软件质量保证与测试

Software Quality Assurance and Testing

第 5 章 软件测试过程

5.1 单元测试

5.1.1 单元测试简介

金陵科技學院

单元测试的成本效率

在软件开发过程中，越早发现问题、解决问题，花费的成本就越小，经验表明一个尽责的单元测试将会在软件开发的早期发现很多的问题和缺陷，并且修改它们的成本也较低，而如果拖到后期阶段，缺陷的发现和修改将会变得更加困难，并要消耗更多的时间和费用。

对于程序代码而言，开发出来之后，应首先对其进行单元测试，力争尽早发现程序中缺陷和问题。

单元测试的成本效率

有统计数据表明，以软件的一个功能点为基准，单元测试的成本效率大约是集成测试的两倍 系统测试的三倍。

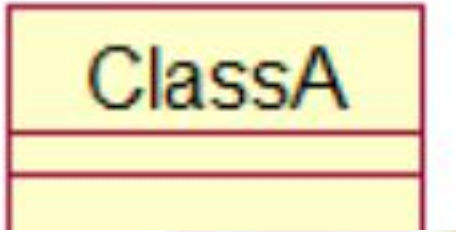
单元测试的概念

单元测试是针对软件设计的最小单位——程序模块，进行检查和验证，其目的在于发现每个程序模块内部可能存在的问题和差错。单元测试通常应用于实施模型中的构件，核实是否已覆盖控制流和数据流，以及构件是否可以按照预期工作。执行单元测试，就是为了验证单元模块代码的行为和我们期望的一致。

单元的粒度

单元测试是对软件基本组成单元的测试。单元的粒度具体划分可以不同，在传统的结构化编程语言如C语言中，单元一般是模块，也就是函数或子过程；在象C++中，单元是类或类的方法；在Ada语言中，单元可为独立的过程、函数或Ada包；在第四代语言(4GL)中，单元对应为一个菜单或显示界面。

```
public int findMin( )
      {    ......
```



*** 打印模块

... ...

单元的粒度

单元测试中的一个可测“单元”应符合以下要求：

(1)是可测试的、最小的、不可再分的程序模块。

(2)有明确的功能、规格定义。

(3)有明确的接口定义，清晰地与同一程序的其他单元划分开来。

单元测试的完成人

单元测试通常是由程序员自己来完成，有时测试人员也加入进来，但编程人员仍会起到主要作用。单元测试是程序员的一项基本职责， 程序员有责任对自己编写的代码进行单元测试。程序员必须对自己所编写的代码保持认真负责的态度，这是程序员的基本职业素质之一。同时单元测试能力也是程序员的一项基本能力，这种能力的高低直接影响到程序员的工作效率与软件的质量。

编程与单元测试

在编码的过程中进行单元测试的话，由于程序员对设计和代码都很熟悉，不需要额外花时间去阅读、理解、分析程序的设计书和源代码，所以测试成本是最小的，测试效率也是最高的。在编码的过程中考虑测试问题，得到的将是更优质的代码，因为此时程序员对代码应该做些什么了解得最清楚。如果不这样做，而是先匆匆忙忙完成模块开发，等过了很久，甚至是一直等到模块报错崩溃了，再来做单元测试，到那时程序员可能已经忘记了这个模块的代码是怎样工作的，需要花较多的时间来重新阅读、理解、分析程序的设计书和源代码。

编程与单元测试

由于单元测试是在编码过程中进行的，若发现了一个错误，修复错误的成本会远小于集成测试阶段，更是小于系统测试阶段。

在编码的过程中考虑单元测试问题，有助于编程人员养成良好的编程习惯，提高源代码质量。

单元测试的目标

单元测试是软件测试的基础。通过单元测试，应分别完成对每个单元的测试任务，确保每个单元模块能正常工作，程序代码符合各种要求和规范。

通常合格的代码应该具备以下性质：正确性、清晰性、规范性、一致性、高效性等，要具备的这些性质中，优先级最高的的正确性，只有先满足正确性，其它特性才具有实际意义，其次是清晰性、规范性和一致性，而对有些会反复执行的代码，还需要具有高效性，否则会影响整个系统的效率和性能。

单元测试的目标

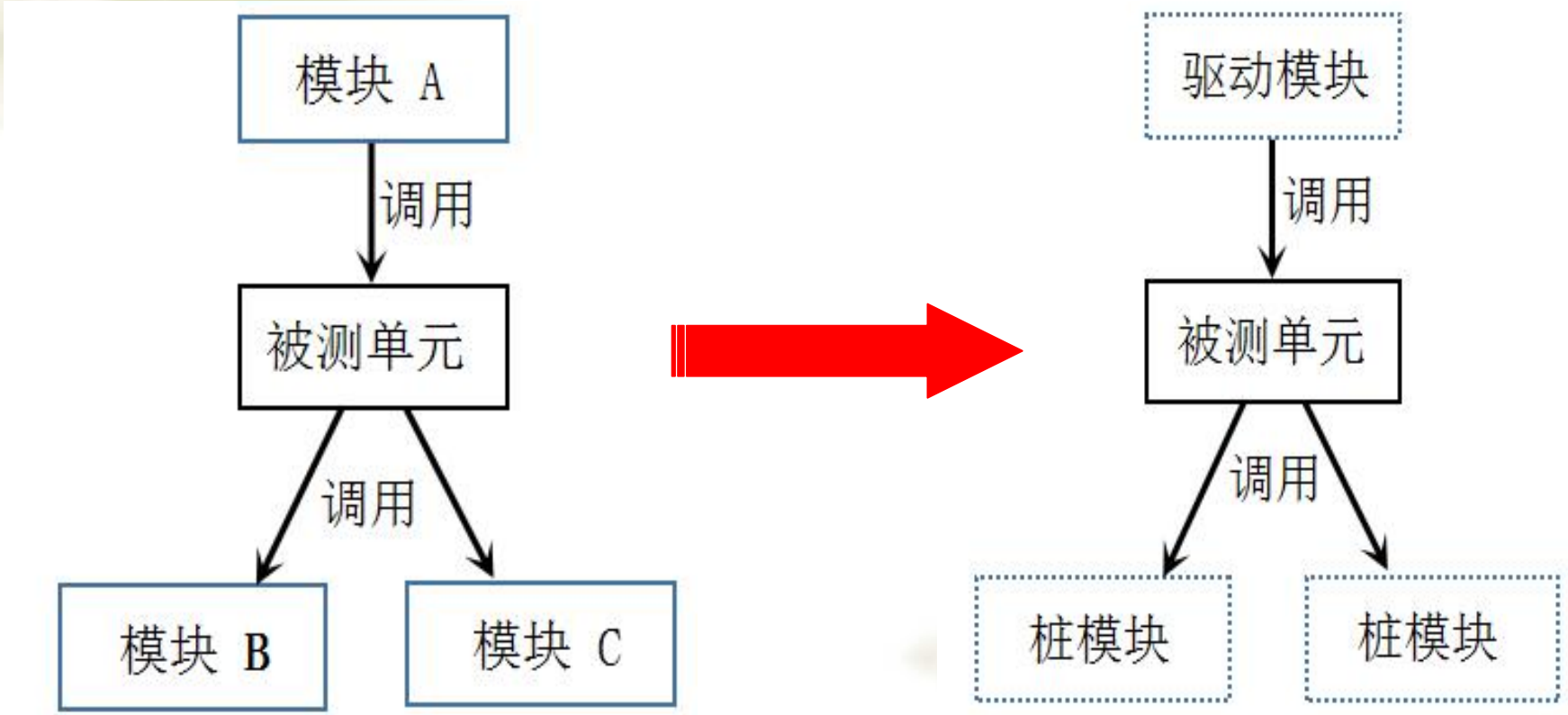
1. 正确性是指代码逻辑必须正确，能够实现预期的功能。
2. 清晰性是指代码必须简明、易懂，注释准确没有歧义。
3. 规范性是指代码必须符合企业或部门所定义的共同规范包括命名规则，代码风格等
4. 一致性指代码必须在名称、风格等方面保持统一
5. 高效性是指代码不但要满足以上性质，而且需要尽可能降低代码的执行时间。

单元测试的辅助模块

一个单元模块或一个方法（Method）等并不是一个独立的程序，在测试它时要同时考虑它和外界的联系，用一些辅助模块去模拟与被测模块相联系的其他模块。这些辅助模块分为驱动模块和桩模块两种。

例如有如图所示的程序模块图，如果要对中间的单元做测试，则需要辅助的一个驱动模块和两个桩模块。

单元测试的辅助模块



驱动模块

驱动模块用来模拟被测单元的上层模块的程序模块，它能够接收或者设置测试数据、参数、环境变量等，调用被测单元，将数据传递给被测单元，如果需要还可以显示或者打印测试的执行结果。

可将驱动模块理解为被测单元的主程序。

桩模块

桩模块又称为存根模块，它用来模拟被测单元的子模块。设计桩模块的目的是模拟被测单元所调用的子模块，接受被测单元的调用，并返回调用结果给被测单元。桩模块不一定需要包括子模块的全部功能，但至少应能模拟满足被测单元的调用需求，而不至于让被测单元在调用它时出现错误。

单元测试的测试环境

被测试的单元模块与它的驱动模块及桩模块一起，共同构成了一个“测试环境”。建立单元测试的环境，需完成以下一些工作：

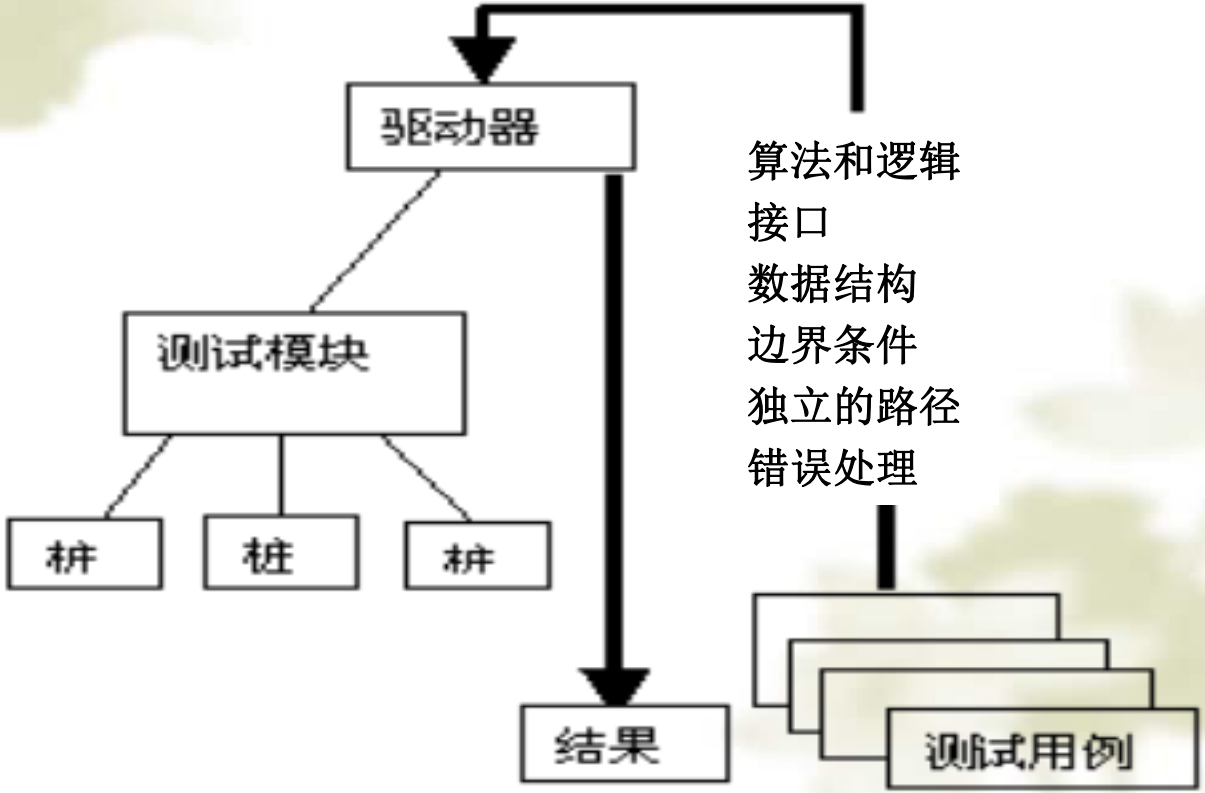
① 构造最小运行调度系统，即构造被测单元的驱动模块。

② 模拟被测单元的下级调用接口，即构造用于提供给被测单元调用的桩模块。

③ 模拟生成测试数据及状态，为被测单元运行准备动态环境。

单元测试模型

单元测试模型如图所示



单元测试的测试环境

驱动模块和桩模块的编写会产生一定的工作量，带来额外的开销。因为它们在软件交付时并不作为产品的一部分一同交付。

特别是桩模块，为了能够正确、充分的测试软件，桩模块可能需要模拟实际子模块的功能，这样桩模块的建立就并不是一件很轻松的事情了。

单元测试的测试环境

有时编写桩模块是非常困难和费时的，但也可以采取一定的策略，来避免编写桩模块，只需在项目进度管理时将实际桩模块的代码编写工作安排在被测模块前编写即可。而且这样可以提高测试工作的效果，因为不断调用实际的桩模块可以更好的对其进行测试，保证产品的质量。

单元测试的测试环境

对于每一个包或子系统我们可以根据所编写的测试用例来编写一个测试模块类来做驱动模块，用于测试包中所有的待测试模块。而最好不要在每个类中用一个测试函数的方法，来测试类中所有的方法。

下面给出一个源代码和测试代码的结构示例。

程序源代码

```
import java.util.Comparator;
import java.util.Random;

public final class Sorting
{
    public void insertionSort( int [ ] a )   {
            .....     }

    public  boolean isSorted(int [] a) {
            .....     }
            .....
}
```

一个测试类和多个测试函数

```
import static org.junit.Assert.*;
import org.junit.Test;
public class TestSorting {
        @Test
        public void testInsertionSort() {
                ......    }
        @Test
        public void testIsSorted() {
                ......    }
    ......
        }
```

单元测试的测试环境

这样的好处在于：

能够同时测试包中所有的方法或模块，也可以方便的测试指定的模块或方法。

能够联合使用所有测试用例对同一段代码执行测试，发现问题。

便于回归测试，当某个模块作了修改之后，只要执行测试类就可以执行所有被测的模块或方法。

单元测试的测试环境

不但能够方便的检查、测试所修改的代码，而且能够检查出修改对包内相关模块或方法所造成的影响，使得因修改而可能引进的错误得以及时被发现；

能够复用测试方法，使测试单元保持持久性，并可以基于已有的测试来编写相关的其它测试。

能够将测试代码与产品代码分开，使代码更清晰、简洁；

可以提高测试代码与被测代码的可维护性。

本节内容就讲到这里，谢谢，再见！

金陵科技學院