软件开发的测试管理

李玉芬

主要内容

- 软件测试基础
  - 软件测试理论
  - 单元测试
- 配置管理
  - 源代码管理
  - 缺陷管理
- 开发制胜策略
  - 13条有用的编码法则

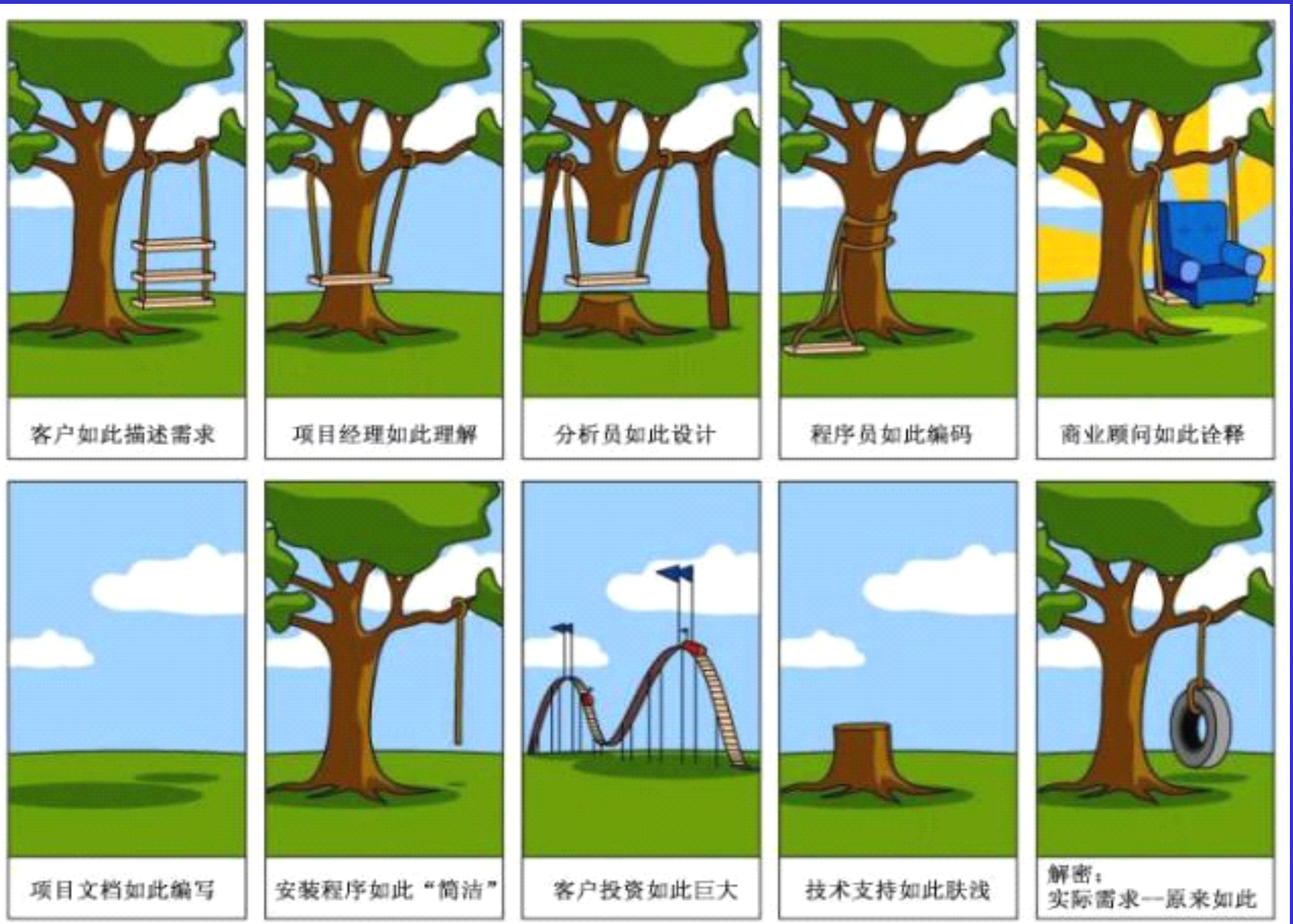
软件测试基础

软件测试

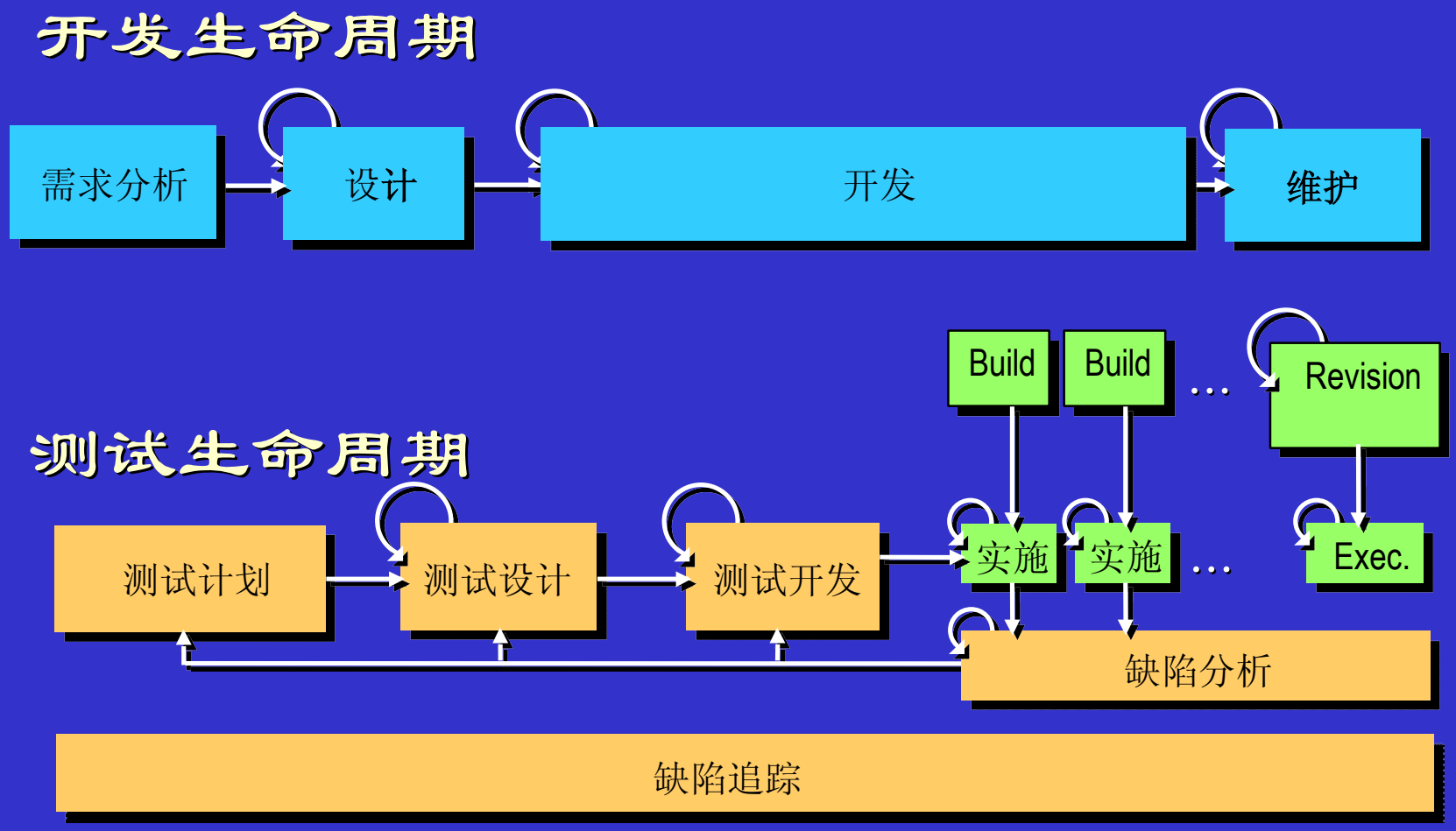
- 软件测试是为了发现错误而执行程序的过程。
- 软件测试是根据软件开发阶段的规格说明和程序的内部结构而精心设计的一批测试用例（即输入数据及预期的输出结果），并利用这些测试用例去运行程序，以发现错误的过程。

软件产生错误的原因

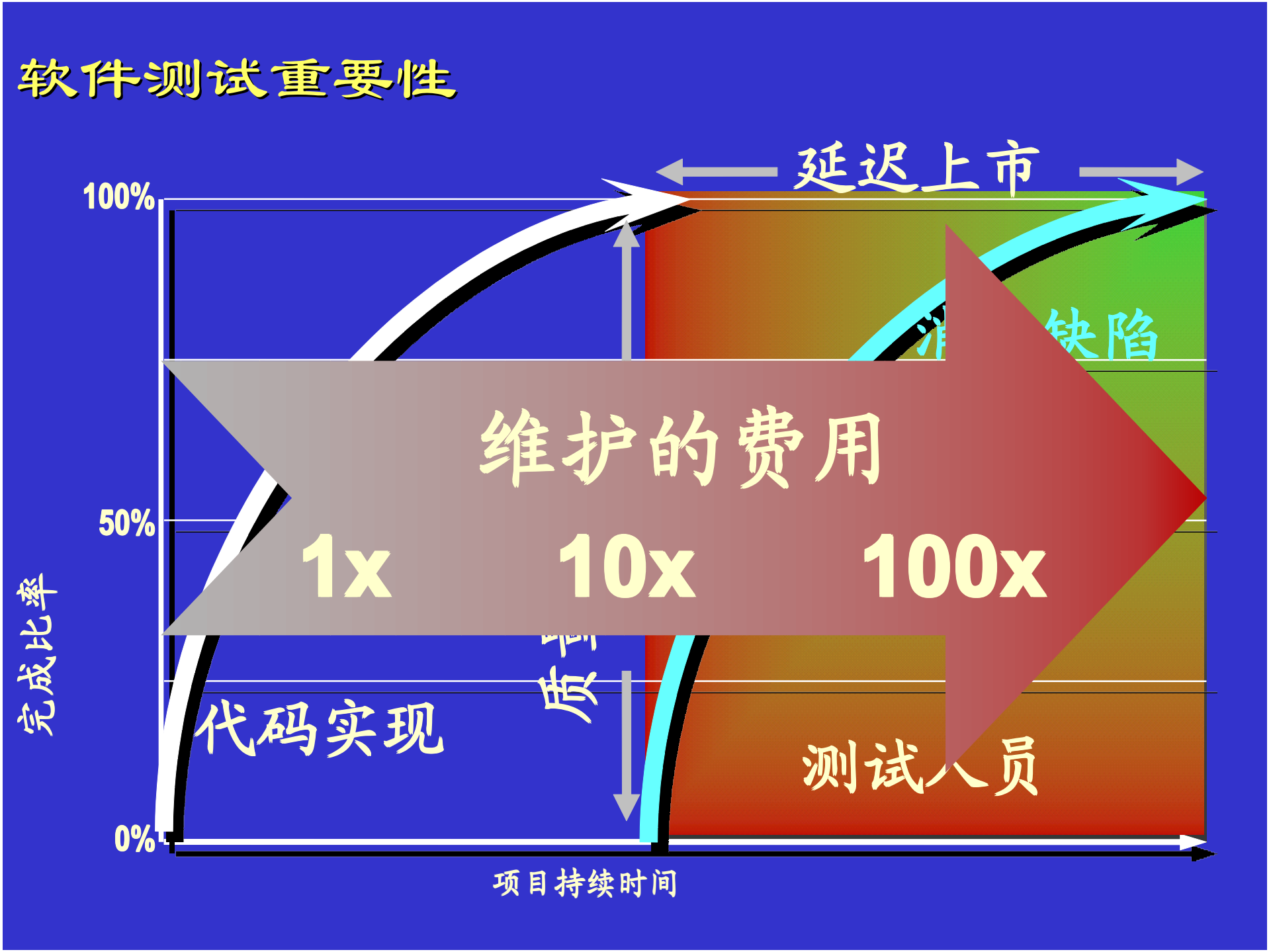
- 交流不够、交流上有误解或者根本不进行交流
- 软件复杂性
- 程序设计错误
- 需求变化
- 时间压力
- 开发人员不切实际的自负
- 代码文档贫乏
- 软件开发工具


客户如此描述需求
项目经理如此理解
分析员如此设计
程序员如此编码
商业顾问如此诠释
项目文档如此编写
安装程序如此“简洁”
客户投资如此巨大
技术支持如此肤浅
解密：
实际需求—原来如此

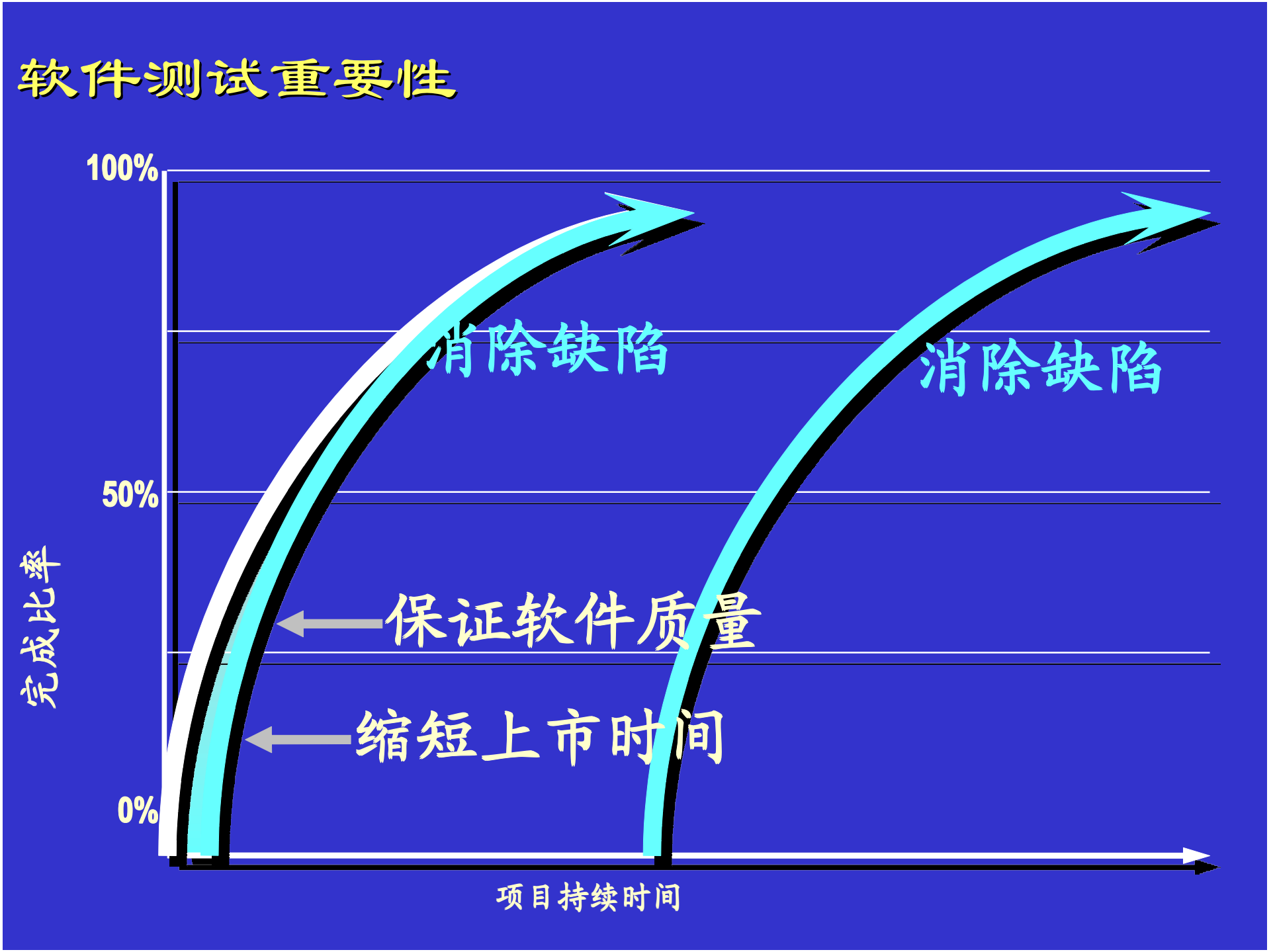
测试生命周期


开发生命周期
需求分析
设计
开发
维护
测试生命周期
测试计划
测试设计
测试开发
Build
Build
...
Revision
实施
实施
...
Exec.
缺陷分析
缺陷追踪

软件测试重要性



软件测试重要性

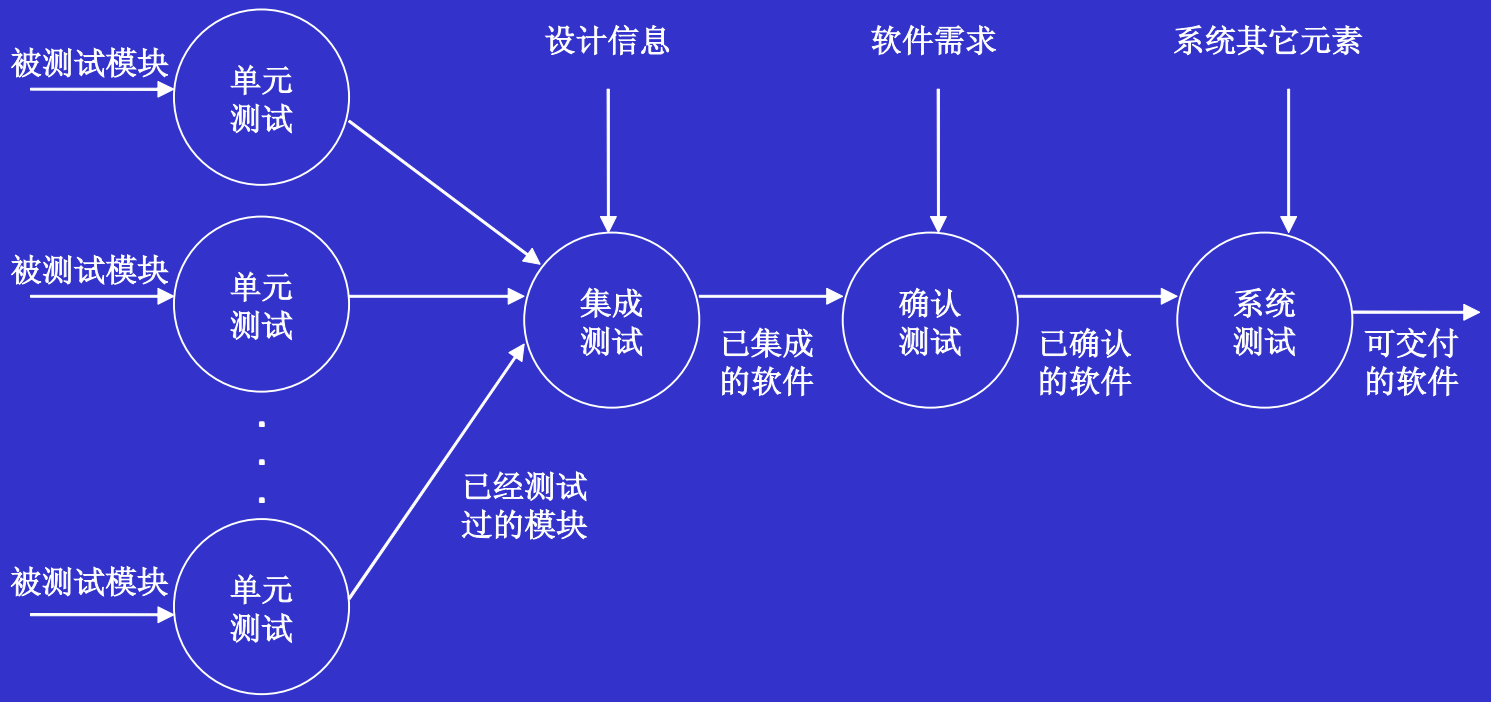


测试目的

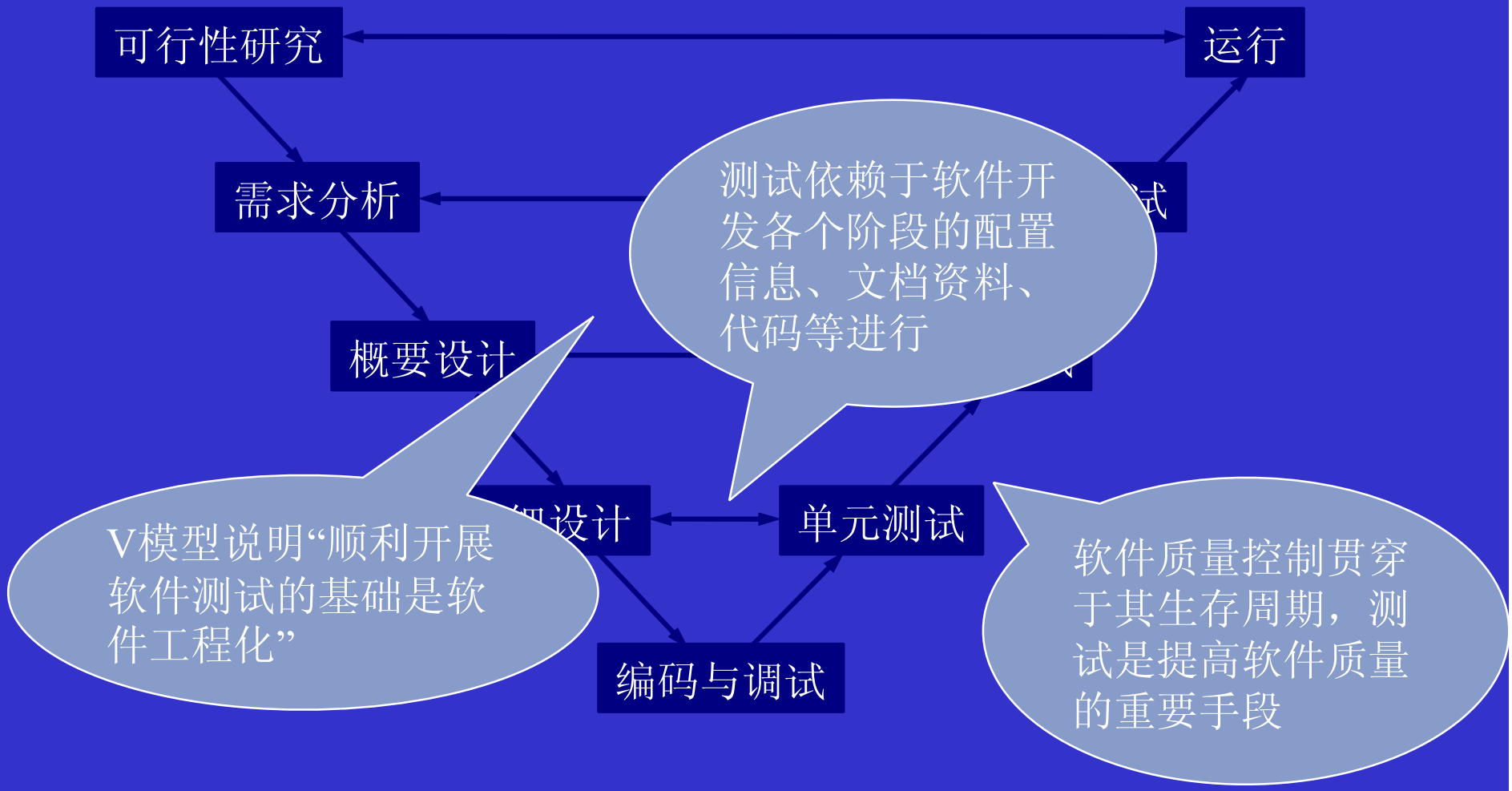
- 测试是运行程序的过程，目的在于发现错误，在用户使用应用之前发现问题。
- 测试是为了证明程序有错，而不是证明程序无错误。
- 一个好的测试用例在于能够发现至今未发现的错误。
- 一个成功的测试是发现了至今未发现的错误的测试。

测试策略

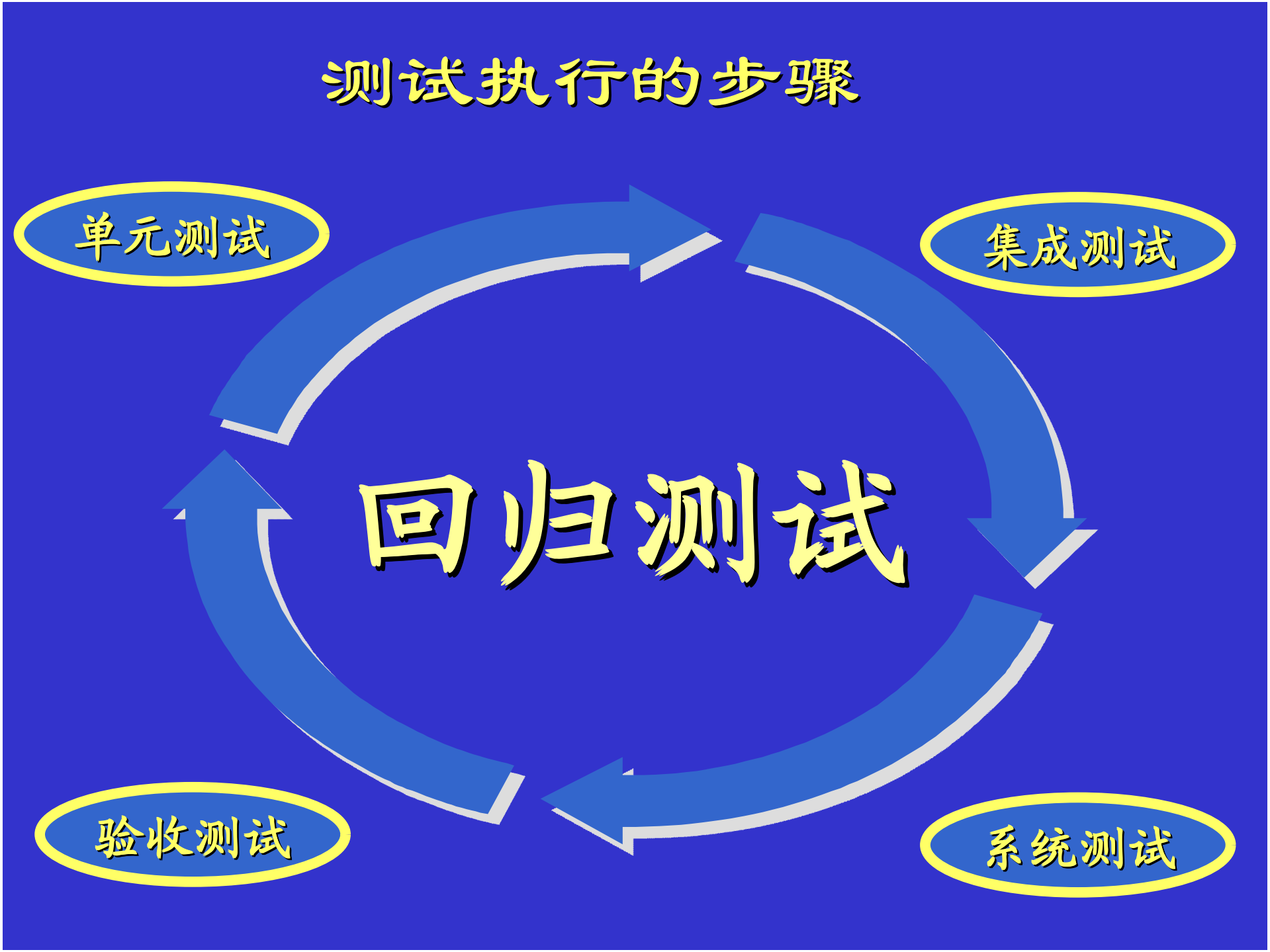
- 一般测试过程分为四个阶段



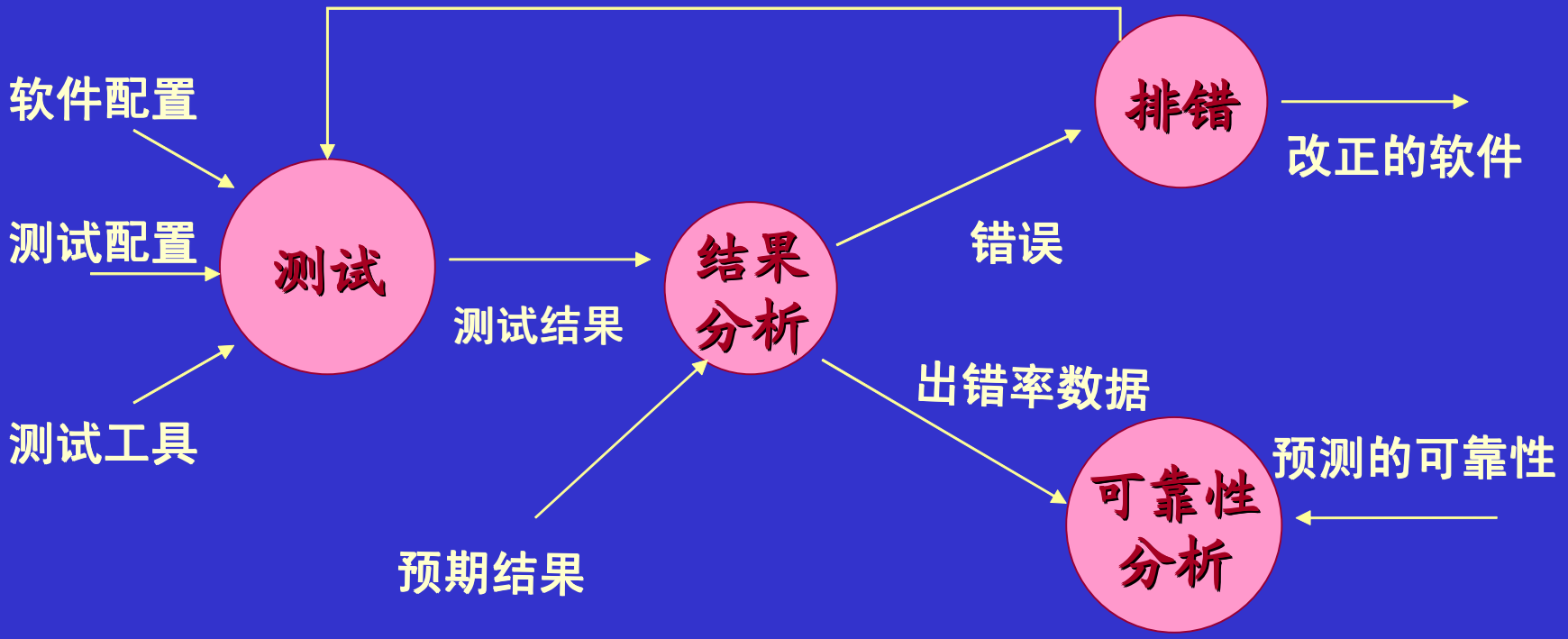
“V模型”


可行性研究
运行
需求分析
概要设计
单元测试
编码与调试
测试依赖于软件开发各个阶段的配置信息、文档资料、代码等进行
V模型说明“顺利开展软件测试的基础是软件工程化”
软件质量控制贯穿于其生存周期，测试是提高软件质量的重要手段

测试执行的步骤



测试信息流


软件配置
测试配置
测试工具
测试
测试结果
结果分析
错误
排错
改正的软件
出错率数据
预期结果
可靠性分析
预测的可靠性

软件测试方法

- 黑盒测试
  - 功能
  - 系统性能……
- 白盒测试
  - 静态质量分析
  - 覆盖测试分析
  - 代码执行性能……
- 测试执行方式
  - 动态
  - 静态

单元测试

- 单元测试常常和模块编写同步进行，在完成了模块代码的编写、通过语法编译之后就可以进行单元测试。
- 单元测试任务包括：

  模块接口测试、
  局部数据结构测试、
  边界条件测试、
  独立执行的通路测试、
  错误处理通路测试、
  资源运用测试、
  模块程序结构测试、

  ……

单元测试

- 对函数模块进行功能测试
  - 单元的功能、接口以及局部数据结构
  - 在特定的条件下，根据需求规格说明，特定的输入获得相应的输出，且有明确的测试通过准则
- 对类模块进行功能测试
  - 对类提供的方法进行功能、接口测试
  - 由于C++的特性造成面向对象程序的黑盒测试过程与结构化程序的黑盒测试过程不完全相同

单元测试内容

- 黑盒测试
  - 功能测试
- 白盒测试
  - 代码审查
    - 静态质量分析
    - 编程规则检查
  - 覆盖测试分析
    - 侧重于模块内部的语句、分支、条件的覆盖

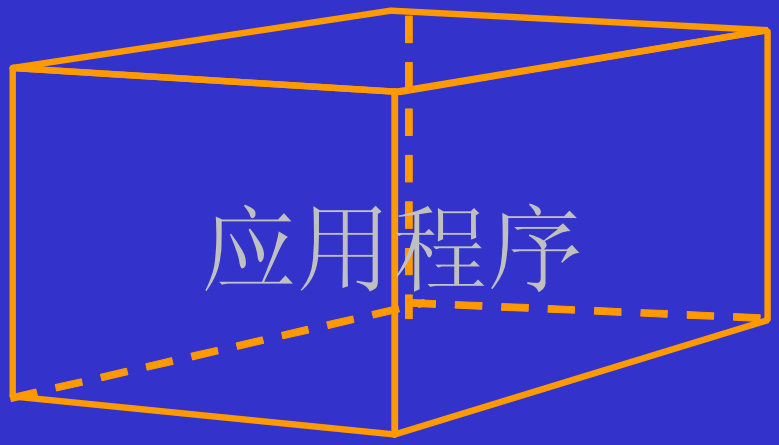
黑盒测试

输入

输出

- 黑盒测试
  - 功能测试、数据驱动测试、基于规格说明的测试
  - 将程序的执行表现与功能需求规格说明作比较

白盒测试



- 白盒测试
  - 结构测试、逻辑驱动测试
  - 结构测试将程序的执行表现与编码意图作比较
  - 随着CASE工具的发展，白盒测试的内容相应扩展

黑盒与白盒之间的关系

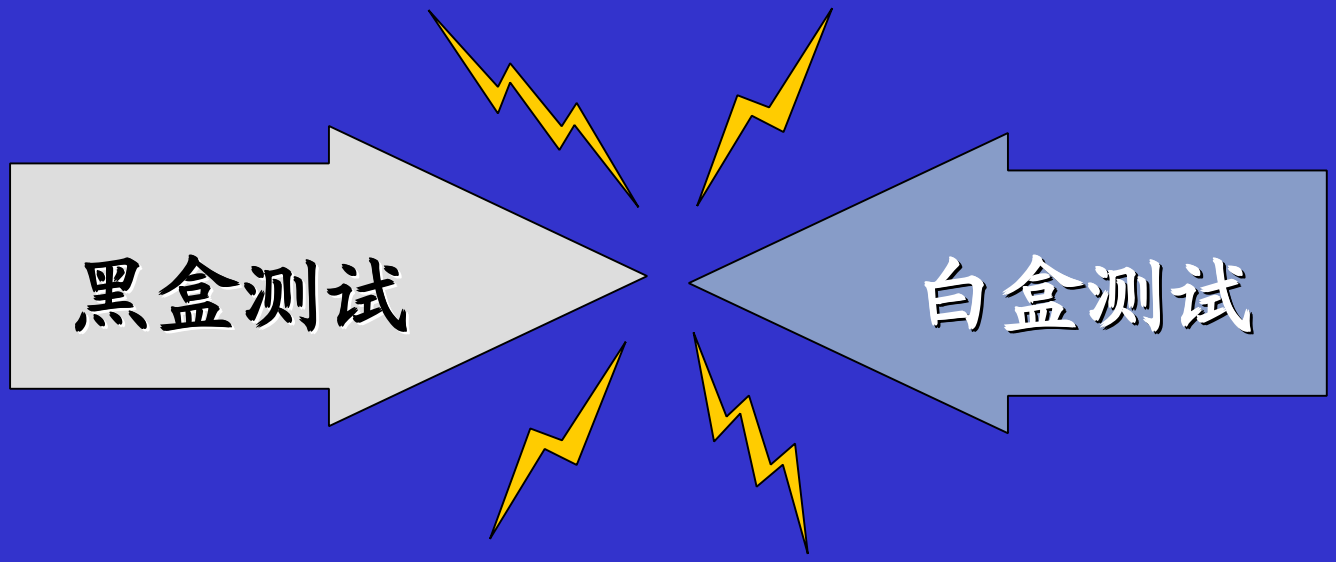
- 如果想用黑盒测试来发现程序中的所有错误，就必须用输入数据的所有可能值来检验程序是否都能产生预期的结果，但是这个显然是不可能的：

  穷举输入

  设计规格说明书

- 因而不能单纯依靠功能测试，还必须研究程序的逻辑和结构，来分析程序的总体质量状况。

黑盒与白盒之间的关系



两种测试方法从不同的角度出发，反映了软件的不同侧面

单元测试阶段的白盒测试

- 代码静态质量分析
- 编程规则检查
- 覆盖测试分析

单元测试－代码静态质量分析

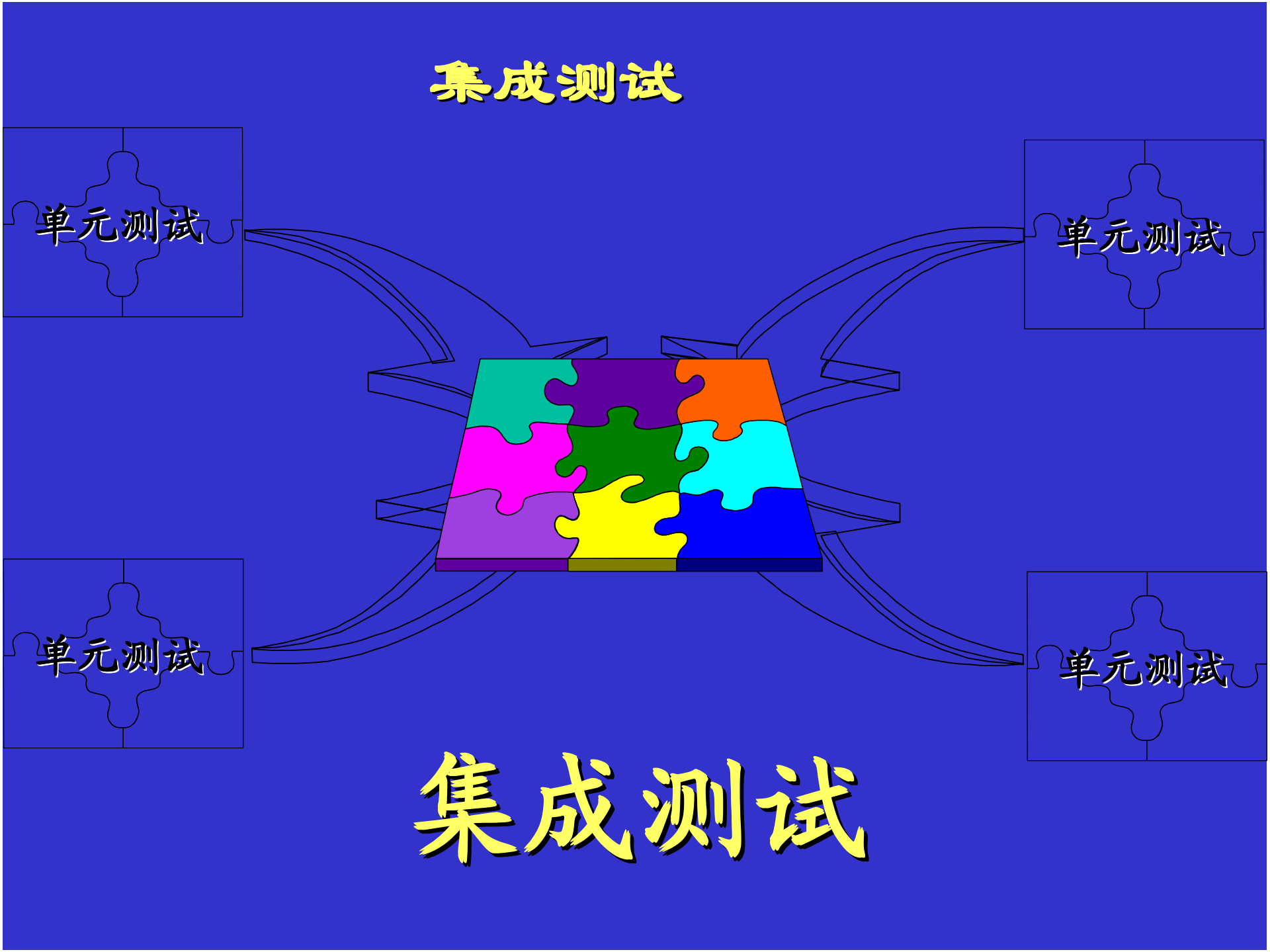
- 代码静态质量分析
  - 依据相关标准，采取度量统计的方法能够分析程序的某些质量因素
  - 通过对软件进行度量，结合适当的质量模型，可以给出具有参考价值的判断结果，尤其在软件可维护性方面

单元测试－编程规则检查

- 编程规则检查
  - 进行编程规则的检查，达到增加程序的可理解性、降低维护成本的目的
  - 编码规则
    - 针对程序指令、运算符、代码结构、声明等方面制定规则并检查，如：
    - 为了保证程序模块的结构化，规定不得使用GOTO语句
  - 命名规则
    - 对程序中局部变量、全局变量、类等的命名制定规则并检查，以利于程序的理解、维护

单元测试－覆盖测试分析

- 覆盖测试分析
  - 衡量软件被测试执行的程度
  - 在尽可能多地执行程序的路径，进行逻辑覆盖的同时，考察程序执行表现是否异常，尤其是某些复杂的和"正常"情况下不易执行的路径。
- 覆盖测试级别
  - 通常从低到高分别是：语句覆盖、判定覆盖、条件覆盖、判定/条件覆盖、条件组合覆盖等

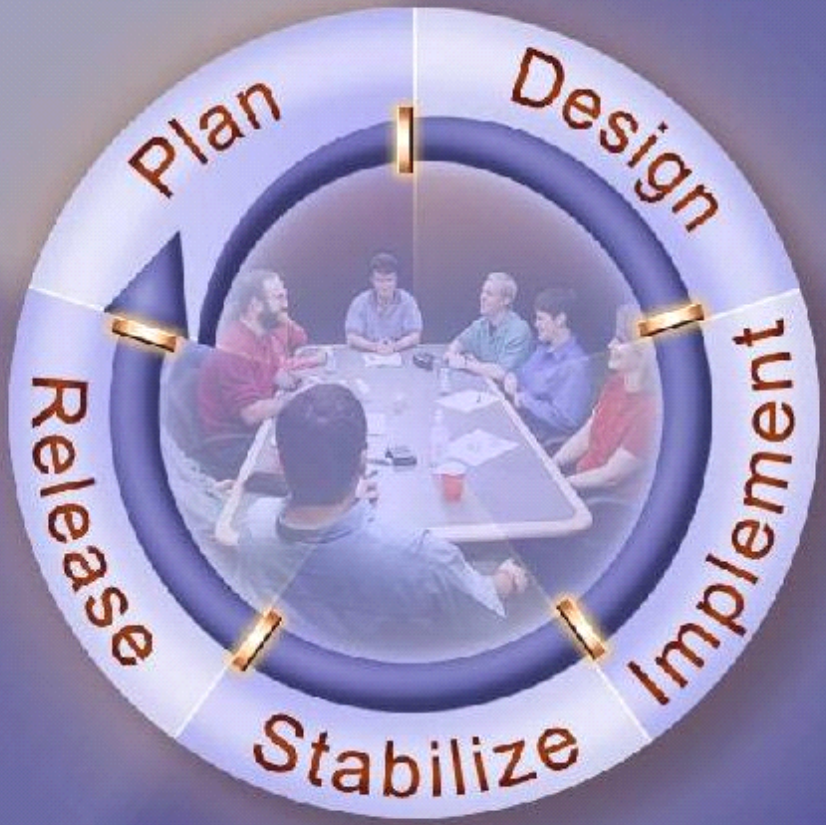

集成测试
单元测试
单元测试
单元测试
单元测试
集成测试

配置管理

产品开发周期

1、规划

2、设计



3、实施

4、稳定

5、发布

流程管理

- 利用工作流, 模板, 文件
- 好处
  - 提高效率
  - 降低成本
  - 减少人为错误
  - 有效沟通和合作
  - 有效应变
  - 白纸黑字, 有据可查
- 避免过于强调流程

编码管理

- 源代码控制
- 源代码树
- 树的分支
- Check-in and Check-out

源代码控制

- 什么是源代码控制
- 缺少源代码控制工具而引起的问题
  - 最新版本的源代码在谁的机器上？
  - 多人修改一个文件时，有些人的修改被抹掉了
  - 昨天的修改引发了新缺陷，但不知道做了哪些修改
  - 上周五的代码肯定能运行，但没法退回去了
  - 一不小心把有用的GetXYZ()函数删了并存盘了！

源代码控制工具的常用功能

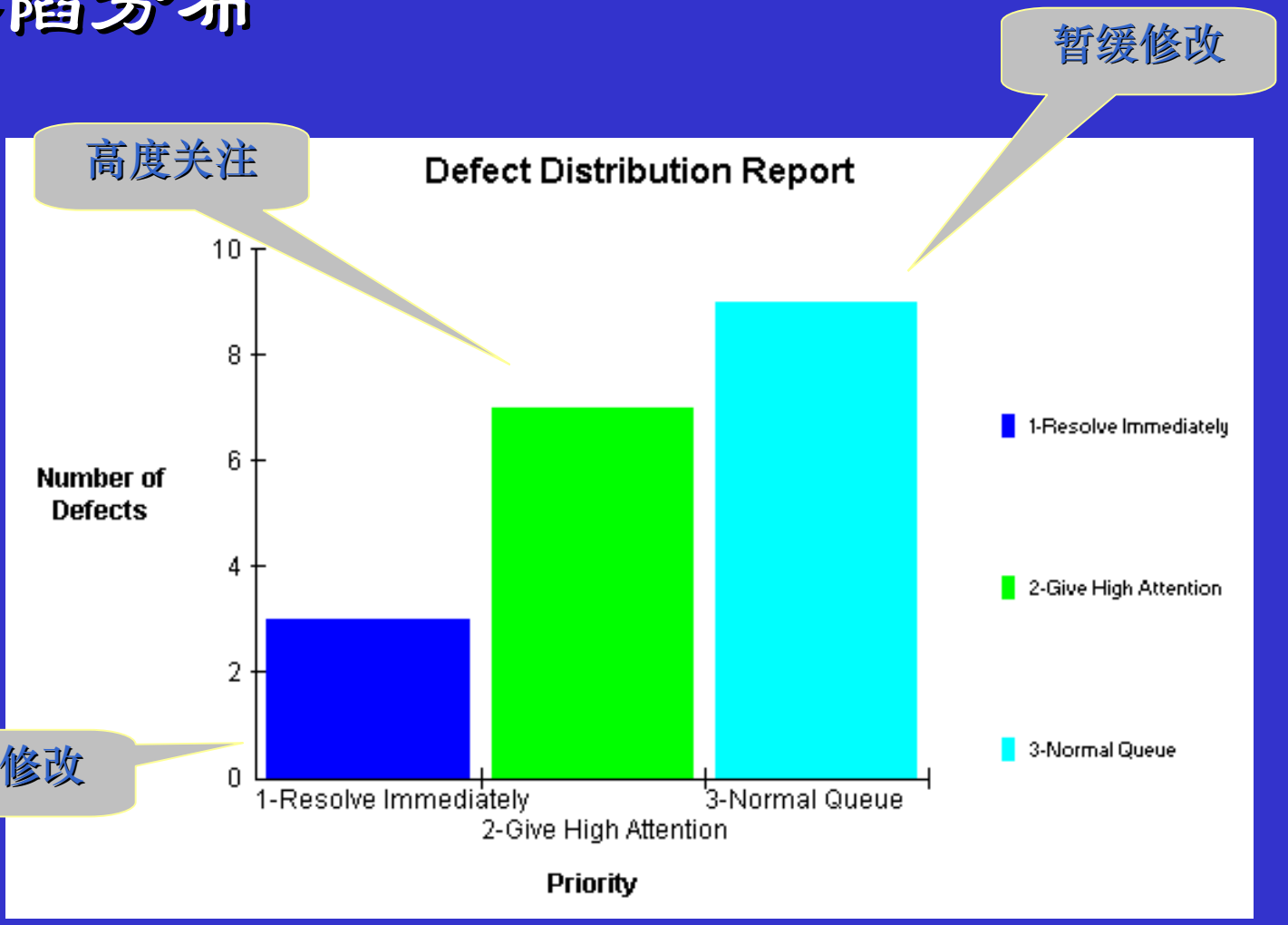
- 获得文件修改权
- 上传文件的修改
- 下载所有文件的最新版本
- 取消无效修改
- 合并
- 创建标签
- 创建分支
- 合并分支

缺陷管理

- Bug 的生命周期
- Bug 的处理流程
- Bug 的分类

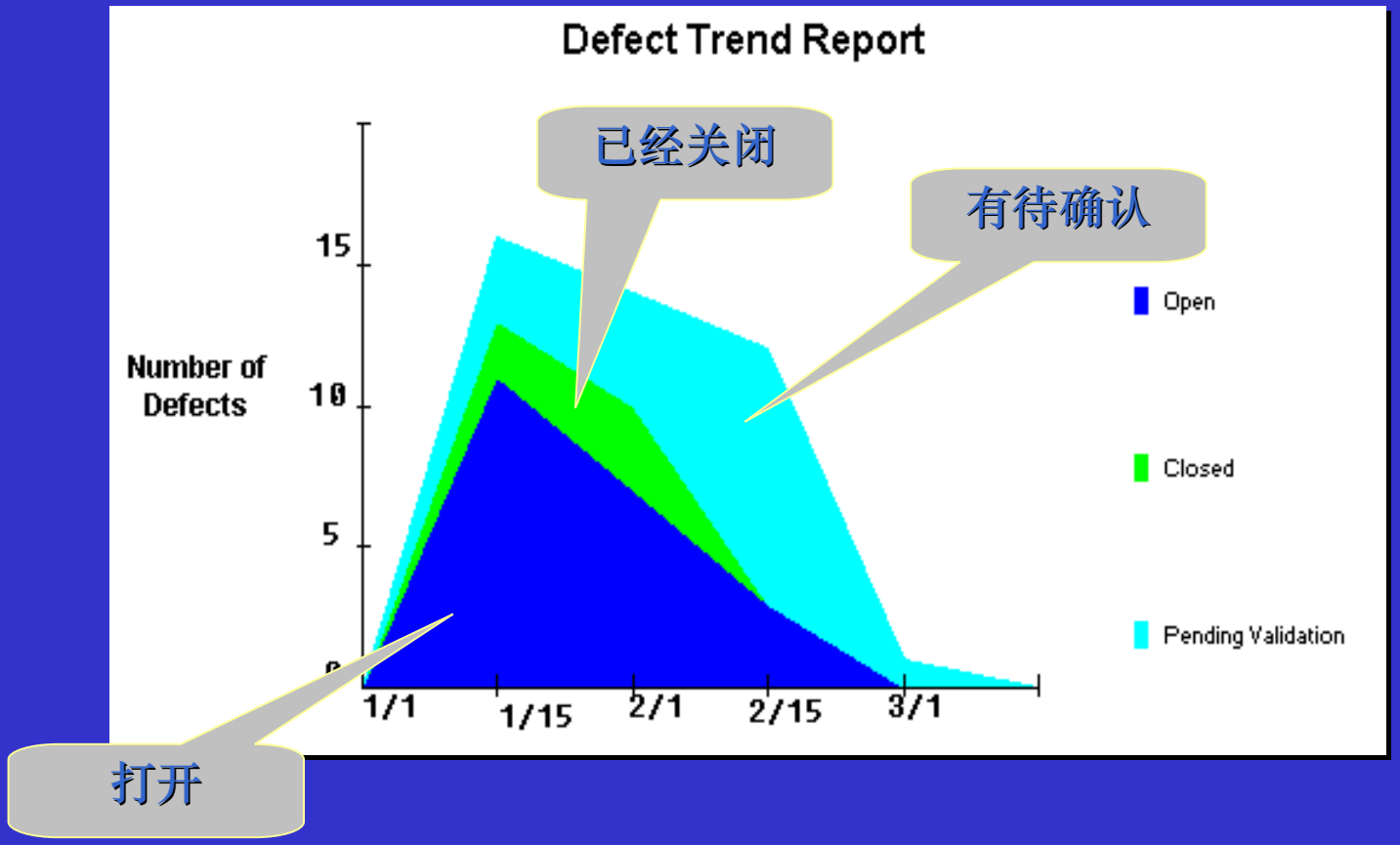
缺陷报告

- 缺陷分布


暂缓修改
高度关注
立刻修改
Defect Distribution Report
Number of Defects
10
8
6
4
2
0
1-Resolve Immediately
2-Give High Attention
3-Normal Queue
Priority
1-Resolve Immediately
2-Give High Attention
3-Normal Queue

缺陷报告

- 缺陷趋势


Defect Trend Report
已经关闭
有待确认
Number of Defects
15
10
5
1/1
1/15
2/1
2/15
3/1
Open
Closed
Pending Validation
打开

缺陷统计

序号	ID	模块	用例号	用例名称	类别	状态	子状态	优先级	描述	责任人	测试人	发现日期	通知日期	解决日期	可重复
1	177215	产品框架	PCA801	维护控制...	与需求不符	活跃	无	中	修改时利润中心货币不能输入	常阳	王立中	2009-04-0...	2009-04-0...		总能出现
2	177214	产品框架	PCA801	维护控制...	与需求不符	活跃	无	中	输入不存在的成本中心没按要...	常阳	王立中	2009-04-0...	2009-04-0...		总能出现
3	177213	产品框架	PCA801	维护控制...	与需求不符	活跃	无	中	输入存在的标准层次检验没通过	常阳	王立中	2009-04-0...	2009-04-0...		总能出现
4	177212	产品框架	PCA801	维护控制...	界面不规范	活跃	无	中	标准层次没搜索帮助	常阳	王立中	2009-04-0...	2009-04-0...		总能出现
5	177211	产品框架	PCA801	维护控制...	与需求不符	活跃	无	中	没有把控制范围里的利润中心...	常阳	王立中	2009-04-0...	2009-04-0...		总能出现
6	177207	产品框架	SYS109	维护允许...	界面不规范	已经处理	无	中	如图05所示,按照规范把表格...	刘丹	蔡文涛	2009-04-0...	2009-04-0...	2009-04-0...	总能出现
7	177150	产品框架	SYS109	维护允许...	建议修改	活跃	无	高	LockForbidUnLock该表应该加...	刘丹	蔡文涛	2009-04-0...	2009-04-0...		总能出现
8	176993	产品框架	PCA801	维护控制...	界面不规范	活跃	无	中	界面控件排列不符合开发规范	常阳	王立中	2009-03-3...	2009-03-3...		总能出现
9	176867	产品框架	MON602	锁监控	校验不完整	已经处理	无	中	12:7001环境用SUPER用户进入...	刘丹	王洋	2009-03-3...	2009-03-3...	2009-04-0...	总能出现
10	176886	产品框架	PCA804	调整行项...	界面不规范	活跃	无	中	计划版本搜索帮助不好使.	常阳	张欣	2009-03-3...	2009-03-3...		总能出现
11	176854	产品框架	PCA412	利润中心...	影响系统运行	活跃	无	中	执行数据如图出现异常.	刘彪	张欣	2009-03-2...	2009-03-2...		总能出现
12	176848	产品框架	PUBLIC	导入excel	校验不完整	已经处理	无	中	导入提示没有数据，客户端出...	刘丹	刘盖轲	2009-03-2...	2009-03-2...	2009-04-0...	总能出现
13	176602	产品框架	WFL901	与OA用户...	校验不完整	活跃	无	中	点击定位按钮，程序没有反应...	李佳	矫立丽	2009-03-2...	2009-03-2...		总能出现
14	176134	产品框架	PUBLIC	表格设置	校验不完整	已经处理	无	中	INV612：第一次设置表格显示...	王笑寒	矫立丽	2009-03-1...	2009-03-1...	2009-03-2...	总能出现
15	176025	产品框架	PCA864	维护凭证类型	校验不完整	活跃	无	中	执行用例出现异常.	林贵泉	张欣	2009-03-1...	2009-03-1...		总能出现
16	175904	产品框架	PCA801	维护控制...	消息不规范	已经处理	无	中	12:9001第一屏输入FO1出现如...	常阳	张艳萍	2009-03-0...	2009-03-0...	2009-03-2...	总能出现
17	175656	产品框架	REM110	维护日产...	校验不完整	活跃	无	中	12:7001环境下执行REM110，...	柳欣	孙会生	2009-03-0...	2009-03-0...		总能出现
18	173655	产品框架	REM412	查询上线...	与需求不符	活跃	无	中	12:7001环境，执行REM412，...	孙雨	孙会生	2009-01-1...	2009-01-1...		总能出现

02周ERP系统缺陷模块汇总表

模块		合计	物料管理	生产管理	销售管理	财务会计	成本管理	人力资源管理	质量管理	报表管理	工作流管理	底层架构
在处理缺陷	合计	113	19	51	8	14	11	0	0	0	0	10
	活跃缺陷	55	13	17	3	9	4	0	0	0	0	9
	已经处理	58	6	34	5	5	7	0	0	0	0	1
关闭缺陷		50160	13072	8511	6782	10836	6905	1384	1187	302	110	1071
缺陷总数		50273	13091	8562	6790	10850	6916	1384	1187	302	110	1081

版本控制

- 程序员需要在每个版本上加上注释：所谓的版本注释，就是该版本的相对于前一版本的改变和在程序中的体现，以及对系统的影响有哪些，每个版本的的区分不是代码改变多少而是功能的改变。

沟通管理

- 电邮
- 会议
- 虚拟团队

测试团队管理

- 团队组建
- 团队的周期
- 几种模式
  - 根据产品
  - 根据工作类别
- 不断更新以适应新环境
  - 服务组
  - 虚拟组(virtual team)

开发制胜策略

主要内容

- 程序员13条制胜法则
  - 编写优质代码——四大法则
  - 测试代码，安身立命之本——四大法则
  - 千锤百炼，不败金身——五大法则

编写优质代码

- 统一代码风格
- 避免冗长代码
- 降低代码间耦合
- 减少冗余代码

法则1：统一代码风格

- 确定统一的编码风格
- 添加注释
- 起个好名字
- 让一切井井有条

代码风格一致（续）

- 代码易读，易理解
  - 每个开发人员可以读懂其他人的代码
- 产品中所有的代码需遵循统一的标准
  - 指定统一编码风格文档
  - 重要的在于让每个开发人员都遵守
  - 将不符合规范的代码当作错误处理

添加注释

- 添加注释的目的
  - 使代码易读、易写、易维护
- 如何添加注释
  - 代码、数据、算法的解释
  - 做标记（时间、所做的改动等）
  - 标识代码的功能和目的
  - 代码如何调用
- 避免
  - 对显而易见的内容进行注释
  - 添加大段注释
  - 注释的拼写错误

起个好名字

- 大小写问题
  - Pascal Case：BackColor
  - Camel Case：backColor
  - Upper Case：System.Web.UI
- 避免混淆

  int myNumber;

  int MyNumber;
- 使用缩写准则
  - GetWindow vs. GetWin
  - 首字母缩写
    - 避免使用不被广泛接受的首字母缩写
  - 词语的选择

起个好名字（续）

- 名字空间、类、接口、方法、参数的命名
  - 名字空间：
    CompanyName.TechnologyName[.Feature][.Design]
  - 类：使用名词、不要使用前缀、不要使用“_”
    FileStream、ApplicationException
  - 接口：使用名词或形容词、加前缀“I”、不要使用“_”
    IServiceProvider、IFormatable
  - 方法：使用动词
    RemoveAll()、GetCharArray()
  - 参数：camel Case、“见文知义”
    Type GetType(string **typeName**)

让一切井井有条

- 类中各个方法的次序
  - 根据public, protected, private分组
  - 每组中再根据各方法直接关系细分
    - openConnection()
    - sendMessage()
    - closeConnection()
  - 其他以首字母为序
  - VS.NET中使用#region / #endregion进行分组

法则2： 避免冗长代码

- 一个method多长合适？
  - 曾经看过1400行的一个method
  - 1986年对IBM OS/360的统计结果：绝大多数的错误出在大于500行的函数中
  - 1991年对148,000行代码的统计结果，在大于143行的函数中修复一个Bug需要多花费2.4倍的代价
- 专家的建议
  - 不要超过一屏

法则3： 降低代码间耦合

- 耦合无处不在
  - 整体系统的构成依靠耦合
  - 松耦合和紧耦合
- 避免紧耦合
  - 修改一处代码不会引起更多的变动
  - 便于代码重用、维护和扩充
- 类之间的耦合
  - Identity耦合，Representational耦合

法则4： 减少冗余代码

- 千年虫问题
  - 简单问题不简单
- 如何减少冗余
  - 避免Hard Coding
    - 使用Config文件
    - 使用常量、枚举类型
  - 使用抽象类
  - 重构
  - 重用代码

小结：编写优质代码

- 统一代码风格
- 避免冗长代码
- 降低代码间耦合
- 减少冗余代码

测试代码，安家立命之本

- Debug vs. Trace
- 活用断言（Assert）
- 异常处理
- 编写Unit Test

法则5：调试与跟踪

- Debug vs. Release
- Debug vs. Trace
  - System.Diagnostics名字空间包含了Debug类和Trace类
  - 建议使用Debug.Assert和Trace.Write
- 在.NET中使用Trace.Listener
  - Demo
- 在.NET中使用Trace.Switch
  - Demo

法则6：活用断言（Assert）

- 不要有任何假设
  - Defensive Programming
- 大多数错误出在代码之间的调用上
- 使用断言
  - 调用参数的检验
  - 返回值的检验
  - 数据有效性检验
  - 算法有效性检验

调用参数的验证

```
public bool FindCustomer (int customerID)
{
   Debug.Assert (customerID > 0,
   "FindCustomer出错","customerID应该大于0");
   // More code...
   return true;
}
```

算法验证

- 使用Debug代码来验证代码的正确性
- 方法
  - 对初始状态、中间状态和最终结果使用断言
  - 使用更简单的算法对程序结果进行确认
- 举例
  - 排序算法中，排序结果的每一个值都大于等于前一个值
  - 压缩算法中，压缩文件解压缩后和原文件匹配
  - 加密算法中，密文解密的结果应该等于原文
- Demo

法则7：异常处理

- 异常处理和Debug代码的关系
- 异常处理和返回值的关系
- 编写稳定的应用程序的关键
  - 增强了程序的可扩展性

    A→B 扩展到A→I→B
  - 错误的处理更加灵活

    A→I→J→K→B （A可以灵活的处理由B引起的错误）
  - 异常不容易被忽视

抛出异常

- 根据method的功能，在需要的地方抛出异常
- 所有的自定义异常类以Exception结尾
- 自定义异常继承System.ApplicationException，不要继承System.Exception

```
public class FileNotFoundException:ApplicationException
{
}
```

- 提供详细的异常信息（Exception Message）
- 不要在异常信息中提供敏感的安全内容
- 抛出最为合适的异常

处理异常

- 捕获特定的异常
- 不要把异常“吃掉”

```
public void Method(){
    try{
      File.Open(…);
    }
    catch(Exception e){ //所有的异常都被“吃掉”了…
    }
}
```

- 使用finally释放资源

```
public void UpdateSet(){
    FileStream stream = null;
    try{
      stream = new FileStream(“SomeFile.dat”, FileMode.Open);
    }finally{
      if(stream != null) stream.Close();
    }
}
```

处理异常（续）

- 恢复原来的状态

```
public void DoSomething(FileStream fs){
    Int64 pos = fs.Position;
    try{
      //do some reading with fs
    } catch{
      fs.Position = pos;  // unwind on the failure
      throw;              // rethrow
    }
}
```

- 添加更加详细的信息

```
public Int32 GetInt (Int32[ ] array, Int32 index){
    try{
      return array[index];
    }catch(IndexOutOfRangeException e){
      throw new ArgumentOutOfRangeException(
                  "Parameter index is out of range.");
    }
}
```

法则8：编写Unit Test

- 提倡先写Unit Test后写代码
  - 测试先行
  - 确保Unit Test不会因为进度压力而取消
  - 先定功能，再定细节
  - 不是强制准则
- 如何写Unit Test
  - 微软内部工具：TTest
  - 其他的.NET Unite Test工具：NUnit， dotNetUnit，HarnessIt，csUnit
- Demo

小结：测试代码,安家立命之本

- 测试与跟踪
- 活用断言（Assert）
- 异常处理
- 编写Unit Test

千锤百炼，不败金身

- Verifier，一个都不放过
- 安全至上
- 性能与功能并进
- 代码审核（Code Review)
- 及时修复 Bug

法则9： 使用Verifier工具——一个都不放过

- 将警告等级设置为最高
- 利用一个有效的Verifier工具自动检验代码
- Demo

法则10：安全至上

- 了解黑客如何攻击产品
- 安全性是微软产品的最大挑战
- 让你的产品防御攻击
  - 将安全性看作需要实现的功能之一
  - 使用安全攻击模型
  - 避免常犯的错误
- Demo
  - Login User Control

法则11：性能与功能并进

- 将性能作为需要实现的功能之一
- 进行性能测试
- Demo

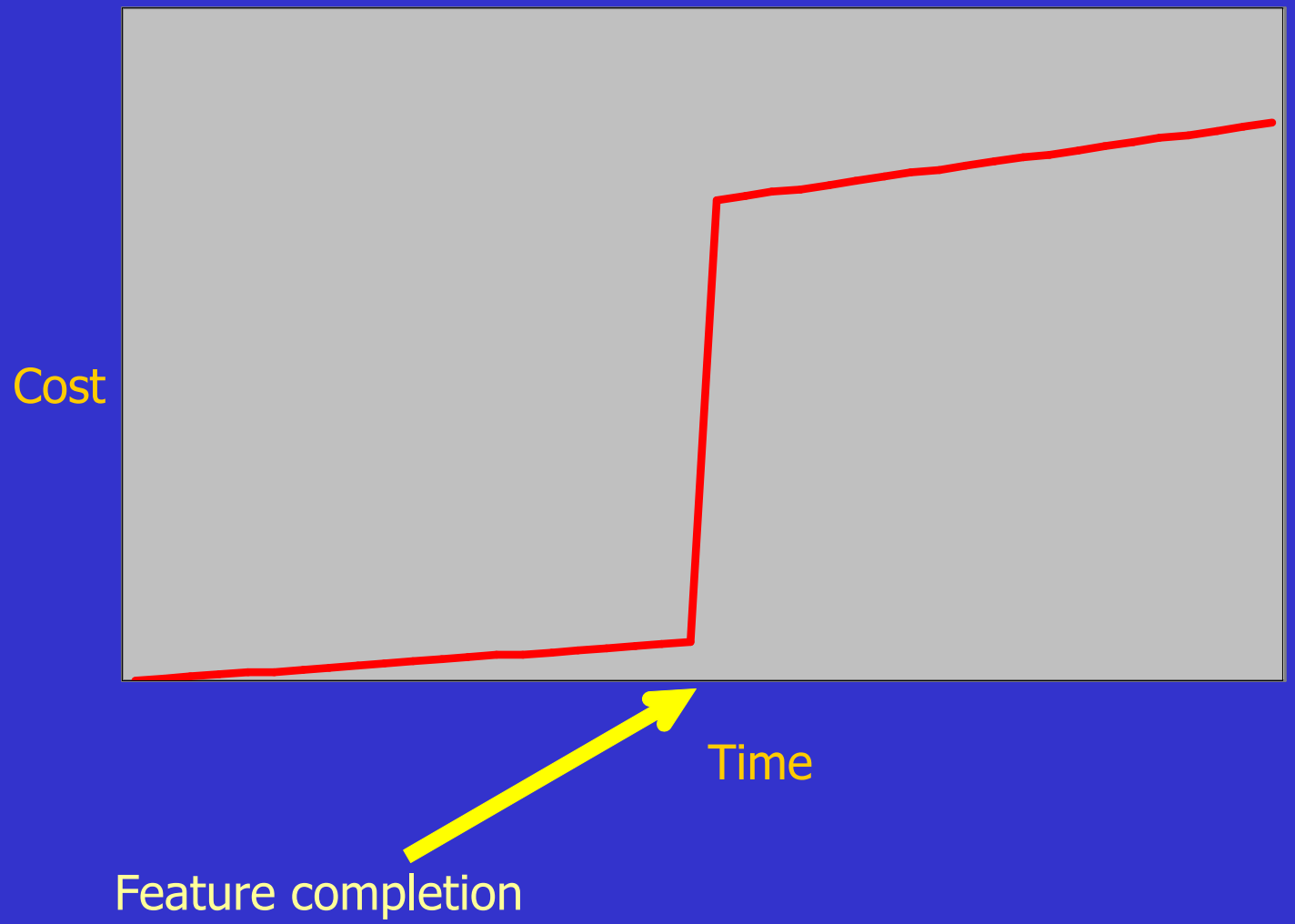
法则12：代码审查

- 打印代码，阅读，做各种标记
- 使用各种diff工具查看代码的改动
- 伙伴代码审核，小组代码审核

法则13：及时修复Bug

- Bug数量激增
  - 影响产品质量
  - 团队态度
  - 能否按时发布产品？
- 设置各个准则控制Bug数量
  - 控制每个开发人员的Bug
  - 控制一个Bug处于“激活”状态的时间
  - 根据优先级/严重级
  - “Bug hell”日

修复一个Bug的代价



修复Bug流程

- 重现Bug
- 理解问题所在
- 是否修复该Bug
- 仔细修复—将Bug修复当作功能实现
- 其他地方是否有同样Bug？
- 该Bug是否能被预防？
- Check-in时记录Bug编号
- 在Bug数据库中将其状态改为修复

小结：千锤百炼，不败金身

- Verifier，一个都不放过
- 安全至上
- 性能与功能并进
- 代码审核（Code Review)
- 及时修复 Bug

百尺竿头，更进一步

- 编写优质代码
- 发布优质产品
  - 发扬主人翁精神
  - 让人信任：按时发布产品，保持良好沟通
- 帮助整个团队
  - 提供各种资源
  - 与他人愉快合作
- 跟踪最新技术，修炼与提高