

软件测试基础与实践

Software Testing: Foundations and Practices

第4讲 单元测试与集成测试

教师: 汪鹏 廖力

软件工程专业 主干课程



大爆炸开发模式与软件测试：

- 开发过程混乱
- 期待奇迹出现

测试的困难：

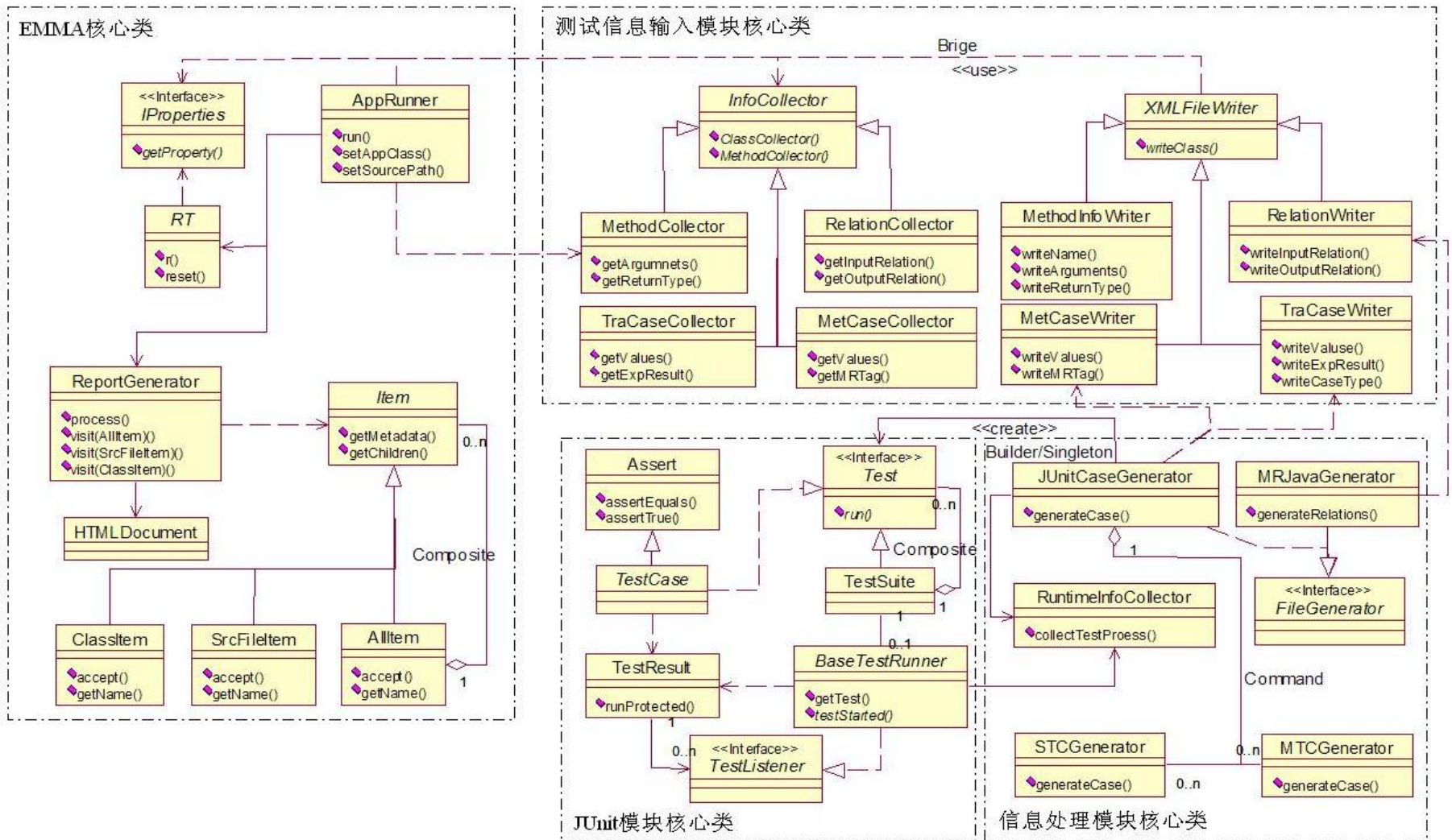
- 难以找到导致缺陷的原因
- 某些缺陷掩盖了其它缺陷，造成测试失败

解决：

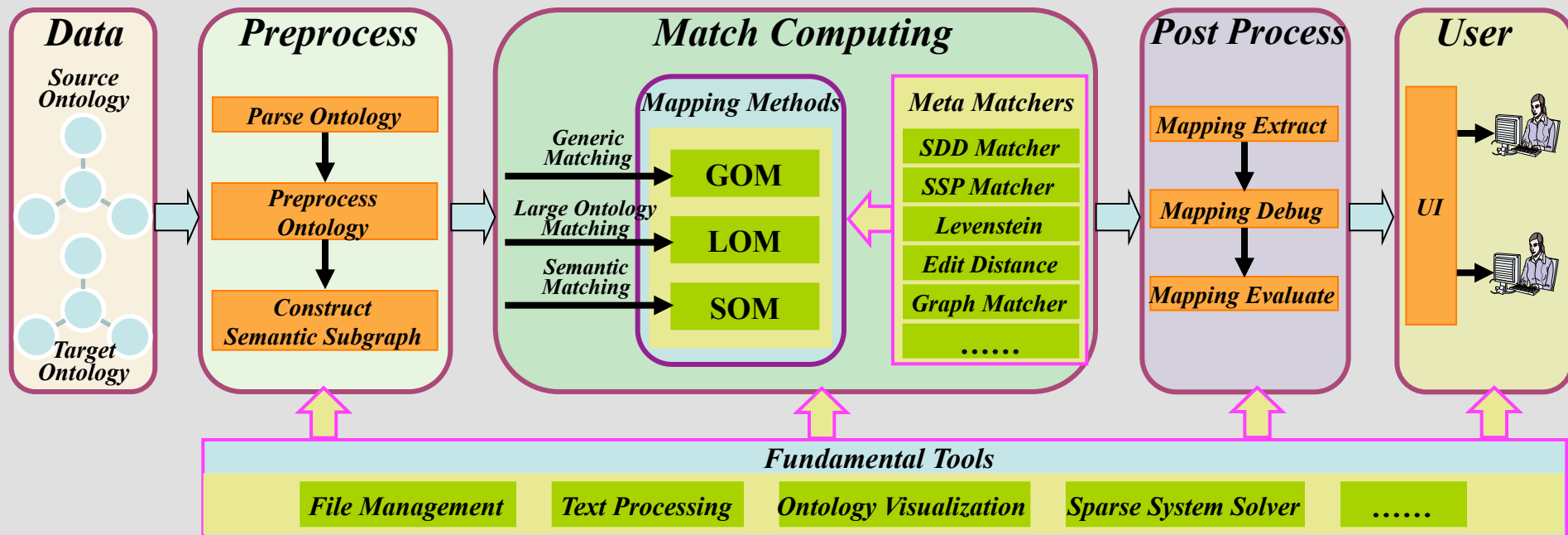
- 代码分阶段构建
- 模块分阶段测试
- 模块组合形成更加的部分

自动化蜕变测试工具jRider

核心类的UML类图



本体映射工具Lily系统架构



模块：实验数据、预处理、匹配计算、后处理、用户平台、基础工具库

本讲内容

单元测试 **基本概念、测试规程、特点 ...**

集成测试 **基本概念、集成测试策略 ...**

测试插桩 **白盒插桩:监测状态/条件/前提/常量
黑盒插桩:测试预言、随机数据生成**

测试工具 **Junit, VSUnit, CPPUnit, ...**

一 单元测试



1. 单元测试的特点?
2. 单元测试的关注点?
3. 单元测试的实施?
4. 驱动器和程序桩?

单元测试——基本概念

■ 软件单元

软件单元(Software Unit):

一个应用程序中的**最小可测部分**

面向过程中的单元:

单独的程序、函数、过程、网页、菜单、...

面向对象中的单元:

类（基类、抽象类、子类）

单元测试——基本概念

■ 定义

单元测试(Unit Testing)/模块测试(Module Testing):
对最小的软件设计单元（模块/源程序单元）的验证工作

■ 意义

- 1.消除软件单元本身的不确定性
- 2.其它测试阶段的必要的基础环节

■ 实施者

软件开发人员

单元测试——基本概念

■测试目标

- 单元体现了预期的**功能** (**黑盒** **白盒**)
- 单元的运行能覆盖预定的各种**逻辑** (**白盒**)
- 单元工作中内部**数据**能够保持完整性 (**白盒**)
- 可接受**正确数据**，也能处理**非法数据** (**黑盒**)
- 在数据**边界**条件上，单元能正确工作 (**黑盒**)
- 单元的算法合理，**性能**良好 (**黑盒**)
- 扫描单元代码未发现任何**安全性**问题 (**白盒**)

.....

单元测试——基本概念

■ 测试技术

白盒测试技术：

使用一种或多种白盒测试方法分析模块的逻辑结构

黑盒测试技术：

使用黑盒测试方法对照模块的规格说明以补充测试用例

单元测试——测试关注点

■ 单元测试关注点

- 模块功能
- 内部逻辑处理
- 数据结构
- 接口
- 边界条件
- 独立路径
- 错误处理
- 性能、安全...

单元测试——测试关注点

■模块或构件接口

目标：

进出模块/构件的数据流正确

关注点：

- (1)接口名称、参数个数、类型、匹配顺序
- (2)输出或返回值及其类型是否正确

单元测试——测试关注点

■局部数据结构

目标：

数据在模块执行过程中维持完整性和正确性

关注点：

- (1)数据结构定义和使用过程的正确性
- (2)局部数据结构对全局数据结构的影响

单元测试——测试关注点

■边界条件

目标:

保证模块在条件边界上能够正确执行

关注点:

- (1)数据结构中的边界 (例如: 数组 $a[n]$)
- (2)控制流中的边界 (例如: 循环次数、判断条件)

单元测试——测试关注点

■独立路径

目标:

保证模块中的每条独立路径(基本路径)都被覆盖, 使得所有语句都被执行过一次

关注点:

对路径的选择性测试 (基本路径测试+循环测试)

单元测试——测试关注点

■ 处理错误的路径

目标:

保证错误处理的正确性,软件的健壮性

好的软件设计要求错误条件是可预知的,并且当错误发生时, 错误处理路径被建立, 以重定向或终止处理。

Java: try、catch、throw、throws、finally

C++: try、catch、throw

单元测试——测试关注点

■ 处理错误的路径

错误处理关注点:

- (1) 异常条件的正确性
- (2) 异常的可发生性
- (3) 异常处理的逻辑正确性

足够进行单元测试了吗？



单元测试规程

■ 驱动器和程序桩

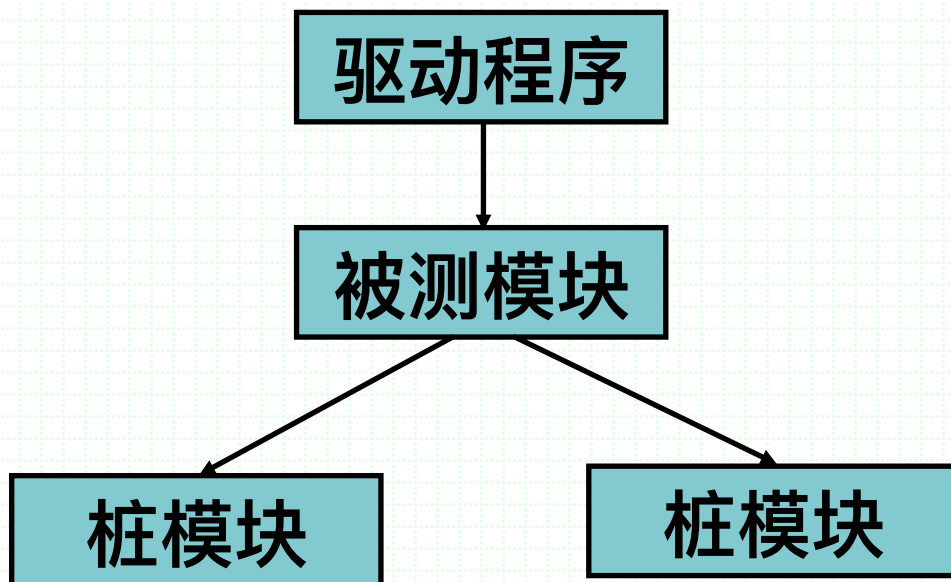
绝大多数情况下，单个单元是不能正常工作的，而是要和其他单元一起才能正常工作。

实际测试活动中的普遍问题：
其它单元不存在的情况下，如何运行单个模块？

单元测试规程

■ 驱动器和程序桩

- 单元测试通常与编码工作结合起来进行。
- 模块本身不是一个独立的程序，在测试模块时，必须为每个被测模块开发一个驱动器(driver) 和若干个桩程序(stub)



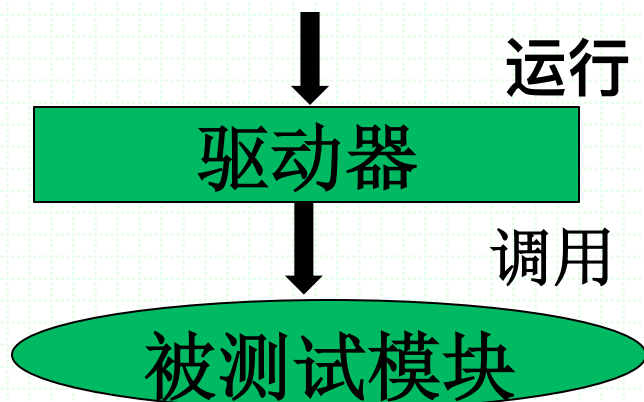
单元测试规程

■ 驱动器和程序桩

驱动器 (Driver) :

对底层或子层模块进行测试时所编制的调用被测模块的程序，用以模拟被测模块的上级模块。

接收测试数据，并把数据传送给被测模块，最后输出相关结果。

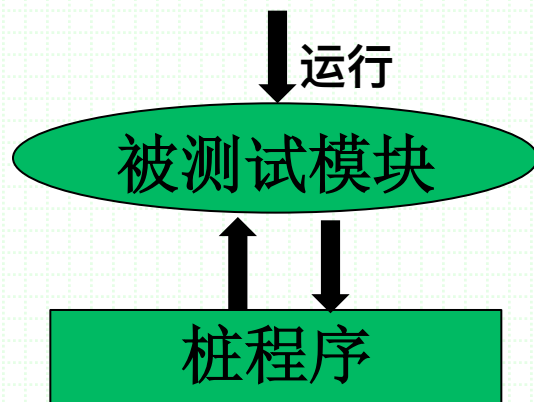


单元测试规程

■ 驱动器和程序桩

桩程序 (Stub) :

对上层模块进行测试时，所编制的替代下层模块的程序，用以模拟被测模块工作过程中所调用的模块。
模拟未被测试模块/隶属模块的功能



单元测试规程

■ 驱动器和程序桩

驱动器结构一般如下：

数据说明；

初始化；

输入测试数据；

调用被测模块；

输出测试结果；

停止

桩程序结构一般如下：

数据说明；

初始化；

输出提示信息（表示进入了哪个桩模块）

验证调用参数；

打印验证结果；

将模拟结果送回被测程序；

返回

单元测试规程

■手工和自动化测试

手工测试：

按照需求规格设计测试用例完成测试

静态测试

自动测试：

自动化方法的效果：有效验证较为独立单元的正确性

单元测试驱使程序员创建松耦合、高内聚的代码体，有助于开发健壮的软件

单元测试局限性

- 只验证单元自身的功能，不能捕获系统范围的错误
系统错误：集成错误、性能问题等
- 被测模块现实中可能接收的所有输入情况难以预料

二 集成测试



1. 集成测试的概念？
2. 集成测试的关注点？
3. 集成测试的策略？

集成测试基本概念

火星登陆事故(Mars Polar Lander)



Nancy G. Leveson, The Role of Software in Spacecraft Accidents

集成测试基本概念

■ 定义

集成测试(Integration Testing):

把单独的软件模块结合在一起作为整体接受测试

■ 意义

1. 验证软件单元间能否协调工作
2. 验证单元集合的功能、性能和可靠性需求

■ 实施者

软件测试人员+软件开发人员

集成测试基本概念

■ 测试技术和步骤

技术：

黑盒测试技术为主
白盒测试技术为辅

步骤：

与集成测试策略相关

集成测试基本概念

■ 进入和退出条件

进入条件：

设计阶段：测试场景和测试数据

编码阶段：若干模块有集成需求时便执行测试

退出条件：

(1)完成测试计划

(2)发现并修正了错误

(3)预算和开发时间

集成测试基本概念

■ 集成测试必要性

接口连接问题：

- 函数相互调用；
- 模块间的相互影响；
- 单个模块中的缺陷发生扩散；
- 全局数据结构；
- ...

集成测试基本概念

■ 集成测试目标

集成测试用来构造程序并实施测试以**发现与接口连接有关的错误**，它把经过单元测试的模块拿来，构造一个在设计中所描述的场景

集成测试基本概念

■ 接口

内部接口：

产品内部两模块之间通信的接口

外部接口：

产品之外第三方可以看到的接口

接口提供方法：API、SDK

集成测试基本概念

■ 接口

显式接口：

写入文档的明确化的接口

隐含接口：

未写入文档，只有开发人员知道
如：Windows隐藏的API

如何进行集成测试？



集成测试策略

- 瞬时集成测试

- 增量集成测试

集成测试策略

■ 瞬时集成测试

又称大爆炸测试策略

特点：

当所有构件都通过单元测试，就把他们组合成一个最终系统，并观察它是否正常运转

集成测试策略

■ 瞬时集成测试

缺陷：

- 无休止的错误：

- 错误很多；

- 错误修复很困难；

- 修正错误后，新的错误马上出现

- 模块一次性结合，难以找出错误原因

- 接口错误和其它错误容易混淆

适用领域：

小型软件开发

集成测试策略

■ 增量集成测试

特点：

将程序分成小的部分进行构造和测试

优点：

1. 错误容易分离和修正
2. 接口容易进行彻底测试

缺点：

会有额外开销，但能大大减少发现和修正错误的时间

集成测试策略—增量集成测试

■ 三种增量集成测试

- 自顶向下集成

- 自底向上集成

- 混合式集成

集成测试策略—增量集成测试

■ 自顶向下集成

集成顺序：

先集成主模块(主程序)，然后按照控制层次向下进行集成

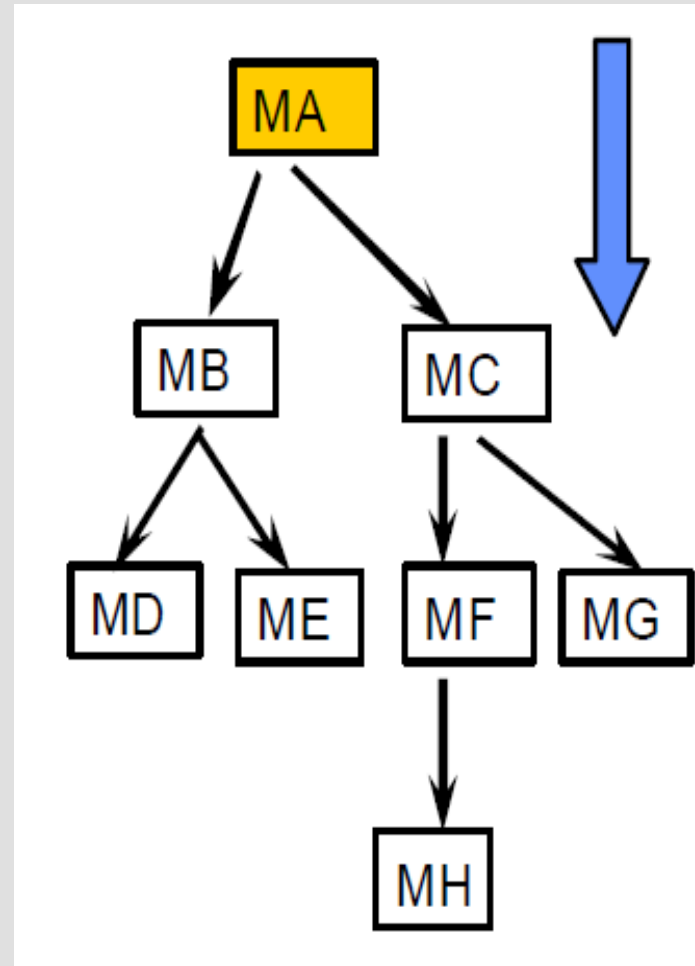
集成方式：

1. 深度优先
2. 广度优先

广度优先集成

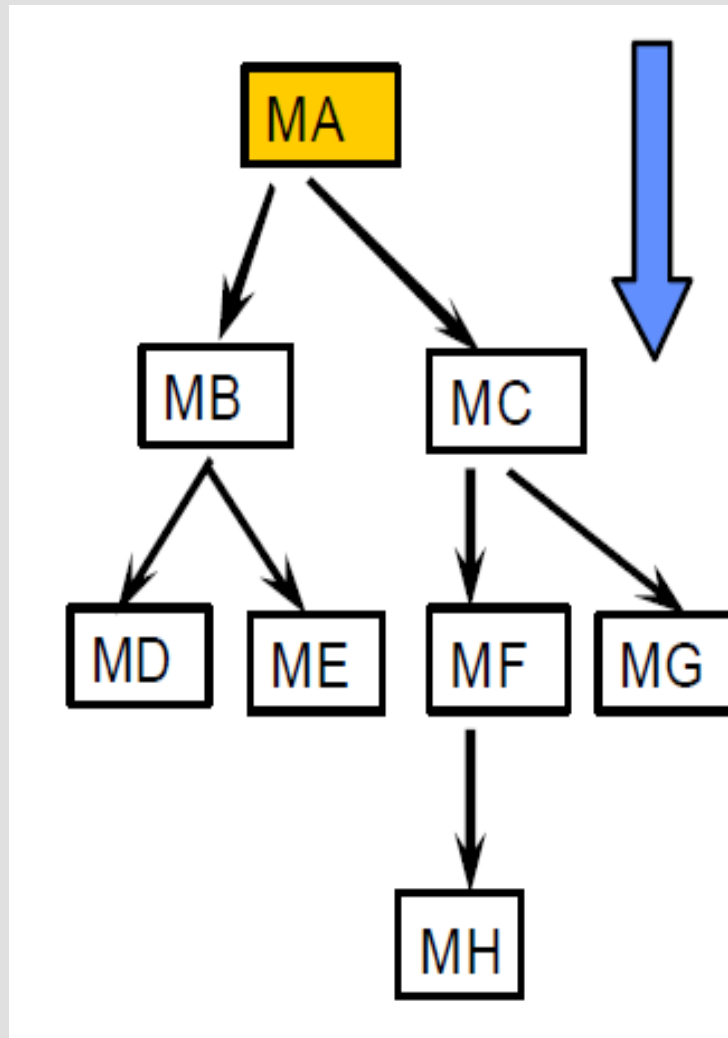
A possible sequence of integration is:

- Unit test MA
- Integration test MA, MB
- Integration test MA, MB, MC
- Integration test MA, MB, MC, MD
- Integration test MA, MB, MC, MD, ME
- Integration test MA, MB, MC, MD, ME, MF
- Integration test MA, MB, MC, MD, ME, MF, MG
- Integration test all modules



广度优先: A B C D E F G H

深度优先集成



深度优先: **A B D E C F H G**

集成测试策略—增量集成测试

■ 自顶向下集成

集成步骤：

*Step1.*主程序作为测试驱动器

*Step2.*根据集成顺序，程序桩逐渐被各模块替换

*Step3.*每个模块集成时都需要进行测试

*Step4.*每完成一次测试后，将新模块替换程序桩

*Step5.*利用回归测试来保证没有引进新的错误

集成测试策略—增量集成测试

■ 自顶向下集成

优点：

1. 尽早发现高层控制和决策错误
2. 只需要一个驱动器
3. 每步只增加一个模块
4. 支持深度优先和广度优先

集成测试策略—增量集成测试

■ 自顶向下集成

实施的主要难点：

高层测试需要在低层测试完成后才可进行的情形

解决方法：

1. 推迟测试，直到低层模块完成测试；
2. 设计程序桩，模拟低层模块；
3. 自底向上测试

集成测试策略—增量集成测试

■ 自顶向下集成

缺点：

- 1.对低层模块行为的验证比较晚
- 2.需要编写额外程序模拟未测试的模块
- 3.测试用例的**输入和输出很难明确表示**

集成测试策略—增量集成测试

■ 自底向上集成

集成顺序：

从原子模块，即程序最底层模块开始就行构造并进行集成测试

原子模块→造件(Build)→应用软件系统

集成测试策略—增量集成测试

■ 自底向上集成

集成步骤：

Step1. 低层模块组成实现特定子功能的构件

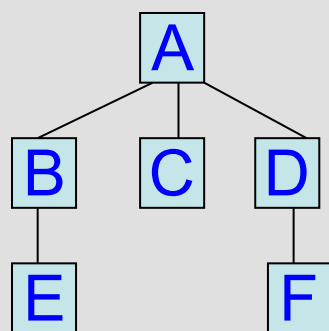
Step2. 编写**驱动器**程序协调输入输出

Step3. 测试特定子功能的构件

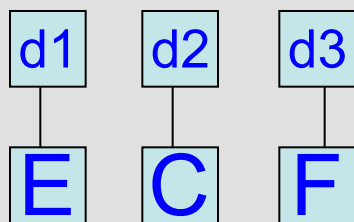
Step4. 沿层次向上对构件进行组合

自底向上

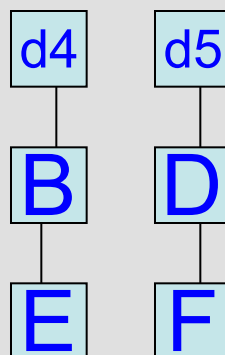
从模块结构的最底层的模块开始组装和测试



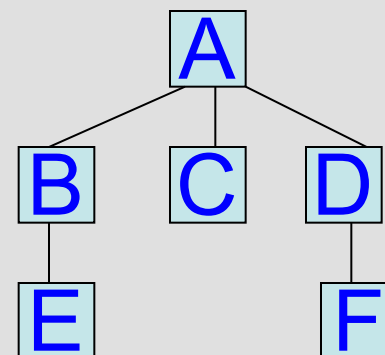
如何测试？



第一步



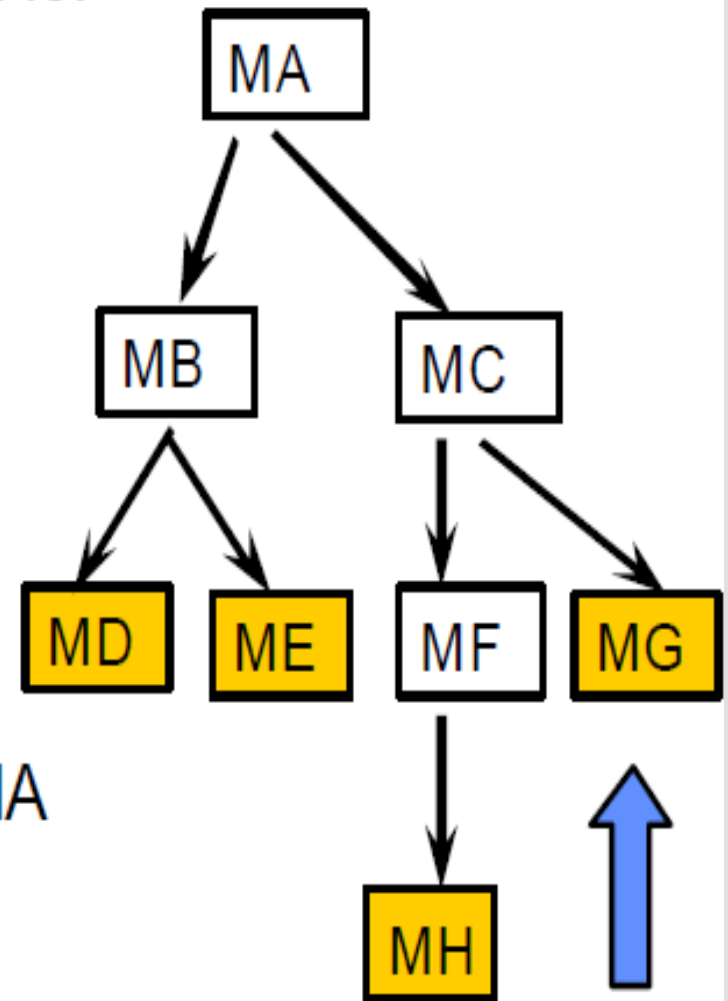
第二步



第三步

A possible sequence of integration is:

- Unit test MD
- Unit test ME
- Unit test MH
- Unit test MG
- Integration test MD, MB
- Integration test MD, MB, ME
- Integration test MD, MB, ME, MA
- Integration test MH, MF
- Integration test MH, MF, MC
- Integration test MH, MF, MC, MG
- Integration test all modules



集成测试策略—增量集成测试

■ 自底向上集成

优点：

1. 尽早确认底层行为
2. 无需编写程序桩
3. 对实现特定功能的树容易表示输入输出

缺点：

1. 推迟确认高层行为
2. 需编写驱动器
3. 组合子树时，有许多元素要进行集成

集成测试策略—增量集成测试

■ 混合式集成

集成顺序：

综合自顶向下和自底向上，是实际测试中的实用集成测试策略。

特点：

开发小组对各自的低层模块向上集成；
专门的集成小组进行自顶向下集成；

集成测试策略—增量集成测试

■混合式集成

步骤：

Step1:用程序桩独立测试上层模块；

Step2:用驱动器独立测试低层模块；

Step3:集成时对中间层进行测试；

集成测试策略—增量集成测试

增量测试策略比较

	优点	缺点
自顶向下	<ul style="list-style-type: none">1.如果主要的缺陷发生在程序的顶层将非常有利2.一旦引入I/O功能，提交测试用例会更容易3.早期的程序框架可以进行演示，并可激发积极性	<ul style="list-style-type: none">1.必须开发桩模块2.桩模块要比最初表现的更复杂3.在引入I/O功能之前，向装模块引入测试用例比较困难4.创建测试环境可能很难，甚至无法实现5.观察测试输出很困难6.会导致特定模块测试的完成延后
自底向上	<ul style="list-style-type: none">1.如果主要的缺陷发生在程序的底层将非常有利2.测试环境比较容易建立3.观察测试输出比较容易	<ul style="list-style-type: none">1.必须开发驱动模块2.直到最后一个模块添加进去，程序才形成一个整体

端到端集成测试

■ 端到端集成测试简介

目的：

验证一组相互连接的系统形成的集成系统是否正常运行

特点：

从最终用户的角度出发，强调系统的功能测试
不关注接口

适用领域：

大型软件系统

集成测试方法选择

集成测试方法选择指南

因素	建议使用集成方法
设计和需求清晰	自顶向下
需求、设计和体系结构不断变化	自底向上
体系结构改变、设计稳定	混合
体系结构较小，规模小	瞬时集成
体系结构复杂，软件规模大	混合、端-端

集成测试总结

集成测试是一个必要的测试阶段：

从将两个组件集成到一起开始，到所有系统组件在一起运行位置的所有测试活动，都是集成测试阶段的一部分

集成测试是一种测试类型：

集成测试测试组件间的接口

集成测试不应被淡化：

集成测试能减少系统测试阶段的缺陷

三 测试插桩



1. 插桩在测试中的作用？
2. 如何在测试中插桩？

测试插桩

■ 测试插桩背景

测试过程中，往往需要知道某些信息：

语句覆盖

路径覆盖

变量的使用、定义

...

传统方法：

跟踪被测程序的执行过程，人工收集信息

测试插桩

■ 测试插桩背景

程序插桩技术最早是由**J.C. Huang**教授提出的,它是在保证被测程序原有逻辑完整性的基础上在程序中插入一些**探针**(又称为“探测仪”),通过探针的执行并**抛出程序运行的特征数据**,通过对这些数据的分析,可以获得程序的控制流和数据流信息,进而得到逻辑覆盖等动态信息,从而实现测试目的的方法。



<http://www2.cs.uh.edu/~jhuang/>

测试插桩

■ 测试插桩基础

原理：

在程序特定部位附加一些操作或功能，
用来检验程序运行结果以及执行特性，
以便支持测试

两类插桩技术：

- (1) 白盒测试插桩技术
- (2) 黑盒测试插桩技术

测试插桩

■ 测试插桩基础—白盒测试插桩

作用：

- (1) 生成特定状态，**检验状态**的可达性；
- (2) 显示或读取内部**数据**或私有数据；
- (3) 监测**不变数据**
- (4) 监测**前提条件**
- (5) 人为**触发事件**
- (6) 监测**事件时间**

测试插桩

■ 测试插桩基础—白盒测试插桩

插桩方式一:目标代码插桩

优点:

- 目标代码的格式主要和操作系统相关, 和具体的编程语言及版本无关, 应用范围广;
- 特别适合需要对内存进行监控的软件中

缺点:

- 目标代码中语法、语义信息不完整

测试插桩

■ 测试插桩基础—白盒测试插桩

插桩方式二:源代码插桩

优点:

- 词法分析和语法分析的基础上进行的，这就保证对源文件的插桩能够达到很高的准确度和针对性。

缺点:

- 工作量较大，而且随着编码语言和版本的不同需要做一定的修改

测试插桩

■ 测试插桩基础—白盒测试插桩

插桩问题1：探测什么信息？

Answer: 由具体需求而定

覆盖率？

中间结果？

测试数据生成？

统计信息？

...

测试插桩

■测试插桩基础—白盒测试插桩

插桩问题2：插桩位置？

Answer:由具体需求而定。

原则：

探针的植入要做到紧凑精干，才能保证在做到收集的信息全面而无冗余，减少代码的膨胀率。

测试插桩

■ 测试插桩基础—白盒测试插桩

探针的植入常见位置：

- a. 程序的**第一条语句**；
- b. **分支语句的开始**；
- c. **循环语句的开始**；
- d. 下一个**入口语句之前**的语句；
- e. 程序的**结束语句**；
- f. **分支语句的结束**；
- g. **循环语句的结束**；

测试插桩

■ 测试插桩基础—白盒测试插桩

插桩问题3：探测点数目？

Answer:个数越少越好

插桩会影响原有程序执行的效率

很多测试工具能根据需求进行自动插桩

测试插桩

■ 测试插桩基础—白盒测试插桩

常用插桩策略：

语句覆盖插桩（基本块插桩）：在基本块的入口和出口处，分别植入相应的探针，以确定程序执行时该基本块是否被覆盖。

分支覆盖插桩：分支由分支点确定。对于每个分支，在其开始处植入一个相应的探针，以确定程序执行时该分支是否被覆盖。

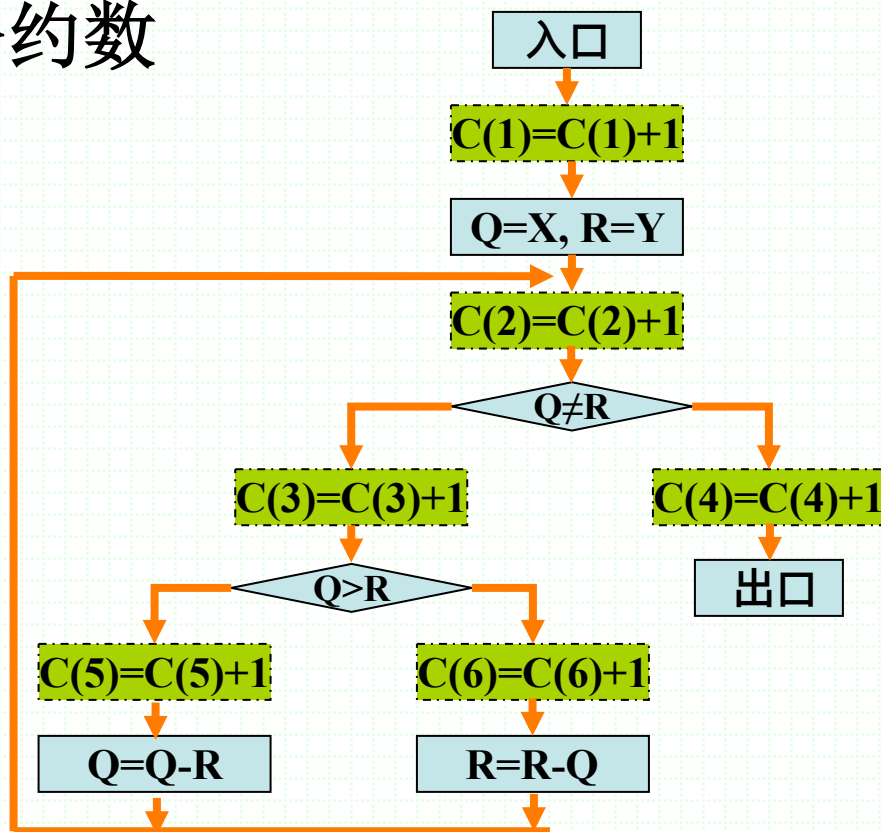
条件覆盖插桩：if, swich, while, do-while, for 几种语法结构都支持条件判定，在每个条件表达式的布尔表达式处植入探针，进行变量跟踪取值，以确定其被覆盖情况。

测试插桩

■ 测试插桩基础—白盒测试插桩

例：分析覆盖率和测试用例有效性

求整数X和Y的最大公约数
程序的插桩



测试插桩

■ 测试插桩基础—白盒测试插桩

例：断言检测

断言：在程序特定部位插入用以证实程序某些特性的语句

```
void *memcpy(void *pvTo, const void *pvFrom, size_t size)
{
    assert((pvTo != NULL) && (pvFrom != NULL));

    byte *pbTo = (byte *) pvTo;

    byte *pbFrom = (byte *) pvFrom;
    while(size -- > 0) *pbTo ++ = *pbFrom ++ ;
    return pvTo;
}
```


测试插桩

■ 测试插桩基础—黑盒测试插桩

作用：

(1) 测试预言 (Test Oracle)

(2) 随机数据生成

测试插桩

■ 测试插桩基础—黑盒测试插桩

测试预言插桩

定义：

一种独立于被测系统的程序或机制，用于确认对于给定的输入，系统是否有一个给定输出。

特点：

不能获得源代码，但可以调用API等信息
不属于纯黑盒测试，而是一种灰盒测试

作用：

检查函数正确性、系统安全性、...

[http://en.wikipedia.org/wiki/Oracle_\(software_testing\)](http://en.wikipedia.org/wiki/Oracle_(software_testing))

测试插桩

■ 测试插桩基础—黑盒测试插桩

例：喂鱼的测试预言

测试预言程序

API和数据结构 定义

```
Public class FeedingView {  
    FeedingSchedule schedule=null;  
    f.end-f.start<=30 minute; //初始化;  
    喂鱼时间不超过30分钟  
    getFeedingSchedule(); //获取时间  
    表  
    updateFeedingSchedule();//更新时  
    间表  
}
```

```
Public boolean  
    testUpdateFeedingSchedule(){  
    FeedingSchedule fs=new  
        FeedingSchedule();  
    Calendar now=Calendar.getInstance();  
    fs.setStart(now);  
    now.add(now.MINUTE,31);  
    fs.setEnd(now);  
    fs.updateFeedingSchedule(fs);  
    return checkInvariant();  
}
```

测试插桩

■ 测试插桩基础—黑盒测试插桩

随机数据生成器（随机测试技术）

定义：

一种在可能输入集中选取一个任意子集进行测试的技术。

作用：

避免只测试所知道的将奏效的场景

测试插桩

■ 测试插桩基础—黑盒测试插桩

有效测试用例生成方法：

- (1) 软件输入域进行等价划分；
- (2) 各子域边界进行边界值选取；
- (3) 各子域内进行随机测试选择输入样本；

测试插桩

■ 测试插桩基础—黑盒测试插桩

例：测试输入为1-100整数的输入框

(1) 等价类划分：

无效域： $x < 1$

有效域： $1 \leq x \leq 100$

无效域： $x > 100$

(2) 边界值：

$\{0, -1\}$

$\{1, 2, 99, 100\}$

$\{101\}$

(3) 随机数据生成：

Java API `random()`

四 单元/集成测试工具



1. 单元和集成测试中的工具？

单元/集成测试工具

■ C/C++语言测试工具

■ 通用各种操作系统的测试工具

C Unit Test System, CPPTest, CPPUnit, CxxTest

■ Win32/Linux/Mac OS X

UnitTest++

■ Mac OS X

ObjcUnit, OCUnit, TestKit

■ UNIX

cutee

单元/集成测试工具

■ C/C++语言测试工具

■ Windows

simplectest

■ 嵌入式系统

Embedded Unit

■ 其它

Cgreen, POSIX Check

单元/集成测试工具

■ Java语言测试工具

■ TestNG

灵活的Java测试框架

■ JUnit

最著名的测试框架之一

■ 其它

PMD, MockRunner, jWebUnit, JSFUnit等

单元/集成测试工具

■ 其它语言测试工具

■ HtmlUnit

Junit扩展的测试框架之一

■ Nunit

C#, .Net

■ 其它

PHPUnit, DUnit, SQLUnit, RSpec等

单元/集成测试工具

■ 商业测试工具

- DevPatner Studio
- Parasoft C++ Test
- Parasoft Jtest
- Parasoft .TEST
- ...

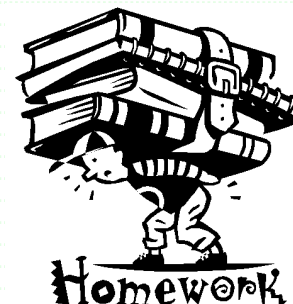
参考文献

1. 郁莲, 软件测试方法与实践, p84-p100, 清华大学出版社
2. Srinivasan Desikan, Gopalaswamy Ramesh, (韩柯, 李娜 译), 软件测试: 原理与实践, p68-p78, 机械工业出版社
3. 朱少民, 软件测试

课后习题



习题将结合实验



本讲总结

■ 单元测试

■ 集成测试

内容预告

■ JUnit

衷心感谢各位老师莅临指导！
欢迎各位老师同学批评指正！

Email: pwang@seu.edu.cn

