Southeast University

软件工程导论

廖力

lliao@seu.edu.cn

课程结构

Unit1. 软件工程概述

Unit2. 软件工程技术

一、系统工程

二、需求工程

三、设计工程

四、软件构建与测试

（一）软件实现

（二）软件测试概述

（三）软件测试策略

（四）软件测试技术

Unit3. 软件项目管理

（一）软件实现

- 1. 程序设计语言的选择
- 1）选择编程语言时，通常要考虑如下因素：
  - 系统的应用领域
  - 用户的要求
  - 软件的执行环境
  - 目标系统的性能要求
  - 程序员的知识水平
  - 软件的可移植性要求

（一）软件实现

- 1. 程序设计语言的选择
- 2）项目所属的应用领域常常是首要的标准
  - FORTRAN适用于工程和科学计算领域
  - Prolog、Lisp适用于人工智能领域
  - Java、C++适用于OO系统的开发
  - 有些语言适用于多个应用领域，如C
- 3）若有多种语言都适合于某项目的开发时，也可考虑选择开发人员比较熟悉的语言

（一）软件实现

- 1. 程序设计语言的选择
- 4）选择高级语言还是低级语言
  - 优先选择高级语言（开发维护易）

    开发和维护高级语言程序比开发和维护低级语言程序容易得多
  - 必要时使用低级语言（功效高）

    高级语言程序经编译后所产生的目标程序的功效要要低于完成相同功能的低级语言程序

（一）软件实现

- 1. 程序设计语言的选择
- 5）使用低级语言的情况：
  - 对运行时间和存储空间有很高要求的项目
  - 在某些不能提供高级语言编译程序的计算机上开发程序，如单片机上的软件
  - 大型系统中对系统执行时间起关键作用的模块

（一）软件实现

- 2.程序设计风格——源程序文档化
- 源程序文档化主要包括：标识符的命名，注解，程序的视觉组织
- 1）标识符的命名
  - 选择含义明确的名字，使其能正确提示标识符所代表的实体
    - 例如，Total，Average等
  - 名字不要太长，太长会增加打字量，且易出错。必要时可使用缩写

（一）软件实现

- 2. 程序设计风格——源程序文档化
- 1）标识符的命名
  - 不用相似的名字，相似的名字容易混淆，不易发现错误
    - 如cm, cn, cmn, cnm, cnn, cmm
  - 不用关键字作标识符
  - 同一个名字不要有多个含义
  - 名字中避免使用易混淆的字符。
    - 如数字0与字母O；数字1与字母I或l；数字2与字母z等

```
bool IsCorrect(int y,int m,int d)
{
        if(y<1)
                return false;
        else
        {

        if(((y%4==0)&&(y%100!
=0))||(y%400==0))

        maxA[1]=29;
                if(m<0||m>12)
```


```
public static int
getWeekday(int year, int
month, int day)

int totalDays = 0;
```

（一）软件实现

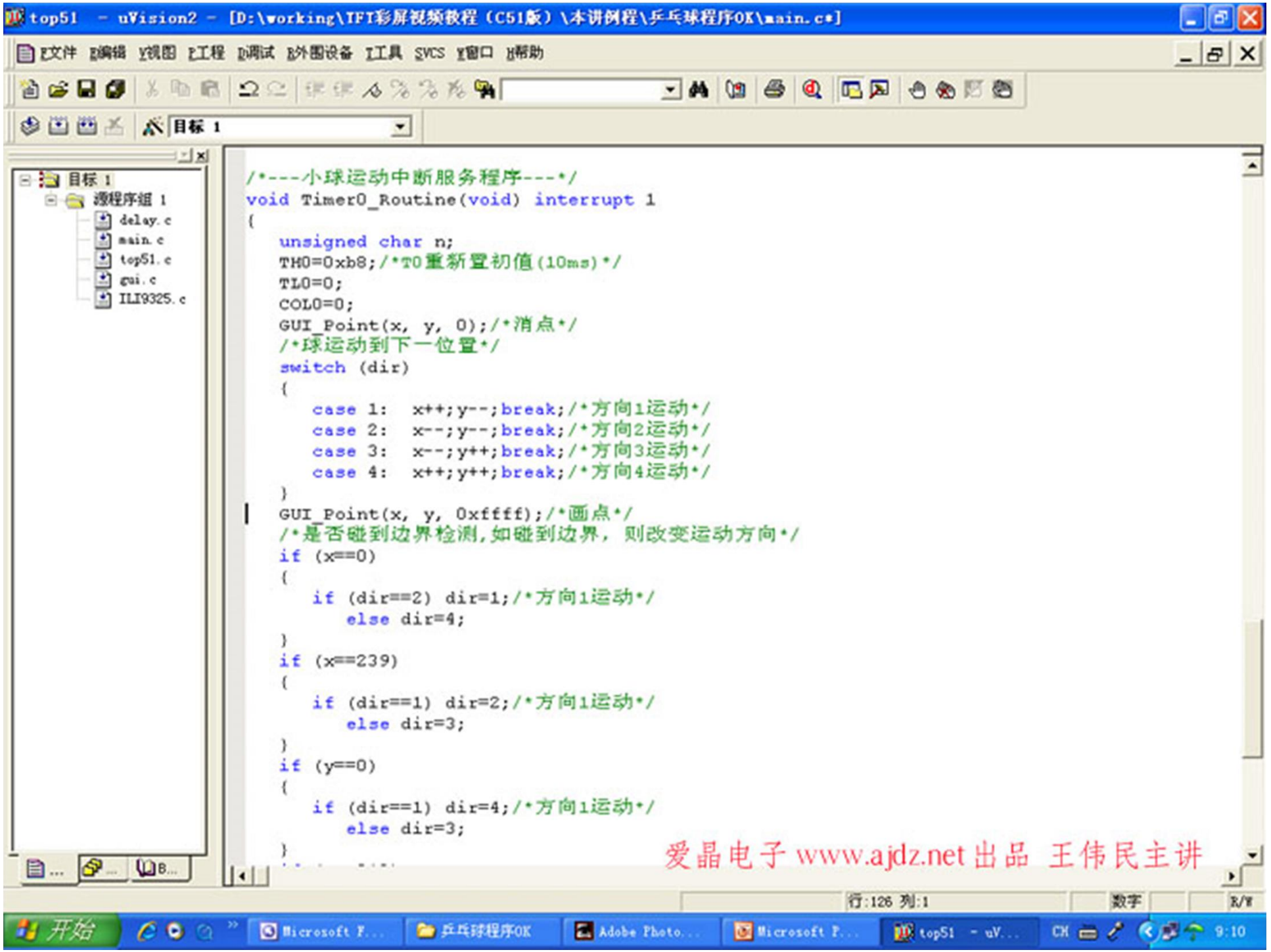
- 2. 程序设计风格——源程序文档化
- 2）程序的注释
  - 程序中的注解用来帮助人们理解程序，决不是可有可无的
  - 一些正规的程序文本中，注解行的数量约占整个源程序的1／3到1／2，甚至更多
  - 注解分为序言性注解和功能性注解

（一）软件实现

- 2. 程序设计风格——源程序文档化
- 2）程序的注释
  - 序言性注解：通常置于每个程序模块的开头部分，主要描述：
    - 模块的功能
    - 模块的接口：包括调用格式、参数的解释、该模块需要调用的其它子模块名
    - 重要的局部变量：包括用途、约束和限制条件
    - 开发历史：包括模块的设计者、评审者、评审日期、修改日期以及对修改的描述

（一）软件实现

- 2. 程序设计风格——源程序文档化
- 2）程序的注释 （A，B）
  - 功能性注解：通常嵌在源程序体内，主要描述程序段的功能。书写时应注意：
    - 注解要正确，错误的注解比没有注解更糟；
    - 为程序段作注解，而不是为每一个语句作注解；
    - 用缩进和空行，使程序与注释容易区分；
    - 注解应提供一些从程序本身难以得到的信息，而不是语句的重复。


top51 - uVision2 - [D:\working\TFT彩屏视频教程（C51版）\本讲例程\乒乓球程序OK\main.c*]
文件 编辑 视图 工程 调试 外围设备 工具 SVCS 窗口 帮助
目标 1
目标 1
源程序组 1
delay.c
main.c
top51.c
gui.c
ILI9325.c
/*---小球运动中断服务程序---*/
void Timer0_Routine(void) interrupt 1
{
unsigned char n;
TH0=0xb8;/*T0重新置初值(10ms)*/
TL0=0;
COL0=0;
GUI_Point(x, y, 0);/*消点*/
/*球运动到下一位置*/
switch (dir)
{
case 1: x++;y--;break;/*方向1运动*/
case 2: x--;y--;break;/*方向2运动*/
case 3: x--;y++;break;/*方向3运动*/
case 4: x++;y++;break;/*方向4运动*/
}
GUI_Point(x, y, 0xffff);/*画点*/
/*是否碰到边界检测,如碰到边界，则改变运动方向*/
if (x==0)
{
if (dir==2) dir=1;/*方向1运动*/
else dir=4;
}
if (x==239)
{
if (dir==1) dir=2;/*方向1运动*/
else dir=3;
}
if (y==0)
{
if (dir==1) dir=4;/*方向1运动*/
else dir=3;
}
爱晶电子 www.ajdz.net 出品 王伟民主讲
行:126 列:1
数字
开始
Microsoft P...
乒乓球程序OK
Adobe Photo...
Microsoft P...
top51 - uV...
CH
9:10

（一）软件实现

- 2. 程序设计风格——源程序文档化
- 3）程序的视觉组织
  - 程序中代码的布局对于程序的可读性也有很大影响。
  - 适当的利用空格、空行和移行能使程序的逻辑结构更加清晰。
  - 空格的合理应用还可以突出运算的优先性，避免发生运算的错误。

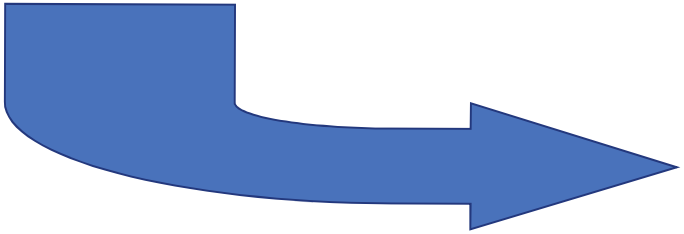
（一）软件实现

- **3.** 程序设计风格——数据说明
- 为了使程序中数据说明更易于理解和维护，可遵循以下规则：
  - 数据说明次序规范化，使数据属性容易查找，也有利于测试，排错和维护
    - 如：常量说明→简单变量说明→ 数组说明
  - 说明语句中变量安排有序化
    - 如：当一个说明语句中有多个变量名时，可以将这些变量按字母的顺序排列，以便于查找
  - 使用注解说明复杂数据结构

（一）软件实现

- **4.** 程序设计风格——语句构造
- 语句构造应遵循的原则是：每条语句应该简单而直接，不应为了片面追求效率而使代码变得过于复杂。

```
A[I] = A[I]＋A[T];
A[T] = A[I]－A[T];
A[I] = A[I]－A[T];
```



```
WORK = A[T];
A[T] = A[I];
A[I] = WORK;
```

（一）软件实现

4. 程序设计风格——语句构造

- 人们在长期的实践中总结了以下一些规则：
  - 不要为了节省空间而把多个语句写在同一行；
  - 用空格或可读的符号使语句的内容更加清晰；
  - 尽量避免太复杂的条件表达式；
  - 避免过多使用循环嵌套和条件嵌套；
  - 利用括号使逻辑表达式或算术表达式的运算次序清晰直观；
  - 尽可能使用库函数；

（一）软件实现

- 5. 程序设计风格——效率
- 1）通常，效率主要指占用处理机时间和主存区域两个方面。
- 2）好的编码可以提高效率，但是要注意三条原则：
  - 效率是一个性能要求，应在需求分析阶段就确定；
  - 通过好的设计可以提高效率；
  - 不该为了提高代码效率而牺牲程序的清晰性和可读性。

（一）软件实现

- **5.** 程序设计风格——效率
- **3**）如何提高代码效率
  - 在编码之前，先化简算术表达式和逻辑表达式；
  - 特别注意嵌套的循环，以确定是否有语句可以从循环内层移到循环外层；
  - 尽量避免使用多维数组和复杂的表格；

（一）软件实现

- **5.** 程序设计风格——效率
- 3）如何提高代码效率
  - 尽量使用执行时间短的算术运算；
  - 尽量避免混合使用不同数据类型的量；
  - 尽量使用整型算术表达式和逻辑表达式。
  - 尽量使用具有自动优化功能的编译程序，以自动生成高效的目标代码。

（一）软件实现——总结

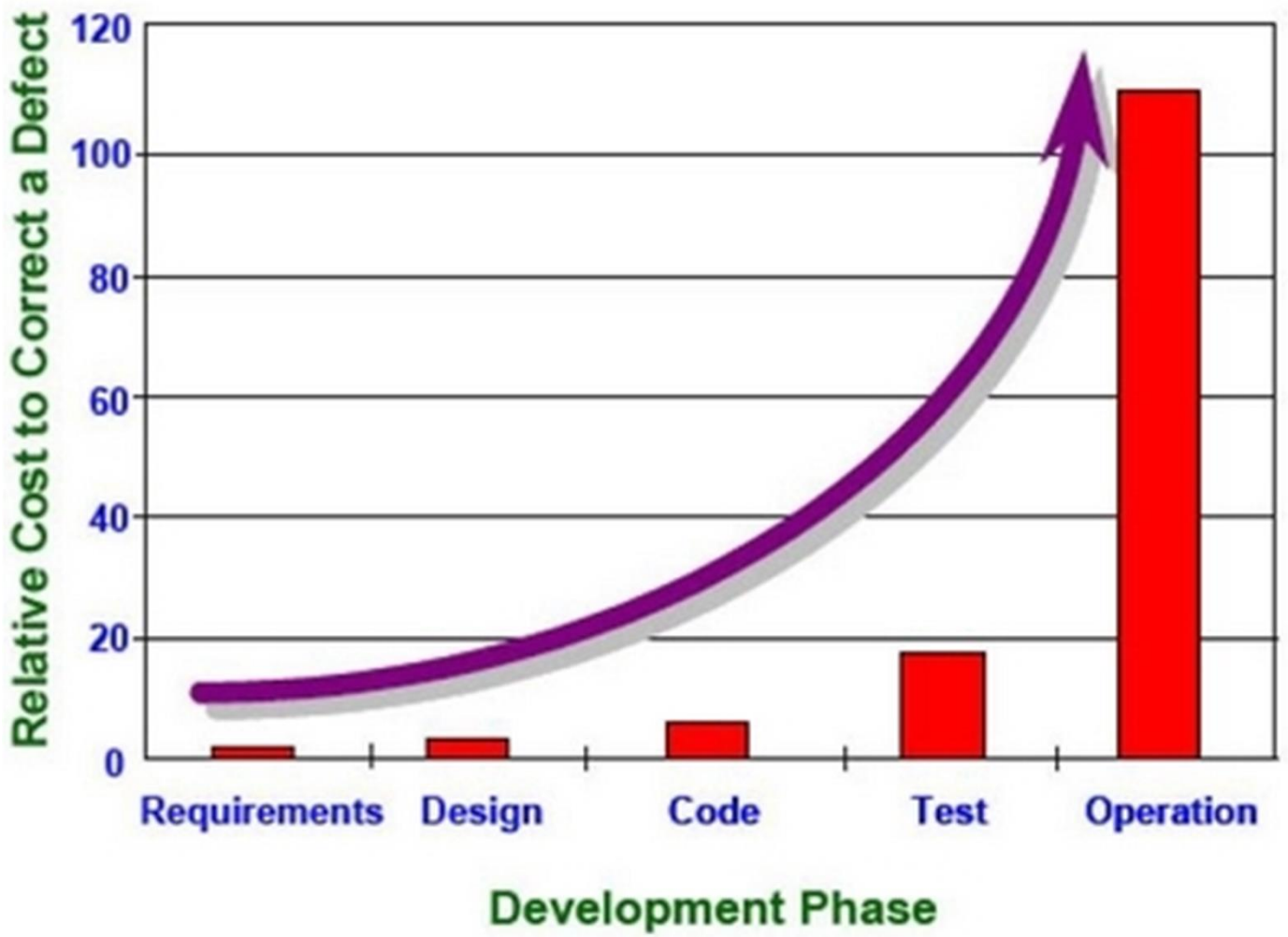
- 1.如何选择程序设计语言
- 2.程序设计风格
  - 源程序文档化：合理命名，添加注释，合理排版，是程序易读易理解。
  - 数据说明：排序，易发现错误
  - 语句构造：在保证可读性的情况下提高效率
  - 效率：提高代码效率的经验

（二）测试概述

- 1. 什么是软件测试（Software Testing）
- Testing is the process of exercising a program with the specific intent of finding errors prior to delivery to the end user.

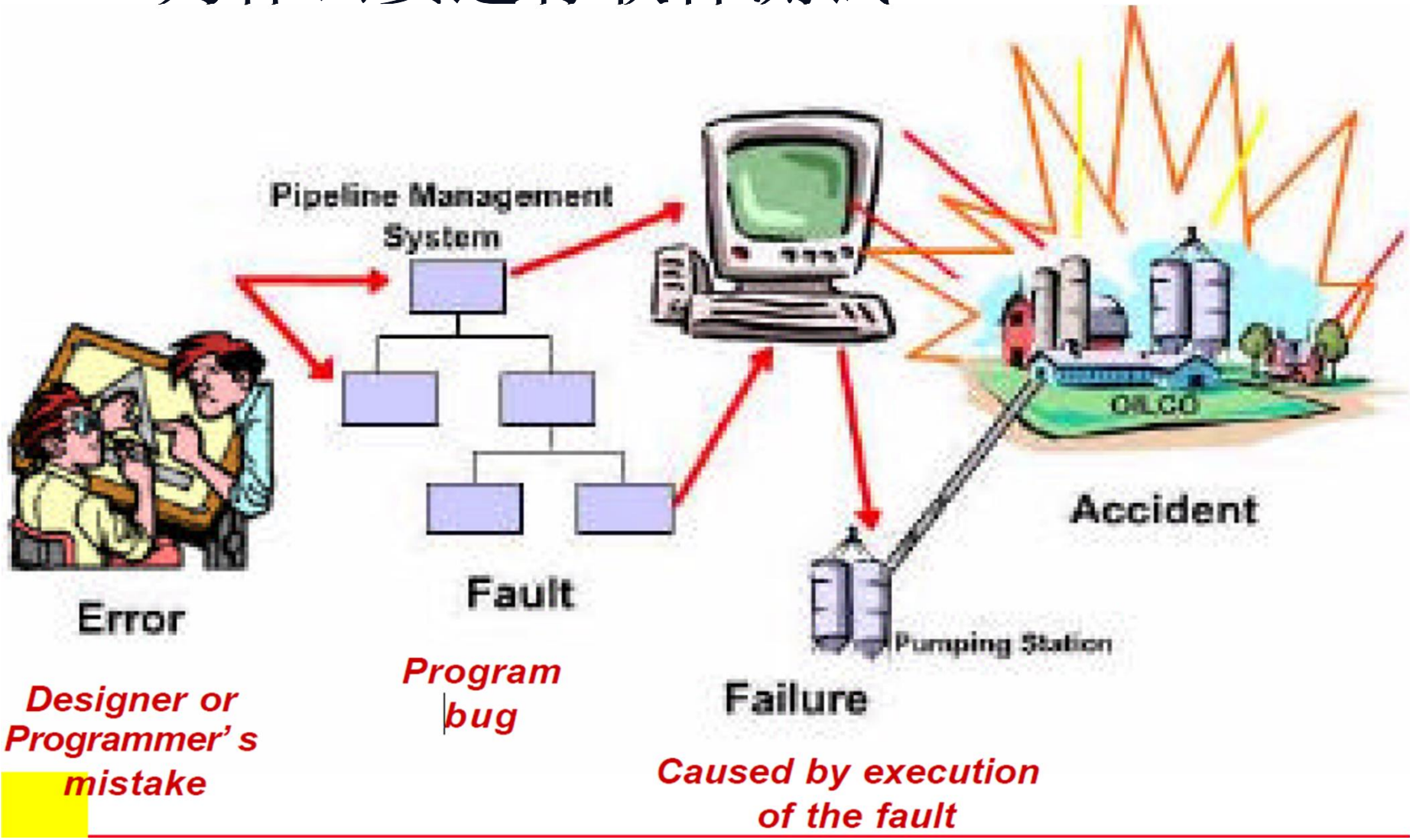
（二）测试概述

2. 为什么要进行软件测试？



（二）测试概述

❖2. 为什么要进行软件测试？



（二）测试概述

2. 为什么要进行软件测试？

- 1994年，英特尔因奔腾处理芯片存在浮点运算缺陷的问题付出4亿美元的更换费用；
- 1991年，因软件系统时钟的记时错误，爱国者导弹在海湾战争中误杀28名美国士兵。
- 2003年8月14日发生的美国及加拿大部分地区史上最大停电事故是由软件错误所引起。

（二）测试概述

2. 为什么要进行软件测试？

- 软件测试的目的是为了发现软件设计和实现过程中的疏忽所造成的错误。

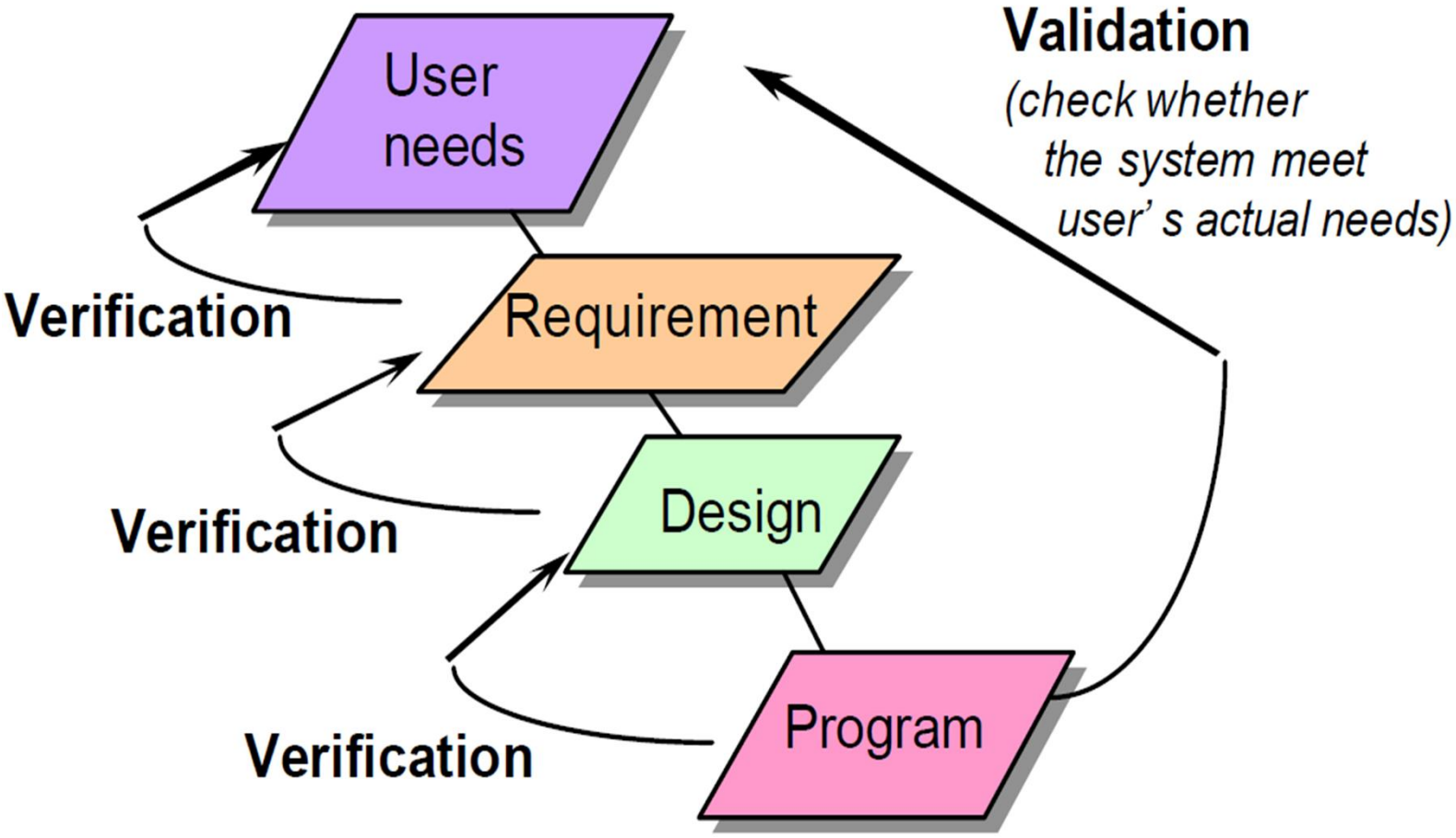
- 软件测试是验证与确认的一部分。而验证与确认是软件质量保证的主要活动。

Validation vs Verification

- **Verification – Are we building the product right?**
  - **Is the code correct with respect to its specification?**

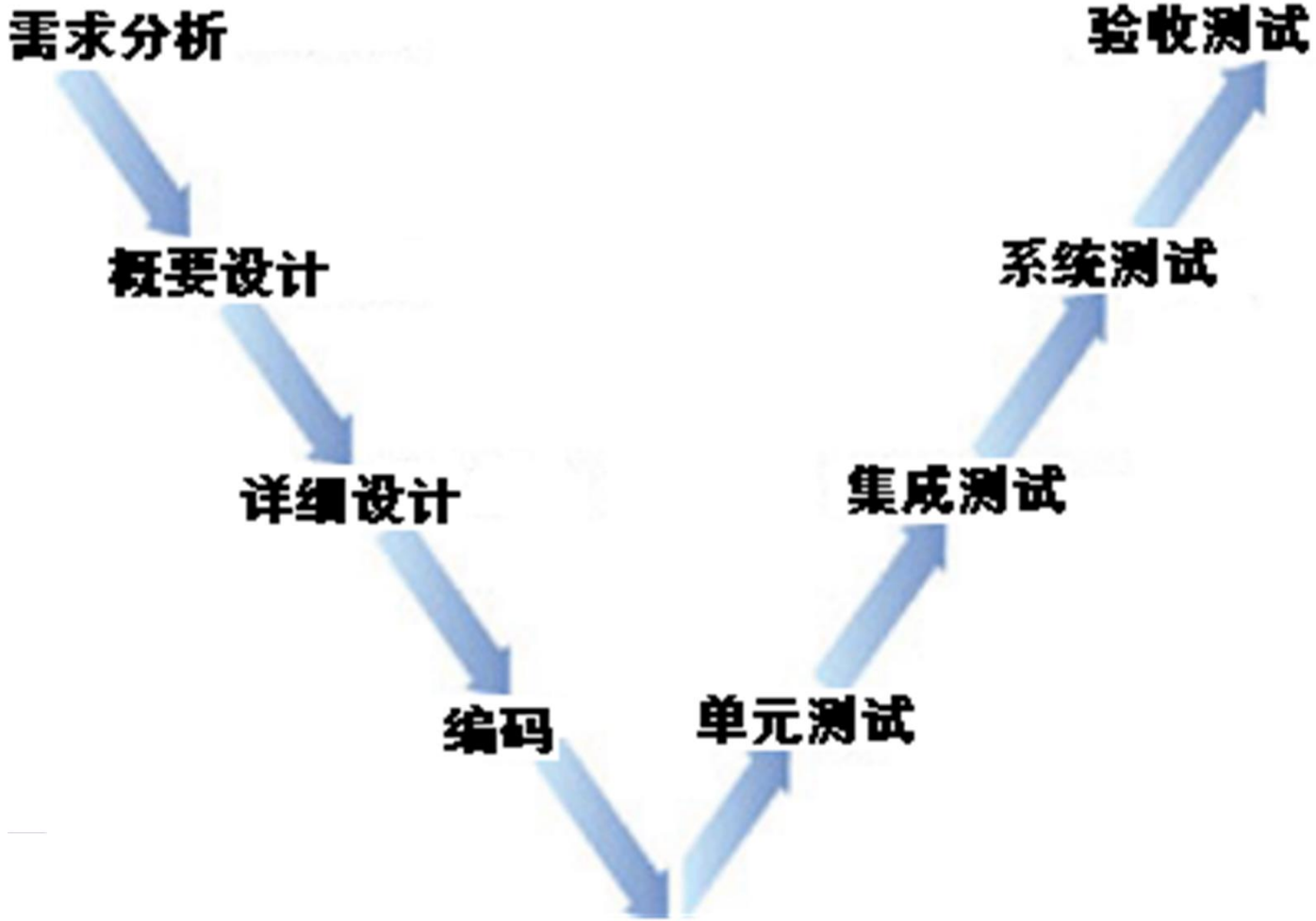
- **Validation – Are we building the right product?**
  - **Does the specification reflect what it should?**

Validation vs Verification



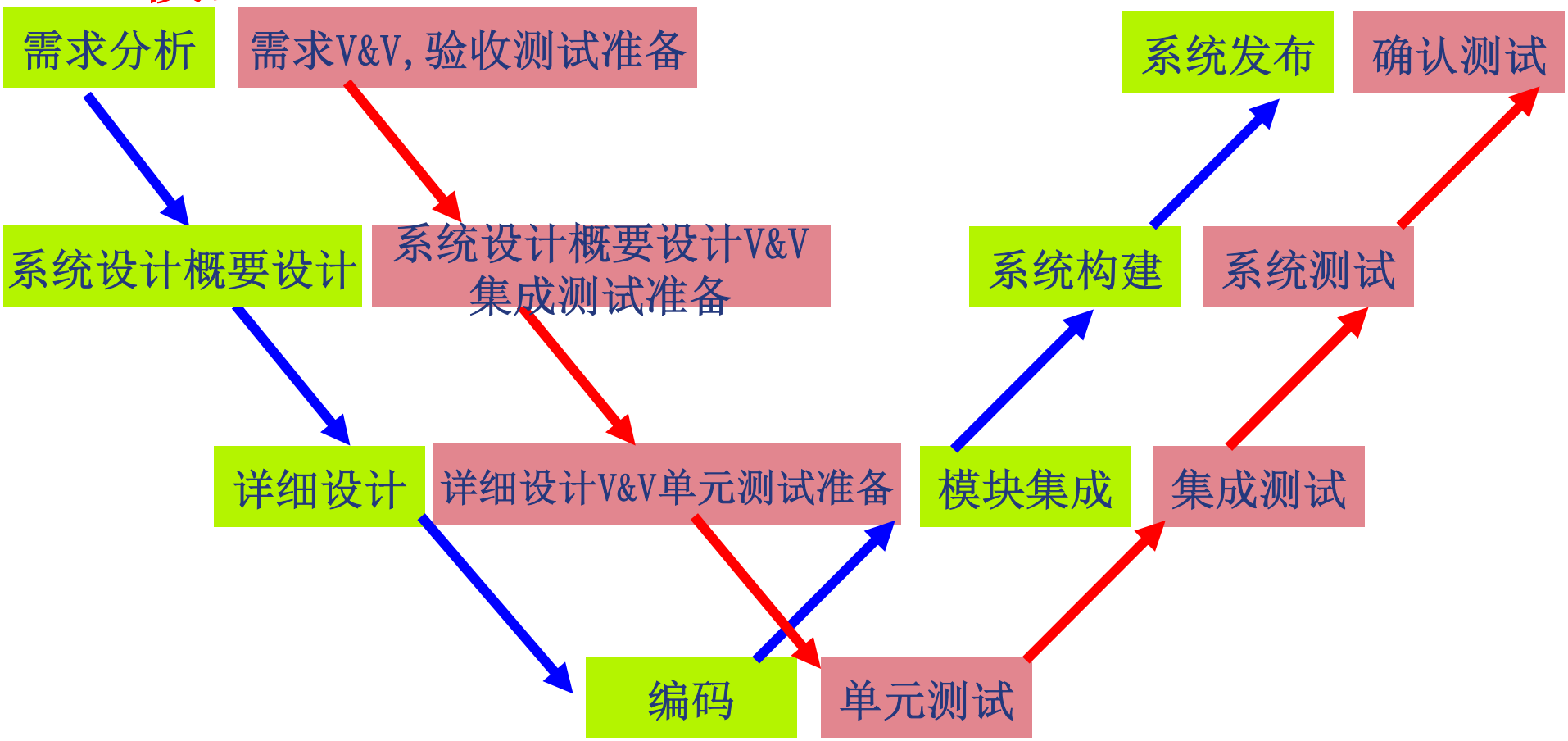
（二）测试概述

- **3.** 测试与开发阶段的对应——软件测试过程V模型



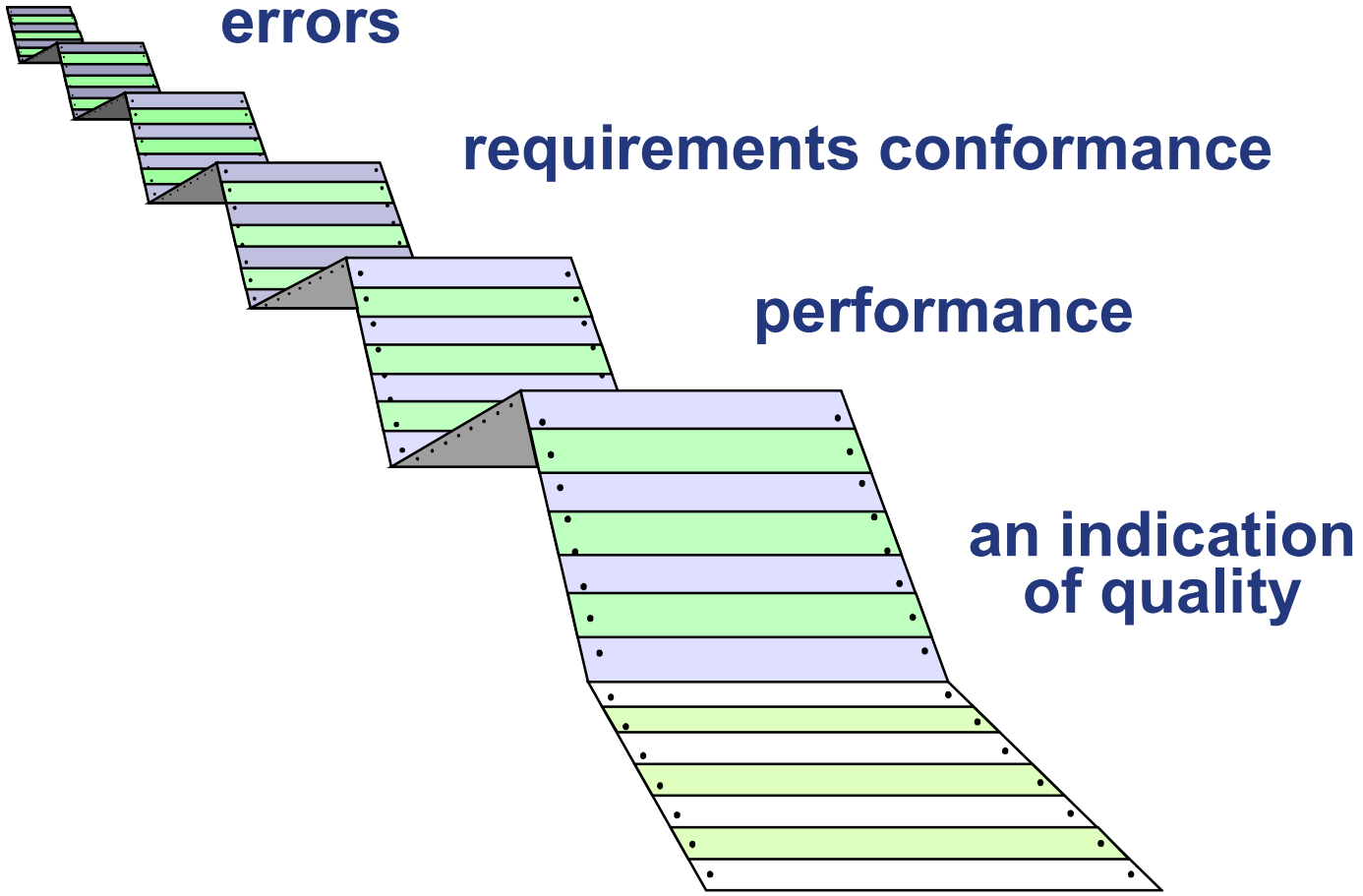
（二）测试概述

- 3. 测试与开发阶段的对应——软件测试过程W模型



（二）测试概述

4. What Testing Shows?



（二）测试概述

5. Who Tests the Software?

developer

Understands the system

but, will test "gently"

and, is driven by "delivery"

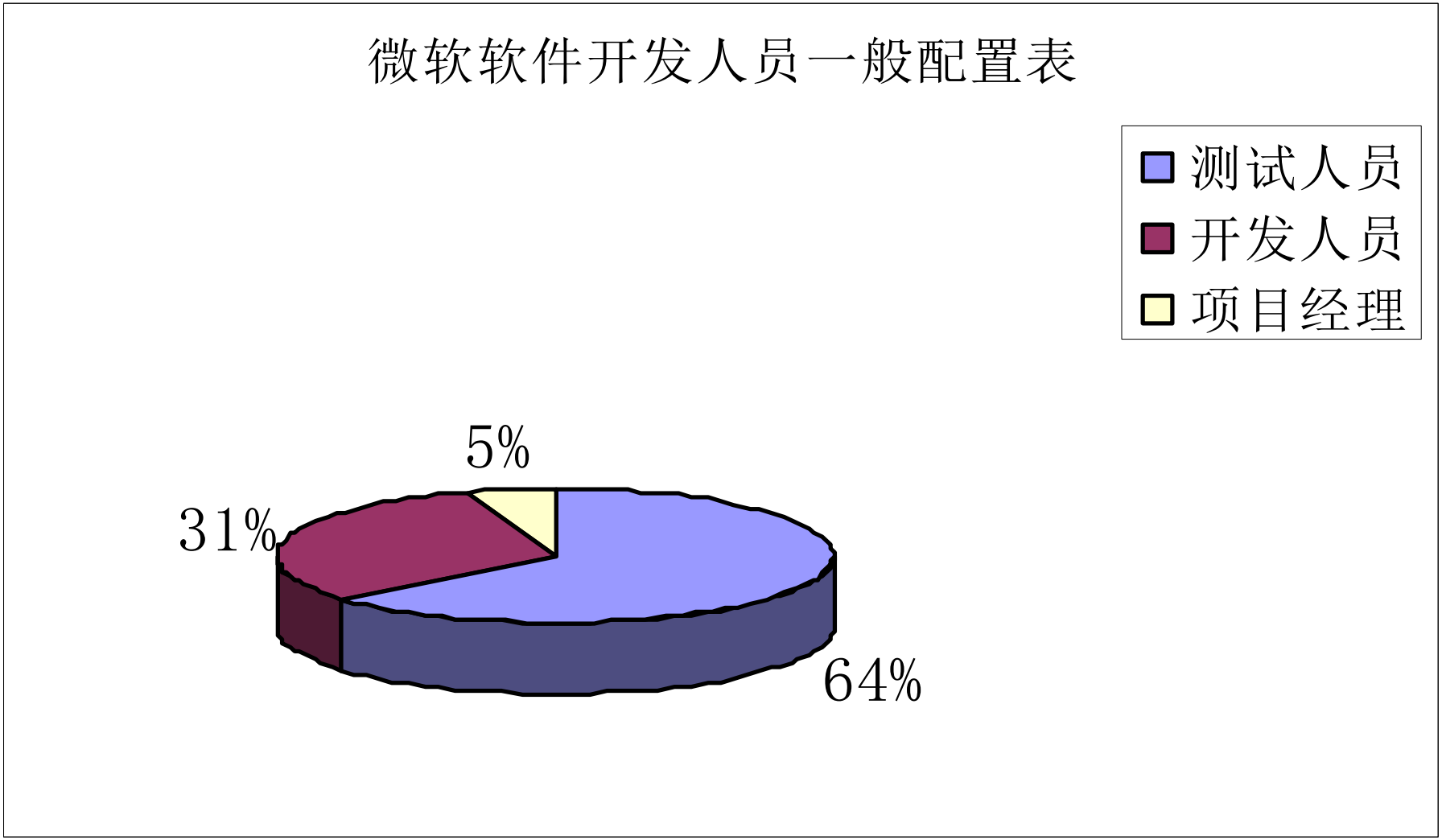
independent tester

Must learn about the system,

but, will attempt to break it

and, is driven by quality

举例：微软测试人员配置



举例：其它公司的软件测试组织配置

- 上海爱立信通讯软件研发中心
  - 开发人员：测试人员=4：1
  - 测试工作：确认测试、系统测试
- 杭州东方通信
  - 开发人员：测试人员=4：1（不包括兼职测试人员）
  - 测试工作：确认测试
- 广东电信科学技术研究院
  - 开发人员：测试人员=5：1
  - 测试工作：确认测试

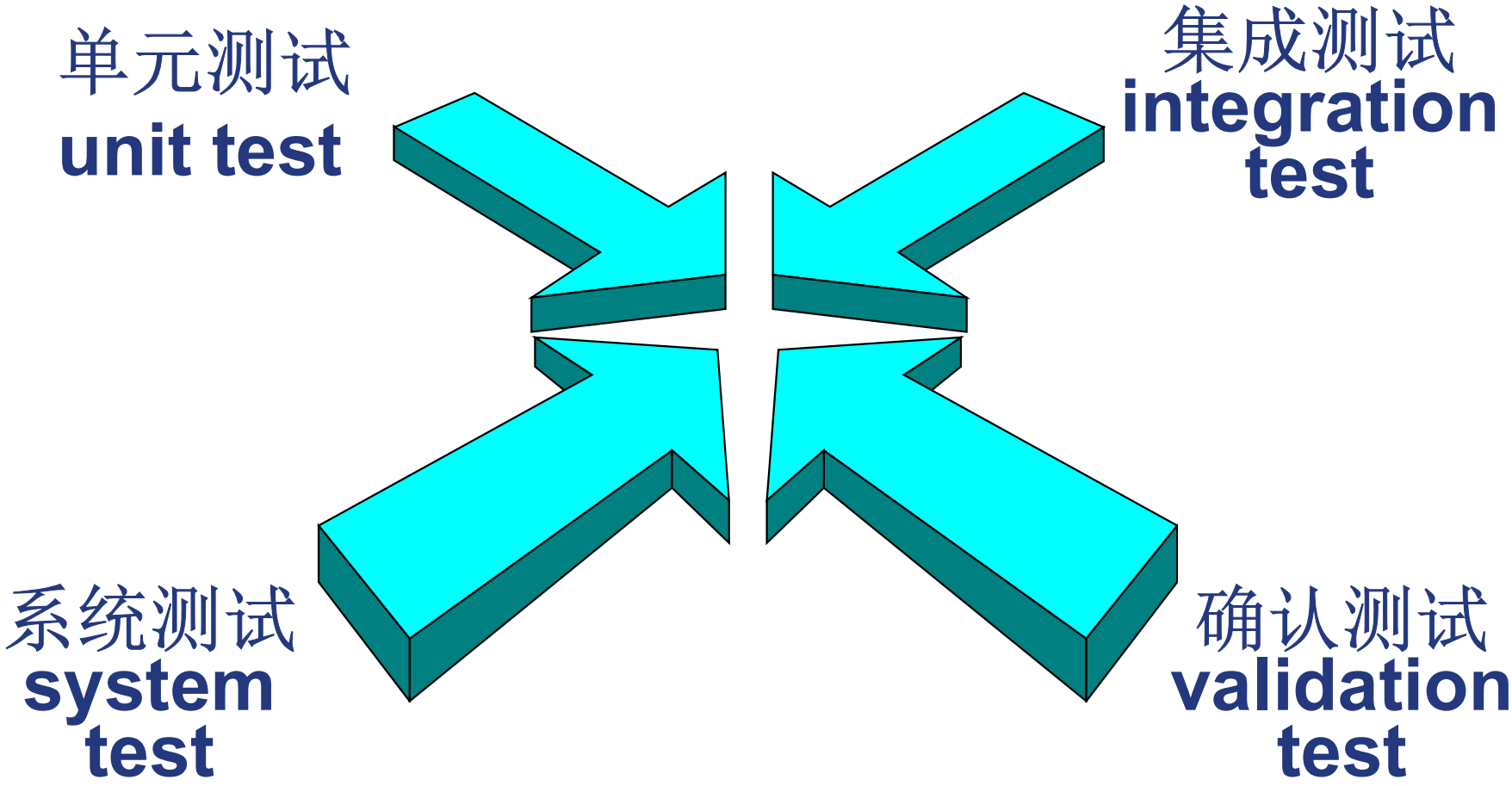
（二）测试概述

6. 软件测试过程

- 1）拟定软件测试计划：
  - 包括测试范围、目标、功能等
- 2）编写软件测试大纲：
  - 详尽规定测试项目和完成标准
- 3）设计和生成测试用例：
  - 测试用例包括的要素有：输入、执行条件、期望输出
- 4）实施测试
- 5）分析测试结果

（三）软件测试策略

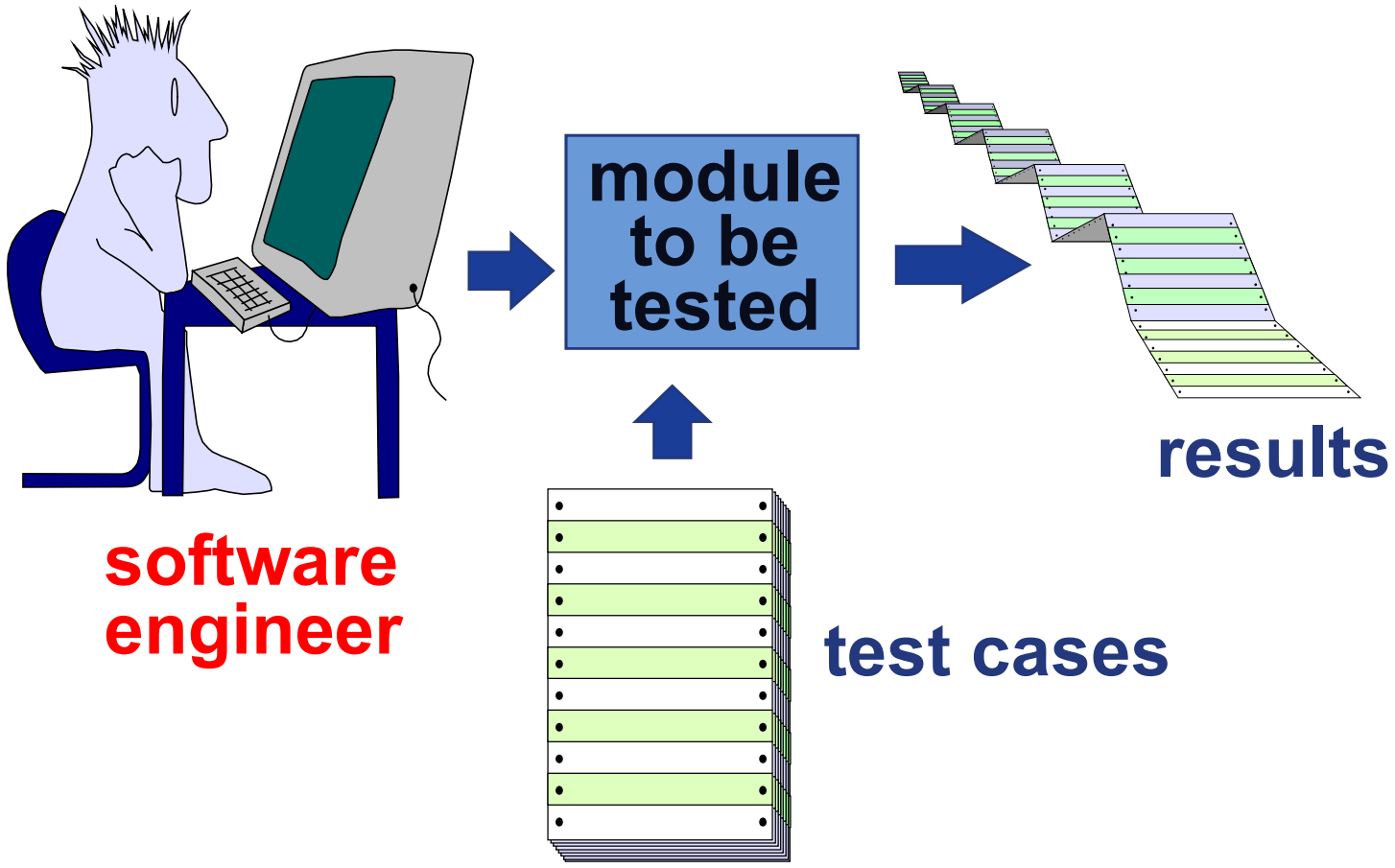
- 1. 测试策略根据软件的开发阶段来划分



（三）软件测试策略

- **1.** 测试策略根据软件的开发阶段来划分
  - 单元测试是针对程序中的模块或构件，主要揭露编码阶段产生的错误。
  - 集成测试针对集成的软件系统，主要揭露设计阶段产生的错误。
  - 系统测试：对于基于计算机系统中的软件，还需将它集成到基于计算机系统中进行测试，以揭露不符合系统工程中对软件要求的错误。
  - 确认测试是根据需求规约对软件进行确认，主要揭露不符合需求规约的错误。

（三）软件测试策略——2.单元测试



（三）软件测试策略——2.单元测试

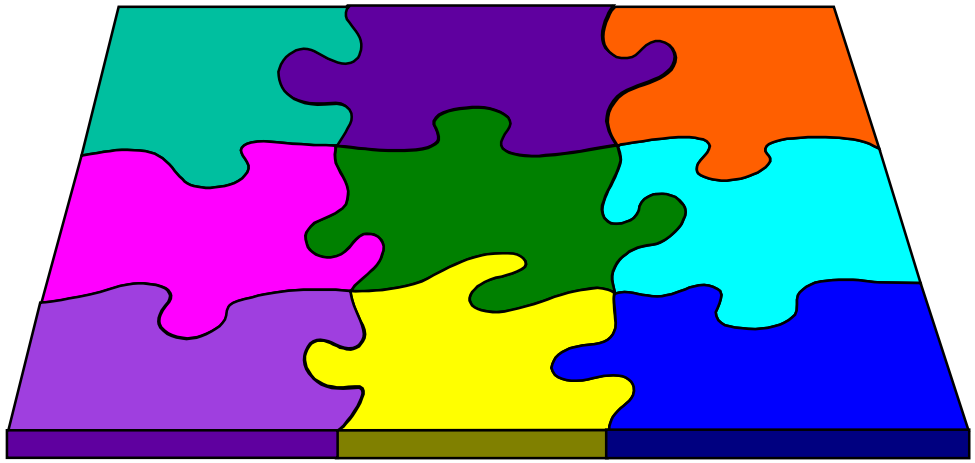
- 1) 单元测试侧重于对软件设计的最小单元的验证工作。
- 2) 单元测试的目的
  - 在开发环境中检查单元程序模块内部的逻辑、算法和数据处理结果的正确性等。
- 3) 单元测试的内容主要是：
  - 算法逻辑、数据定义的理解和使用、接口、各种控制路径、边界条件、错误处理等。

（三）软件测试策略——2.单元测试

- 4) 面向对象环境中的单元测试
  - 软件设计的最小单元的变化
    - 最小可测试单元：类中的操作（方法）
    - 然而，由于类的继承、多态等特性，独立去测试某操作往往是无效的。
  - 面向对象环境中的单元测试主要是对类的测试
    - 类测试是由封装在该类中的操作和类的状态行为驱动的
    - 对操作的孤立测试不可取

（三）软件测试策略——3.集成测试

- 集成测试是构造软件体系结构的系统化技术，同时也是进行一些旨在发现与接口相关的错误的测试。



（三）软件测试策略——3.集成测试

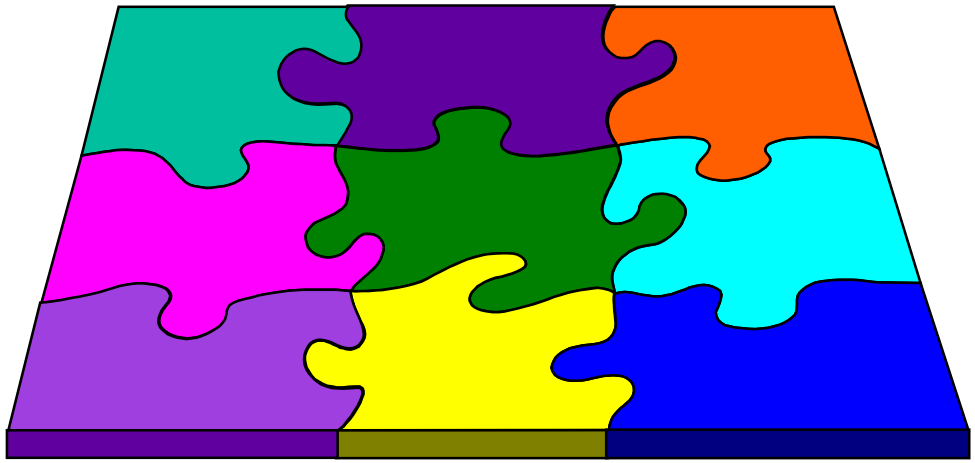
1）为什么要进行集成测试？

- 数据可能在通过接口时丢失；
- 一个模块可能对另一个模块产生非故意的、有害的影响（即副作用）；
- 当子功能被组合起来时，可能不能达到期望的主功能；
- 单个模块可以接受的不精确性（如误差），连接起来后可能会扩大到无法接受的程度；
- 全局数据结构可能会存在问题。

（三）软件测试策略——3.集成测试

2）传统的集成测试策略

- 一步到位的集成方法（**the “big bang” approach**）
- 增量式集成（**an incremental construction strategy**）
  - 自顶向下集成
  - 自底向上集成



（三）软件测试策略——3.集成测试

2）传统的集成测试策略

- 自顶向下集成
  - 从主控模块（主程序）开始，然后按照程序结构图的控制层次，将直接或间接从属于主控模块的模块按深度优先或广度优先的方式逐个集成到整个结构中，并对其进行测试。
- 自底向上集成
  - 从程序结构的最底层模块（即原子模块）开始，然后按照程序结构图的控制层次将上层模块集成到整个结构中，并对其进行测试。

（三）软件测试策略——3.集成测试

3）面向对象的集成测试策略

- 面向对象软件无明显的层次控制结构，所以无法自顶向下集成或者自底向上集成；
- 由于类的成分之间直接或间接的相互作用，往往无法每次仅集成一个操作。
- 通常，面向对象系统集成测试有两种策略：
  - 基于线程的测试（thread based testing）
  - 基于使用的测试（use based testing）

（三）软件测试策略——3.集成测试

3）面向对象的集成测试策略

- 基于线程的测试（thread based testing）
  - 集成响应系统的一个输入或事件所需的一组类，每个线程单独地集成和测试。
- 基于使用的测试（use based testing）
  - 先测试独立类，然后测试依赖类，直到整个系统集成成功

（三）软件测试策略——3.集成测试

- 4）回归测试
  - 对已经进行过的测试的子集的重新执行，以确保对程序的改变和修改，没有传播非故意的副作用。
- 5）冒烟测试
  - **Daily Building**
  - 尽早发现可能造成项目延迟的业务阻塞（**show stopper**）错误

（三）软件测试策略——4.系统测试

- 系统测试：
  - 对完整集成后的产品和解决方案进行测试，用来评价系统对具体需求规格说明的功能和非功能的符合性的测试
- 系统测试的目的/作用：
  - 发现可能难以直接与模块或接口关联的缺陷
  - 发现产品设计、体系和代码的基础问题（产品级缺陷）

（三）软件测试策略——4.系统测试

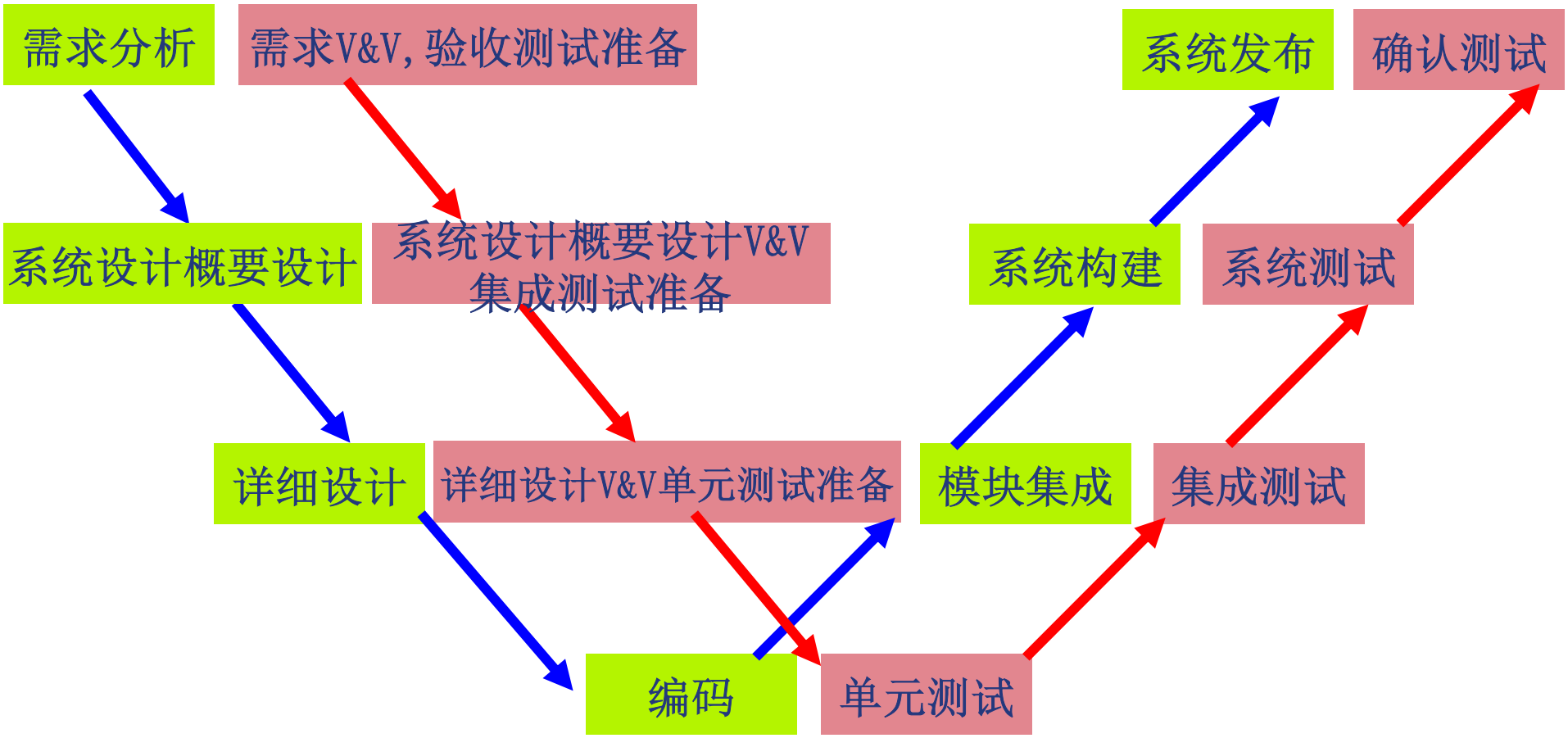
- 系统测试=功能测试+非功能测试
- 功能测试
  - 设计/体系结构测试、业务垂直测试、部署测试，等等
- 非功能测试
  - 可伸缩性测试/容量测试、可靠性测试、压力测试、国际化测试、性能测试、安全性测试，等等

（三）软件测试策略——5.确认测试

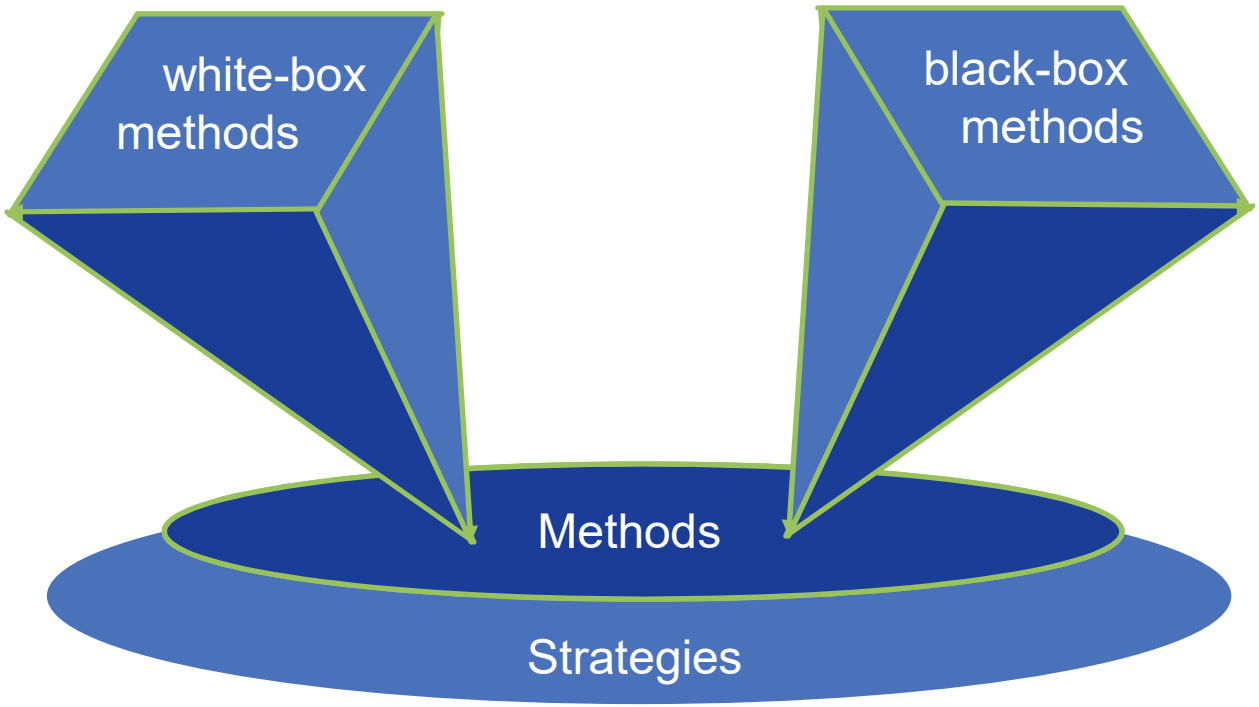
- 确认测试以软件需求规约为依据，以发现软件与需求不一致的错误。
- 主要检查：
  - 软件是否实现了规约规定的全部功能要求
  - 文档资料是否完整、正确、合理
  - 其他的需求，如可移植性、可维护性、兼容性、错误恢复能力等是否满足

（三）测试策略——总结

- 测试策略与开发阶段的对应：



（四）软件测试技术



（四）软件测试技术

1. 黑盒方法和白盒方法

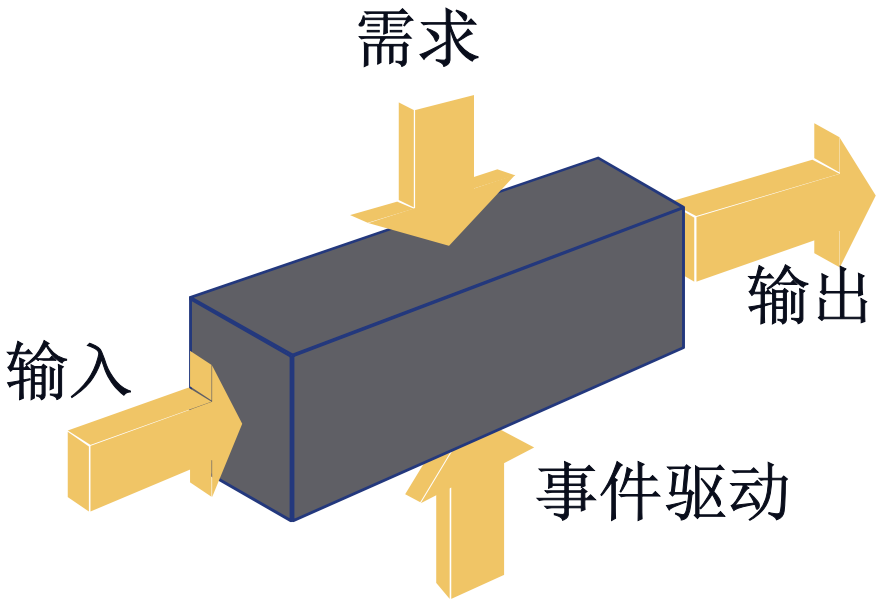
- 黑盒测试方法（**Blake-box Testing**）
  - 也称为行为测试
  - 把程序看作一个不能打开的黑盒子，不考虑程序内部结构和内部特性
  - 考察数据的输入、条件限制和数据输出，以完成测试 。

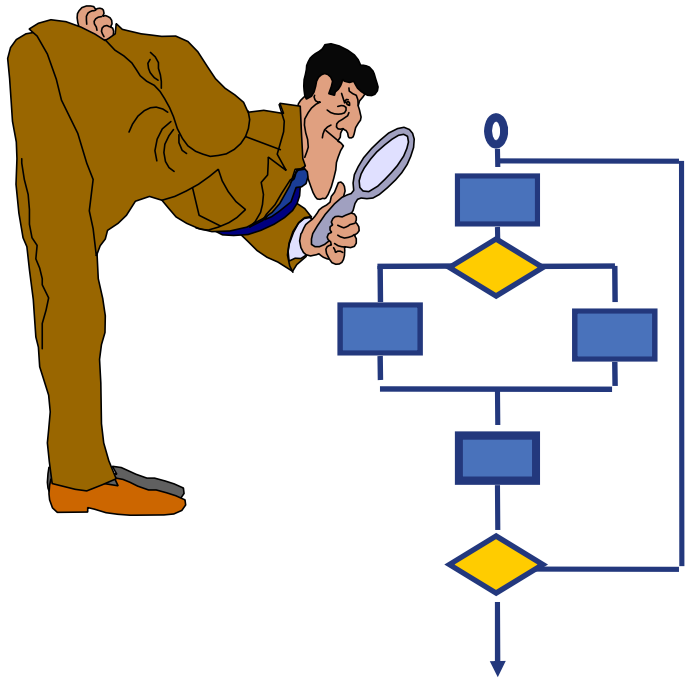
（四）软件测试技术

1. 黑盒方法和白盒方法

- 白盒测试方法（White-box Testing）
  - 也称结构测试或逻辑驱动测试
  - 基于内部逻辑结构，针对程序语句、路径、变量状态等来进行测试
  - 检验程序中的各个分支条件是否得到满足、每条执行路径是否按预定要求正确的工作。

黑盒测试 vs. 白盒测试





黑盒测试 vs. 白盒测试

黑盒测试方法	白盒测试方法
等价类划分 边界值分析 因果图 错误推测法 决策表方法 正交试验法 ……	桌面检查 代码走查 代码审查 逻辑覆盖 基本路径测试 ……

（四）软件测试技术

2. 静态测试 和 动态测试

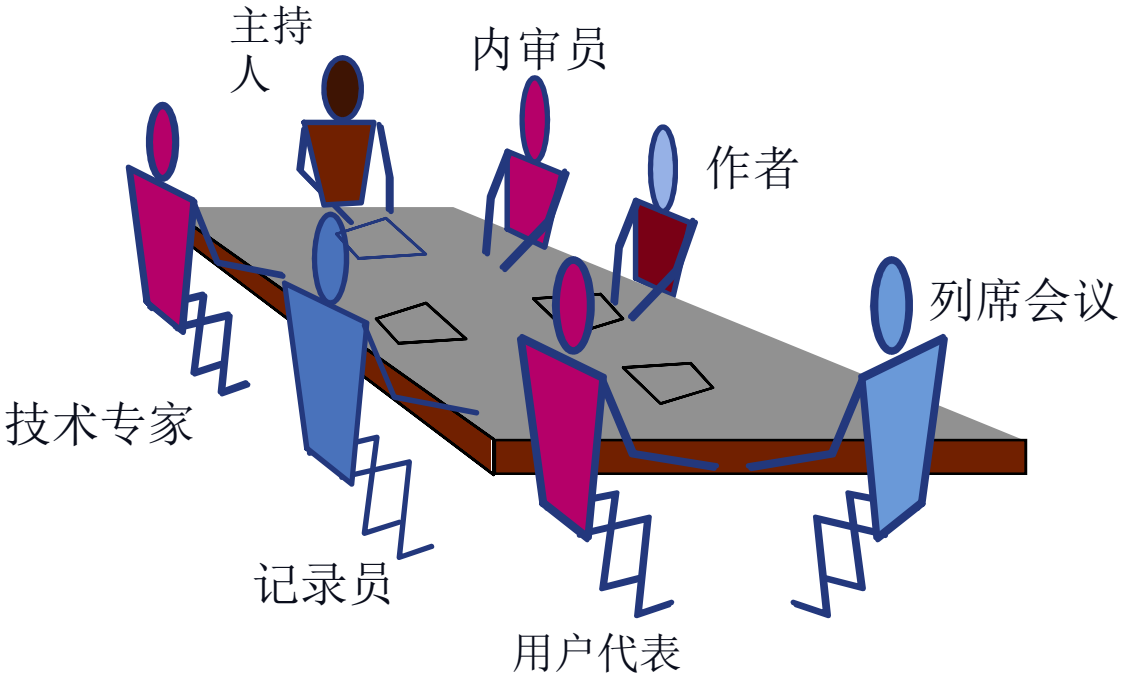
- 静态测试
  - 就是静态分析
  - 对模块的源代码进行研读，查找错误或收集一些度量数据
  - 不需要对代码进行编译和仿真运行
  - 如：桌面检查、代码走查、代码审查

（四）软件测试技术

2. 静态测试 和 动态测试

- 动态测试
  - 动态测试通过真正运行程序发现错误
  - 通过有效的测试用例，对应的输入/输出关系来分析被测程序的运行情况
  - 通过观察代码运行时的动作，来提供执行跟踪、时间分析，以及测试覆盖度方面的信息。
  - 如：黑盒测试方法

静态测试 vs. 动态测试



非正式 正式

桌面检查 走查 评审会议

运行程序

（四）软件测试技术

3. 手工测试 和 自动化测试

- 静态分析通常只能进行手工测试，虽然也有辅助工具帮助，但仍以人为主体。

- 动态测试往往可以部分地或者全部自动化进行。
  - 提供有效的测试用例集，通过测试平台对被测程序进行测试，跟踪并分析测试结果。

（四）软件测试技术

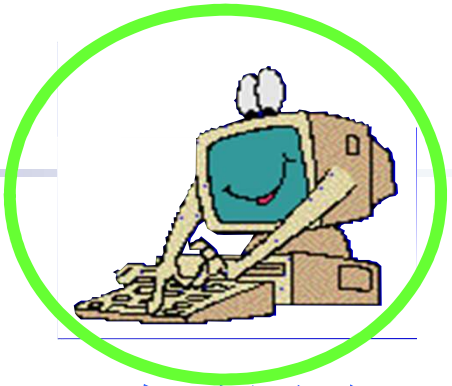
3. 手工测试 和 自动化测试

- 自动化测试的特点
  - 自动运行的速度快
  - 测试结果准确
    - 例如搜索用时是0.33秒或0.24秒，系统都会发现问题，不会忽视任何差异
  - 高复用性
    - 一旦完成所用的测试脚本，可以一劳永逸运行很多遍
  - 永不疲劳
  - 可靠
  - 独特的能力

手工测试 vs.自动测试

手工测试

- 发现缺陷率高
- 容易实施
- 创造性、灵活性
- 覆盖率量化困难
- 重复测试效率低
- 不一致性、可靠性低
- 依赖人力资源



自动测试

- ✓ 高效率（速度）
- ✓ 高复用性
- ✓ 覆盖率容易度量
- ✓ 准确、可靠
- ✓ 不知疲劳
- ✓ 激励团队士气
- ✓ 机械、难以发现缺陷
- ✓ 一次性投入大

（四）软件测试技术

3. 手工测试 和 自动化测试

- 两者互相补充
  - 在系统功能逻辑测试、验收测试、适用性测试、涉及交互性测试时，多采用手工测试方法；
  - 单元测试、集成测试、系统负载或性能、可靠性测试等比较适合采用自动化；
  - 由于工具本身缺乏想象力和创造性，自动测试只能发现**15%**的缺陷，而手工测试可以发现**85%**的缺陷

Southeast University

Thank You !

lliao@seu.edu.cn