人员撤销前面的高能量模式选择，并选择另一种模式—低能量模式；
  - 3. 操作人员熟练到在8秒钟之内完成其它输入操作，并启动放射；
  - 4. 机器将仍按高能量方式照射病人，而不是操作人员重新输入的低能量模式

- 产品的设计和生产厂家对于此类系统中可能存在的资源冲突情况没有深入的分析，也未对于操作不当引起的资源冲突以及可能带来的严重后果给出必要的提示；
- 由于产品是改进型产品，软、硬件组件已在前代产品上分别长时间应用过，厂家对产品非常有信心，以至拒绝承认问题，导致严重后果；
- 有严格时序关系的流程测试是一个非常复杂的情况，在测试设计中一定要根据被测系统的时序要求细致地设计用例和场景；
- 此类缺陷是在实际生产过程中发现的，而且很难重现，在发生了多起致命和致伤的事故后，才由一个操作非常熟练的医师发现了规律。这也说明，此类测试在系统级测试中的困难；
- 而在系统的设计、编码中，单元测试，特别是模块集成测试中，如果能对系统中的架构、软硬件约束条件、资源冲突原因有深入的认识，有可能在早期的测试活动中发现和定位问题。

接口-集成层面的缺陷模式

■某银行中间业务单边账案例

- 本案例并不是缺陷，而是分析银行中间业务产生单边账的可能原因
- 以网上缴电费业务为例，网上缴电费要经过4个系统:网上银行系统、银行前置系统、银行核心系统 和 电力公司（委托方）的系统，具体的流程如：
  - 1：当你从网上银行发起一个缴电费的请求时。网上银行会将请求报文发给银行前置系统。
  - 2：银行前置系统接到代缴报文时，会给银行核心系统发一个代缴报文。
  - 3：银行核心系统从你的账户转帐到电力公司的对公账户，并给银行前置系统一个应答。
  - 4：银行前置系统收到银行核心系统应答后，判断应答码，失败直接给网上银行系统一个捐款和应答码，成功则继续发送报文到电力公司的系统。
  - 5：电力系统收到请求后会给银行前置系统一个应答。
  - 6：银行前置系统判断电力系统的应答，如果应答是失败或长时间没有收到应答，会向银行核心系统发起冲正交易，将刚才扣的钱返还到你的账户，成功则给网上银行系统扣款成功应答。
- 单边账是指银行和电力公司（委托方）出现数据不一致的现象。

- 如上页的网上缴费等业务是一个跨系统的事务，一般采用异步通信的方式进行，出错时自动冲正或人工撤销。
- 本质是一种“乐观”的处理方法，目的是减少对资源的加锁，减少等待时间，增加系统的并发处理能力，同时实现系统间的松耦合。
- 实现方法
  - 异步消息（队列）
  - 共享数据库（文件）+ 轮询
  - 文件交换（传输）

- 银行方和委托方互知对方的处理结果：正常/失败
- 由于银行和委托方分别使用各自的的应用系统，很难协调一致的采用事务中间件技术来保证事务一致性，一般情况下是在保证各自系统的账务作为一个完整的事务处理的情况下，通过其它方式来交换相互的操作结果，业务规则如下表：

银行方处理结果	委托方处理结果	银行方下一步操作
正常	正常	返回操作成功
正常	失败	自动冲正银行方帐务，返回操作失败
失败	不发往委托方	返回操作失败

说明：这里的委托方指的是电力公司，正常/失败指的是缴费成功/失败。

- 在等待委托方的反馈的过程中“超时”，或未收到委托方反馈，
  - 在这种情况下，对于银行方来说，委托方是否收到了数据是未知的；
  - 即便委托方收到的数据包，其处理情况是未知的，也即不知道委托方的处理结果是正常还是失败；
  - 银行方一般只能是按委托方失败来处理，而此时就有50%的可能性在银行和委托方之间出现单边账。
- 什么情况会引起超时
  - 银行和委托方间网络通信不畅；
  - 银行内部交易量过大，而前置等系统的流量控制功能对通信流量进行了限制，导致通信异常；
  - 银行系统存在接收委托方反馈等方面的故障；
  - 委托方内部网络或系统的问题；

接口-分析

- 接口有哪些作用
  - 数据传输
  - 控制
- 接口有哪些特性
  - 结构特性
  - 交互特性
- 接口有哪些质量要求
  - （网络）连通正确
  - 数据收发正确
  - 数据解析正确
  - 处理机制正确
  - 业务功能正确

接口-测试模式-结构特性(1)

- 针对单个元素的测试模式

	测试项	测试目的	测试方法	举例
1	数据类型	检查数据类型	构造异常数据	如：字母、数字、货币
2	数据格式	检查数据格式	构造异常数据	如：字符串长度和标点符号
3	计量单位	检查数字类型的计量单位	构造异常数据	如：米、元、纳秒
4	范围或可能值的枚举	检查数字类型的值域	构造异常数据	如：0~99
5	准确度和精度	正确程度和精确程度	构造异常数据	如：有效数字位数
6	初始值	检查是否赋初值	构造异常数据	

接口-测试模式-结构特性(2)

- 针对元素集合的测试模式

	测试项	测试目的	测试方法	举例
1	数据元素/数据元素集合体的结构	检查集合结构的符合性	构造异常数据集	
2	集合体中的数据元素关系	检查结构中元素的相互关系	构造异常数据集	如：编号、次序、分组
3	约束关系	检查结构中已定义的强制约束	构造异常数据集	如：是否可被修改、业务规则是否适用
4	访问控制	检查数字类型的值域	构造异常数据集	如：加密
5	来源与目的	检查数据发送和接收实体	构造异常数据集	如：点对点、广播、发布\订阅

接口-测试模式-交互特性

- 针对通信与控制的测试模式

	测试项	测试目的	测试方法	举例
1	连接	检查连接的建立、保持、断开	构造异常交互行为	如：长、短连接
2	数据传输控制	检查传输速率、间隔、周期/非周期	构造异常交互行为	如：定时传输和事件触发传输
3	访问控制	检查信息内容的优先级、时序、同步等	构造异常交互行为	
4	流量控制	检查数字类型的值域	构造异常交互行为	如：序列编号和缓冲区处理
5	错误处理	检查错误控制和恢复	构造异常交互行为	如：超时、重复数据
6	安全处理	检查安全性和保密性	构造异常交互行为	如：加密、用户鉴别、审核

接口-集成测试策略

- 参考接口结构特性测试模式设计接口结构测试
  - 找到数据集合和数据元素
  - 应用等价类和边界值等方法
- 参考接口交互特性测试模式设计接口交互测试
  - 找到控制指令和交互方式
  - 应用等价类和边界值等方法
- 保证测试充分性
  - 每种数据集合和每个交互特性都测试到
- 提供可测试性
  - 通过Driver和Stub，实现接口的数据交互和控制，并使接口可观测；
  - 其它集成方式，并使接口可观测；
  - 需要时，对可观察或访问的接口直接采取修改接口数据的方法
  - 需要时，对交互方式（同步、时序、序列、状态）进行修改
- 特别要检查异常的情况（错误处理机制和恢复）
  - 通信连接和数据收发异常
  - 数据解析异常
  - 破坏同步关系

目录

- 模式的概念、层次与测试
- 代码层缺陷模式分析
- 集成与接口层缺陷模式分析
- 系统层测试模式分析–以金融行业为例
- 可积累的缺陷模式框架与模式表达

系统层面缺陷模式与测试-案例

■挪威$100,000网上银行转账案例

- 2006, 某家挪威银行的网银系统
- 一个老妇人通过网银向她女儿的账户转10万美元
- 她女儿的账号是： 71581555022
- 老妇人输入的账号是：715815555022　（多输入了一个数字“5”）
- 挪威标准的账号是11位长，最后1位是校验和。网银系统在两次接受到同样12位账号的情况下，自动将账号截尾，也即只使用前11位数字。 巧的是，这个截尾后的11位数字正好能通过账号的验证规则，更巧的是这个数字是一个酒鬼的账号；
- 10万美元被酒鬼挥霍掉了，老妇人要求银行赔偿，理由是：“since she typed in 12 digits, it was the responsibility of the system to give her an error message instead of just dropping all digits after the 11th.”。银行拒绝赔偿，理由是：“she hit the confirm button”；
- 在一个由五人组成的银行业消费者仲裁委员会裁定中，2：3，老妇人败诉，理由是：“She made an error and has to take responsibility.”。老妇人随后将银行告上了法庭，最终结果未知；

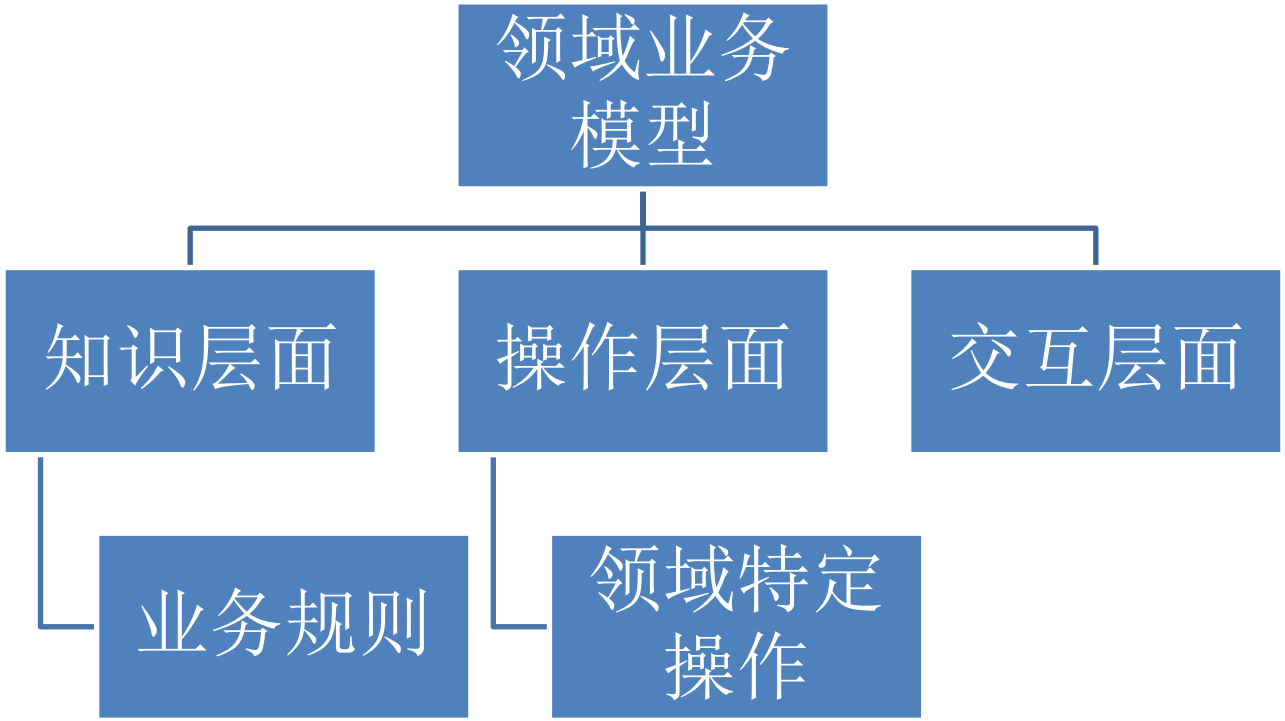
- 系统对于用户不符合要求的输入自动进行了处理，未给客户以必要的提示。这种自动处理不一定是用户的本意；
- 一般情况下，误输的卡号是不符合卡号规则的，但例外情况在这个案例中发生了
- 此类问题，单元测试、模块集成测试都不会认为是缺陷（除非在详细设计中给出明确说明），而只有在系统测试和UAT测试中才会较容易地暴露出来；
- 账号处理是银行软件的基础，此类问题在所有的系统中都将碰到，因此，可做为应用层的一个缺陷模式进行统一处理；
- 在今天电子渠道越来越多的代替传统柜面渠道的情况下（工商的电子渠道业务替代率已超过80%），银行业务越来越多地是由用户直接操作软件完成交易，而不是由受过专业训练的银行柜员完成。如何保证用户能准确、安全、方便地使用电子渠道，是系统设计、开发和测试人员都要关注的问题。

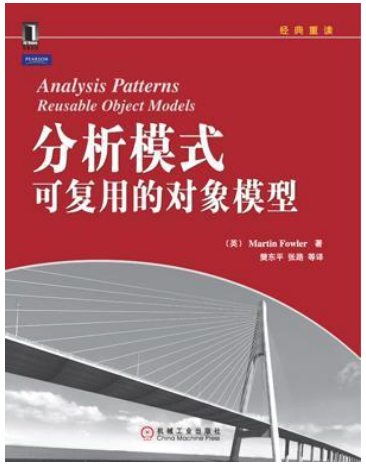
系统层面缺陷模式与测试-案例

- 2007年3月，林先生手中有已买入的认购权证“包钢JTB1”15.35万份，在3月30日（最后一个行权日），林先生进行行权。他利用海通证券网上股票交易系统点击“买入”，交易系统弹出“买入股票”对话框；林先生在“证券代码”栏目填入该认购权证“582002”，交易系统对话框界面显示“证券名称”为“ES070330”（包钢JTB1认购权证的行权简称），“买入价格”为1.94元，“可买（股）”为“269000”份；当他按交易系统提示的这一数字如数填入，委托行权并确认，交易系统对话框当即告知“您的买入委托已成功提交”。
  请注意，269000份是网上交易系统按林先生资金账户余额和行权价格计算出的可买入份数，而不是林先生手中实际的权证数量！
- 与此同时，林先生的资金账户被**证券冻结52.168万元，该交易系统备注栏显示“已报”，但他在3天后查询却发现股票账户内“包钢股份”股票为“0”，海通证券的解释是3天前行权时权证只剩下15.35万份，而行权申报时填单却填了26.9万份，所以行权失败成“废单”。
- 4月2日，当林先生看到“包钢股份”走势不错打算抛出时，他发现3天前委托行权的26.9万股“包钢股份”竟然为“零股”，15.35万份权证全部“化零”，按3月23日“包钢JTB1”权证最后一个交易日的收盘价3.453元/份计算，由此造成的直接损失达53万余元。

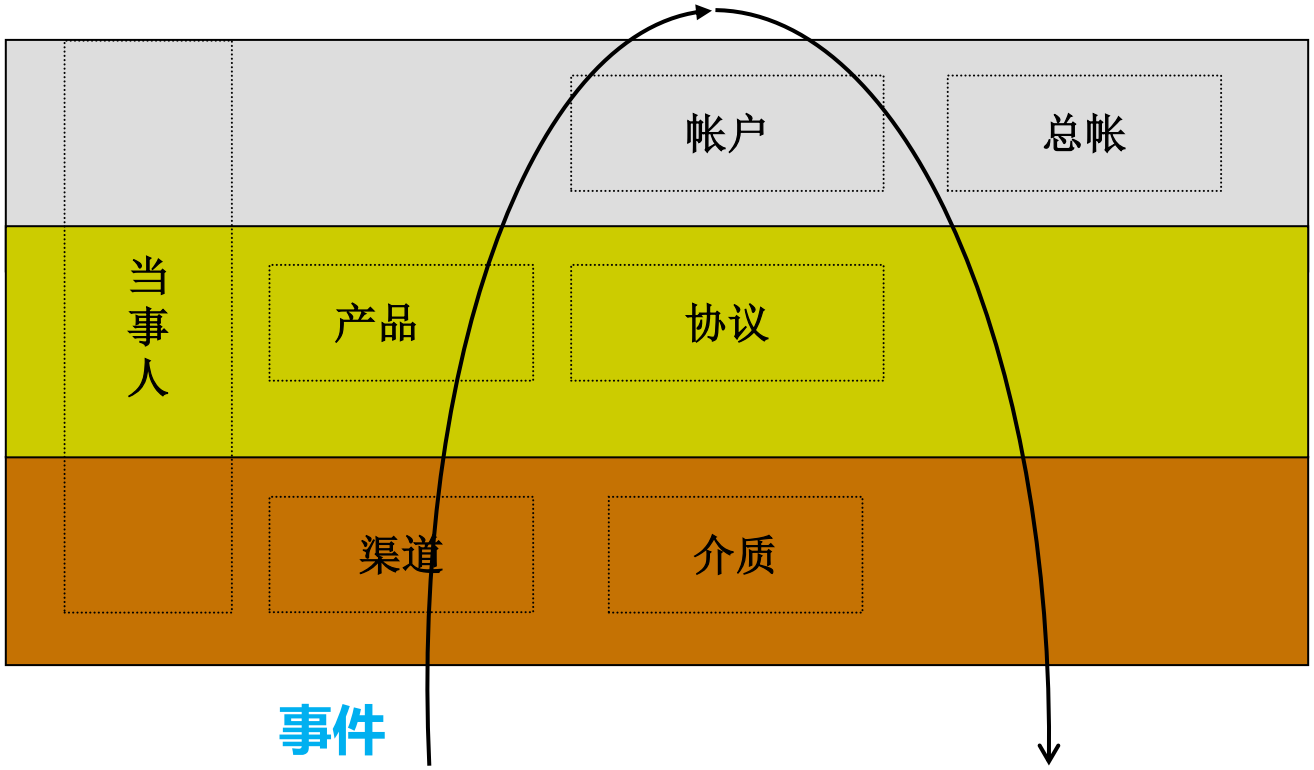
- 券商端系统、交易所端系统业务处理逻辑都正确；
- 但由于交易数量大于限额，该交易被取消。
- 原因：
  - 典型的异步处理系统；
  - 在券商端没有进行交易额的检查，而又由于异步处理的情况，无法及时将交易失败的情况反馈给用户；
  - 券商端在交易前（可行权数量）和交易中（提交成功）给用户的提示信息有很大的误导；
- 深层次原因：开发方和测试方对业务的理解不足。

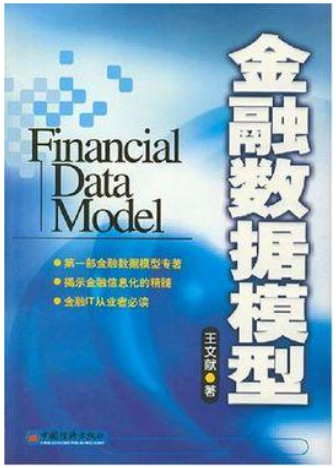
如何来分析


领域业务模型
知识层面
操作层面
交互层面
业务规则
领域特定操作


Analysis Patterns
Reusable Object Models
分析模式
可复用的对象模型

金融/银行–领域主题





缺陷模式–金融/银行–领域主题

分类	缺陷类型/测试项	出现的场景	测试方法	其他
当事人-客户				
当事人-柜员				
产品	费用计算与扣收	费用扣收与本金、利息操作的顺序	发生本金、利息、费用的交易	利率设定与利息计算
协议	额度控制	产品以某协议进行签约操作	申请额度与使用额度不一致的交易 可用金额与实际需用金额不一致的交易	
渠道	数据检查与控制	渠道接收到不符合要求的数据	在上、下游渠道联动中发生不符合要求的数据	
账户				
介质				

缺陷模式–金融/银行–领域主题

分类	缺陷类型/测试项	出现的场景	测试方法	典型缺陷实例
当事人-客户	客户认证、鉴权			
当事人-柜员 当事人-银行 业务单元	柜员认证、鉴权 柜员授权 机构通存通兑			
产品	客户与渠道限制 产品生命周期（期限、提前与展期） 金额上下限 利率确定与变更 利息计算与扣收 费用计算与扣收 复合产品			
协议	额度控制 用户-产品匹配 用户、银行业务单元权利与义务 产品交付协议条款（定价） 产品交付交易条款（活动）			
渠道	渠道信息检查 渠道签约与解约			
账户	账户-产品-客户关系 账户状态 账户权限			
介质	协议-介质关系 介质状态			

目录

- 模式的概念、层次与测试
- 代码层缺陷模式分析
- 集成与接口层缺陷模式分析
- 系统层测试模式分析–以金融行业为例
- 可积累的缺陷模式框架与模式表达

缺陷模式表达

缺陷/测试项

- 缺陷模式的命名，也可对应为测试项

场景

- 描述缺陷出现的场景

测试方法

- 描述适用于检出缺陷的测试方法

典型缺陷示例

- 符合该模式的典型缺陷示例

缺陷模式框架

- 代码层缺陷模式框架

	故障模式	疑问模式	规则模式
1	内存泄露	强制类型转换	
2	空指针引用	变量赋值未使用	
3	数组越界	冗余语句	
4	使用未初始化变量		
5	非法计算		
6	死循环		
7	资源泄漏		

NOTE：以C、C++为例

缺陷模式框架

- 接口层缺陷模式框架

	结构特性	交互特性	其他
1	结构失配	一致性缺陷	

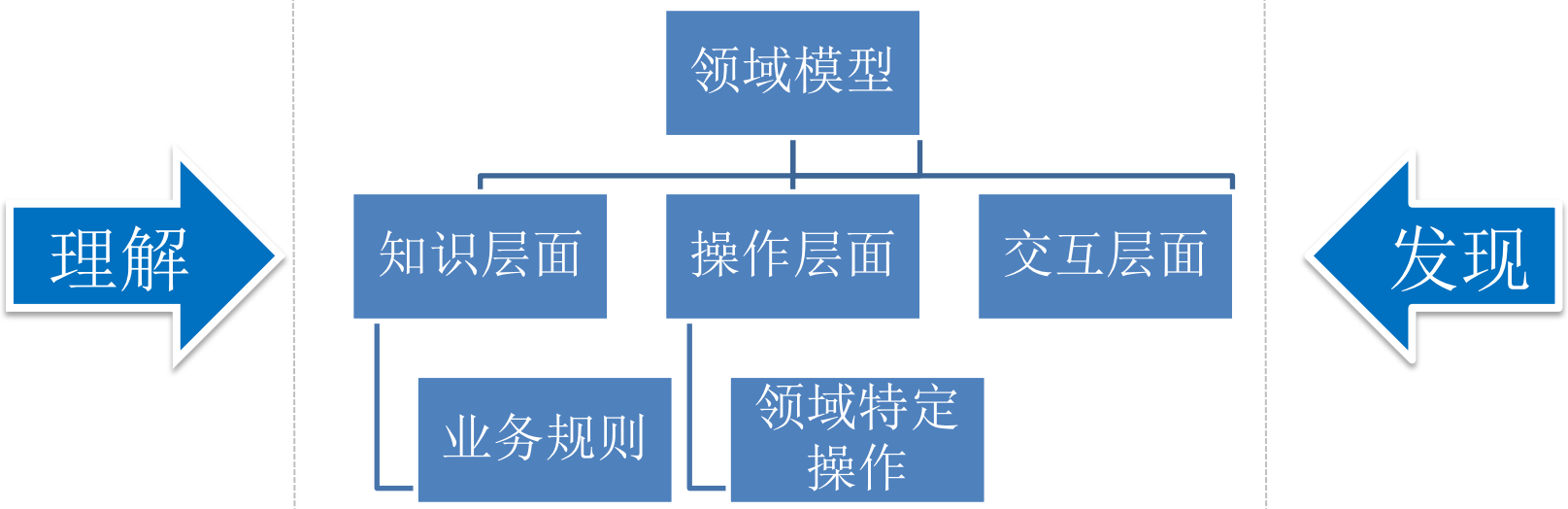
缺陷模式框架

- 系统层缺陷模式框架-金融/银行

	当事人	产品	协议	账户	渠道	介质
1	认证	期限	额度	类型	检查	类型
2	鉴权	金额	权限/交易	状态	签约	状态
3	分级控制	计息		权限	解约	
4	评估	计费				
5	机构权限	复合				
6						

NOTE：以金融/银行业为例

Key words in Testing Pattern



- 实施者：有分析方法-分析类模式是一种方法
- 管理者：抓住风险点-缺陷类模式是一种资产

结束语

- 模式是重用的方法
- 分层次探讨缺陷/测试模式
- 分析模式、设计模式、交互模式有助于测试人员理解系统
- 缺陷模式有助于从分析重复缺陷的角度更好地开展测试
- 模式是一项总结、分析、积累工作

ISTQB®
International Software
Testing Qualifications Board

Thank you

ISTQB®让测试更专业