# 1.2 PIE模型

## 关于bug的三个概念

（在软件运行的不同时期有不同的含义）

### Fault：

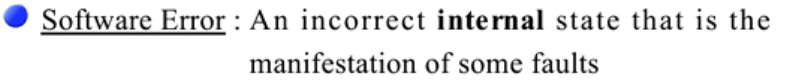
**静态**存在于软件当中的缺陷（由于在coding过程当中犯了个错误写错了，通常称之为缺陷）



（manifestation：表示 显示）

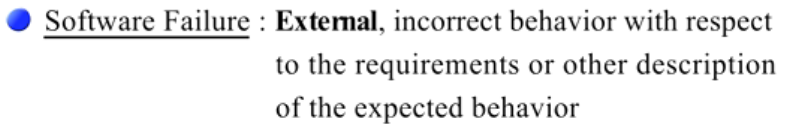
### Error：

在运行过程当中，运行到fault触发了一个错误的中间状态，称之为error



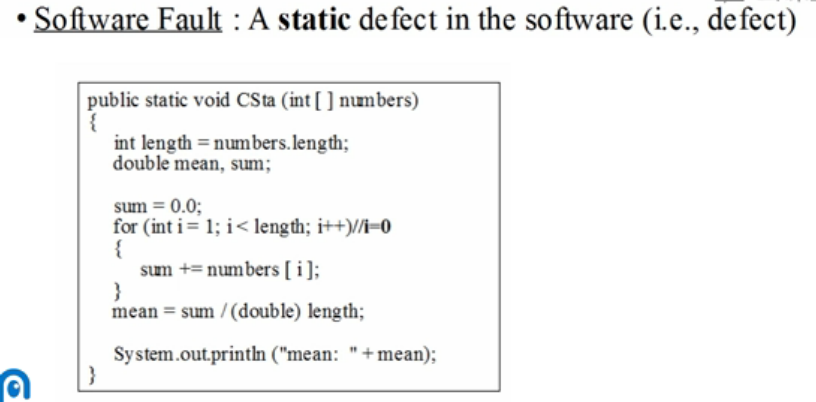
### Failure：

（通常称为失效）是指刚才的error一直传播到软件的外面，使得测试人员或用户能够观测到这个失效的行为

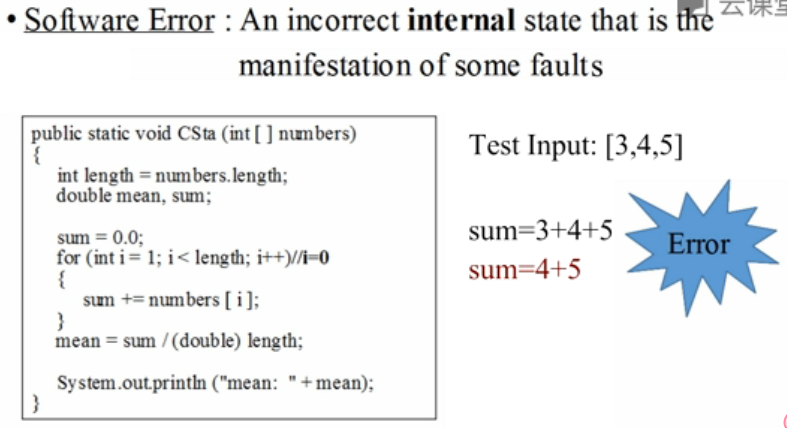


## 用于理解的例子

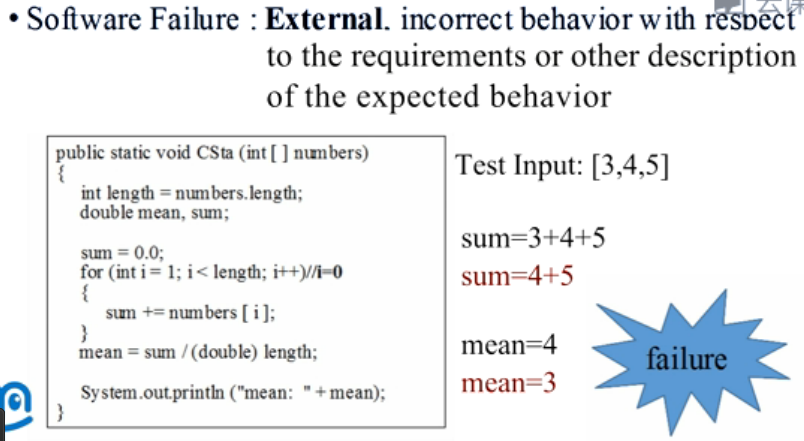
在编码的时候，把循环变量初始值误写成了1——fault



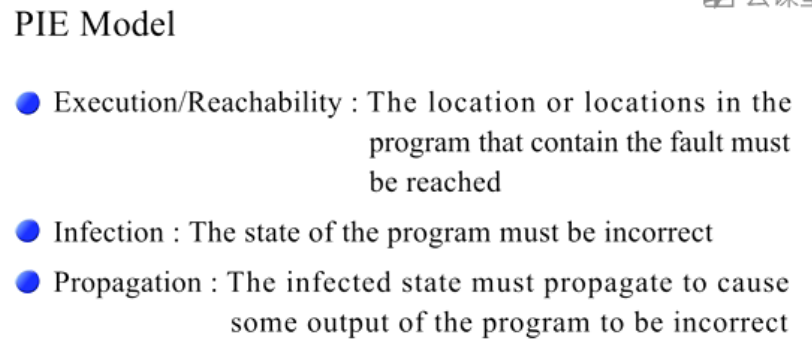
运行程序时，输入[3,4,5]，当执行到循环体时，由于初始条件的错误只累加到了4，5，导致sum现在为9——错误的中间状态



最后这个error传播出去，原来平均值应为4，现在是3，观测到失效的行为，称为failure。



## PIE Model



Propagate：繁衍 扩散 蔓延

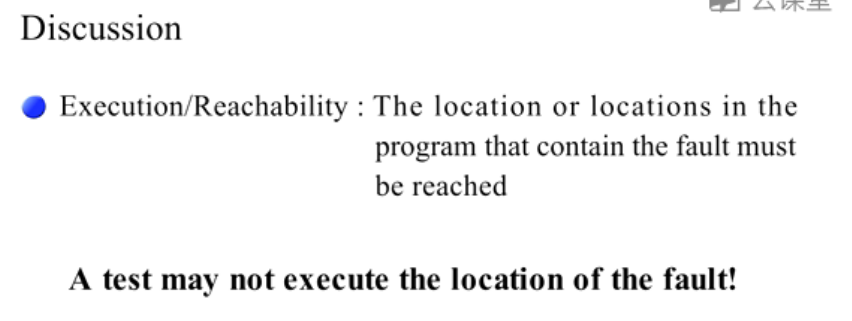
整个运行过程是通过PIE模型来刻画的。

即我们需要观测到一个failure，需要三个必要的条件

1. 执行必须通过错误的代码
2. 感染——在执行这个错误代码的时候，必须触发出一个错误的中间状态
3. 传播——这个错误的中间状态必须传播到最后输出，使得我们能观测到它跟预期的输出不一致，即失效。

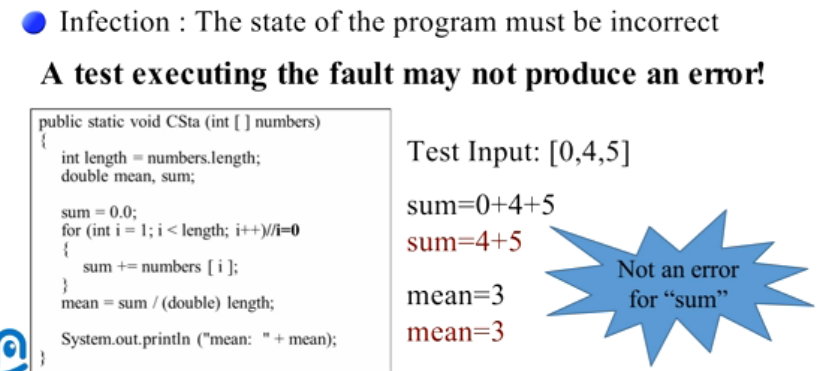
## 对于PIE Model的理解

1. 在很多时候，测试未必能执行到fault的位置。尤其是一些程序的控制流和调用比较复杂的时候。



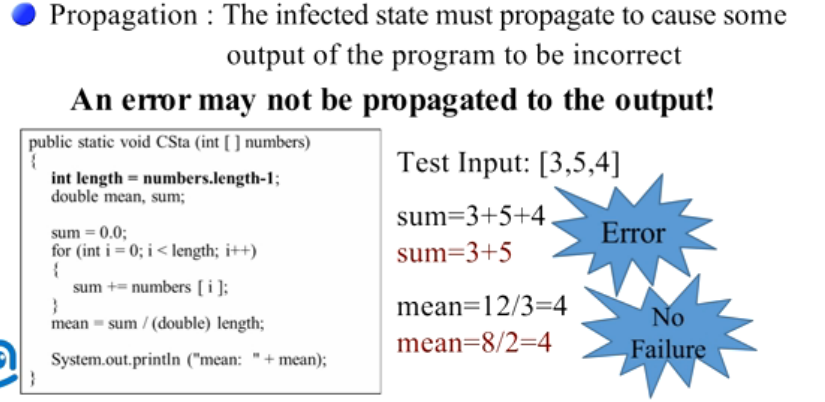
1. 即使测试执行到了错误的代码，也不一定触发error。

这个例子，虽然执行了fault代码，但是对于sum来讲，没有触发error，最终的结果也是正确的。



1. 一个测试，执行到了fault，触发了error，最终也可能没有failure。

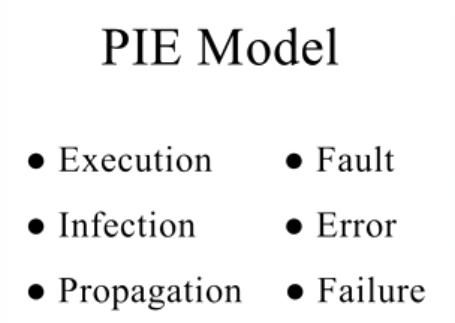
被操作掩盖了当初sum的error，最终没有failure，对于测试人员来讲没有观察到失效的行为，会误判为是对的。

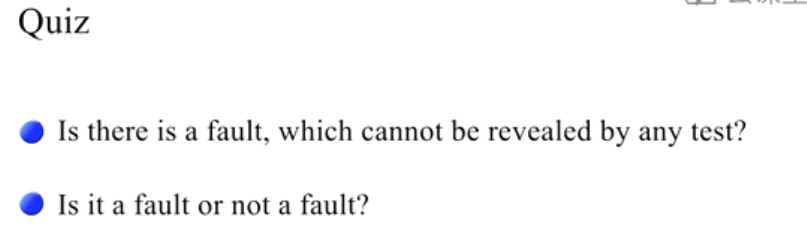


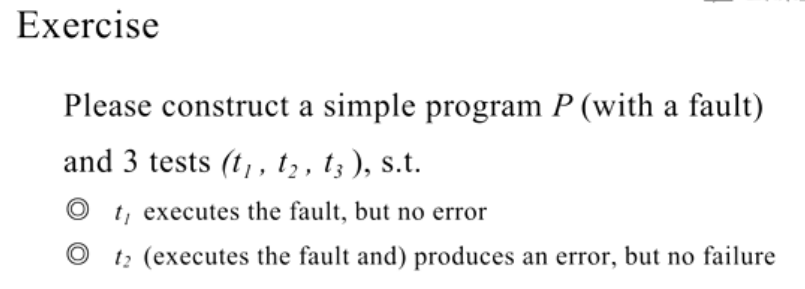
通过执行fault触发error产生失效是一个很复杂的过程。

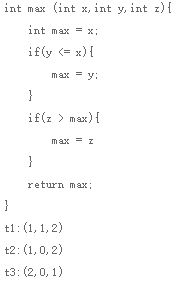
## 总结

Bug分为fault error和failure。整个能够传播出去是靠PIE模型及执行到fault 感染产生error并且传播出去观测到失效的行为。





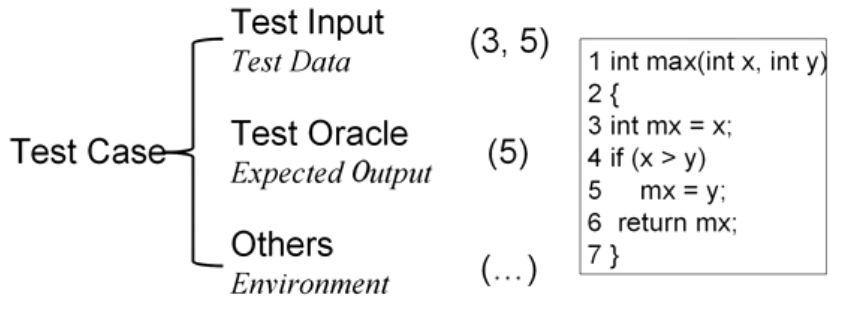




# 1.3 测试术语

## 测试用例

测试用例需要包括测试输入、测试预言即预计的输出，还有其他的一些设置等等，共同构成一个完整的测试用例。但是很多时候讲测试用例会将其等同于测试输入（也就是测试数据），有时候为了叙述方便这样是可以的，但是有时候会带来一些歧义。

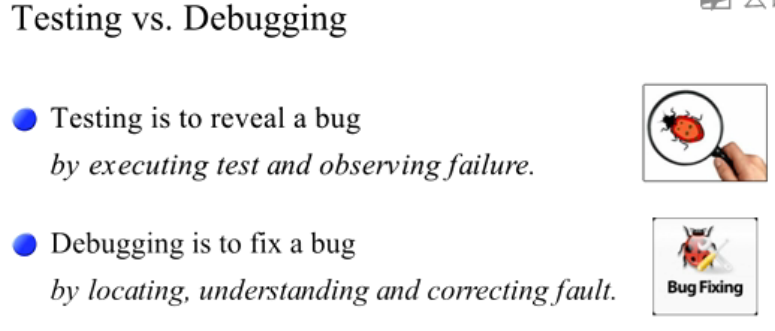


比如上述例子，测试输入是(3,5)测试输出是(5)，测试环境可以是命令行程序、web程序、桌面应用，需要在一个什么样的环境下执行等等，这些共同表达了这个测试用例。

## 测试跟调试

测试是为了发现bug；调试是为了修复这些bug

测试通常是指执行软件去观察是否失效即与预期的行为不一致；调试通常包括去找到fault的位置，理解fault并且修正这个fault。

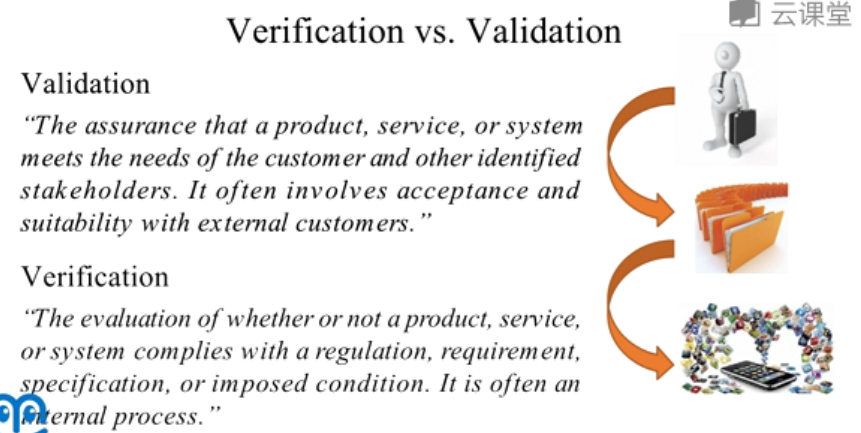


## Validation&verification

validation中文指确认，通常是指我们确认规格文档是否就是最终（比如用户）需要的。

verification是指刚才确认的文档和最终的实现是否一致。也就是实现是否满足此规格文档。

测试就是属于verification，即确认系统与当初的需求是否一致



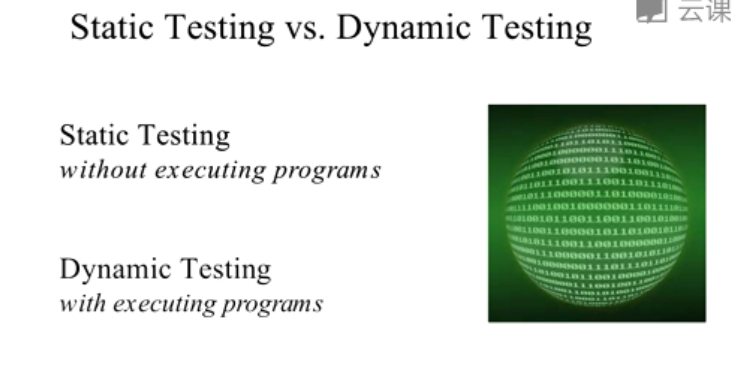
## static testing&dynamic testing

最大的区分点在于是否需要运行程序。

不需要运行程序即静态测试；需要运行即动态测试。

严格来讲，静态测试不是测试，通常通过扫描程序等来判别程序当中可能存在的问题。

动态测试：需要设计一些输入来运行程序，观测结果，作出判断。

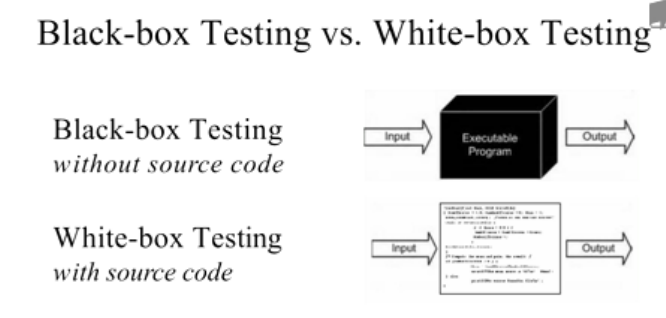


## 黑盒测试跟白盒测试

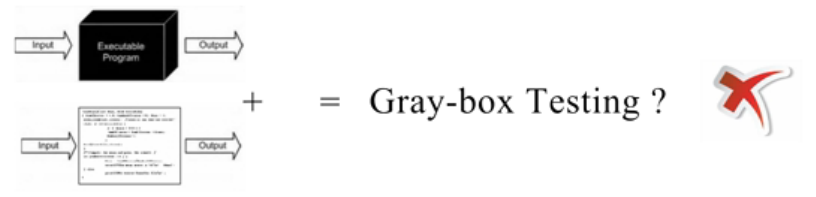
区分点：是否需要源代码。

黑盒：顾名思义，把程序当成一个黑盒子，我们去运行程序，并不需要知道程序的内部信息。

白盒：需要源代码。我们去运行程序，并且通过程序内部的一些结构等信息，去分析这里面可能出现的问题。



介于黑盒和白盒之间，有一种叫灰盒测试（gray-box testing）



~~黑盒加上白盒等于灰盒测试~~，这样的说法是错误的。

灰盒测试是指我们通过其他的软件制品或者通过反编译等手段获得了部分的软件结构信息，进而进行测试。比如在安卓应用当中，通常经过反编译获得其结构信息来设计一些测试，这是一种典型的灰盒测试。

## 测试层次

分为四个层次：单元测试、模块测试、集成测试、系统测试。

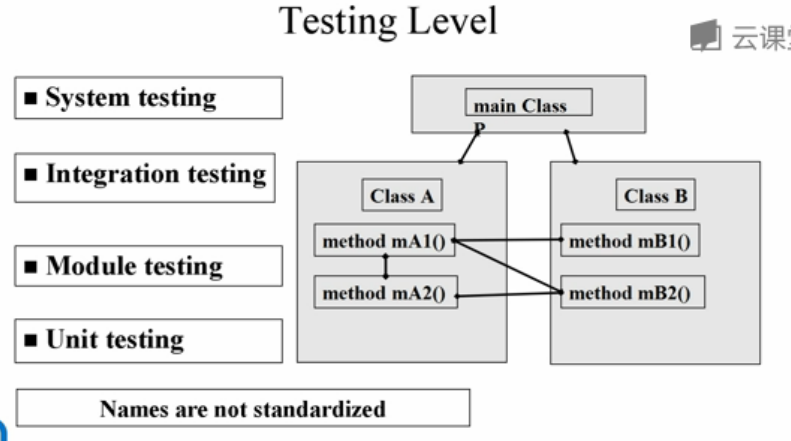
单元测试：最基本的、最小级别的测试。通常去测试一个函数、一个方法等。

模块测试：相对比单元测试大一些。整个模块级的输入输出等等。

集成测试：模块级的一些组合称为集成测试。

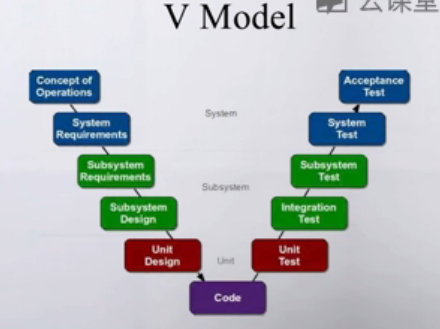
系统测试：整个系统build起来，称为系统级的测试。

（以上并非专业术语）比如单元和模块的级别 很模糊。（比如Google软件测试之道那本书中的小型、中型、大型测试 也不是很清晰）



## V模型

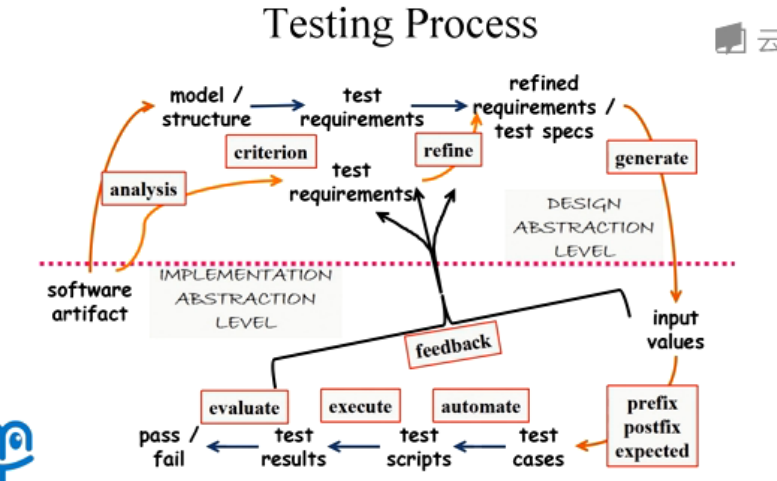
指在整个软件开发过程当中，从需求到设计、详细设计、到编码；反过来再从测试里面有单元测试、模块测试、集成测试、系统测试等。在今天的尤其是移动互联网的开发里面，由于整个开发过程变得短、平、快，这样的V模型可能已经不大适应。



## 测试过程

获得一个软件制品（可能是需求/代码）；通过分析获得某种结构，由此派生测试需求；也可以直接从软件制品当中去派生测试需求；这些测试需求可能会规格化；通过规格化的测试需求去产生测试即测试输入；测试输入再封装成完整的测试用例；为了使测试用例能够自动跑起来，需要将其转变为某种测试脚本，在特定的执行框架下面去执行，获得测试结果。最终的测试结果可以反馈到最初的测试需求，进而提高整个测试过程。

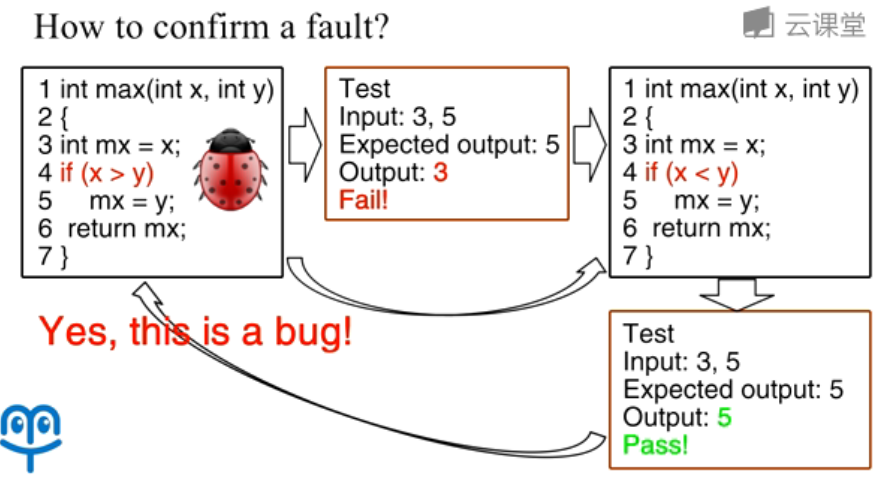
图中的上一层称为测试设计。



# 1.3 fault反思

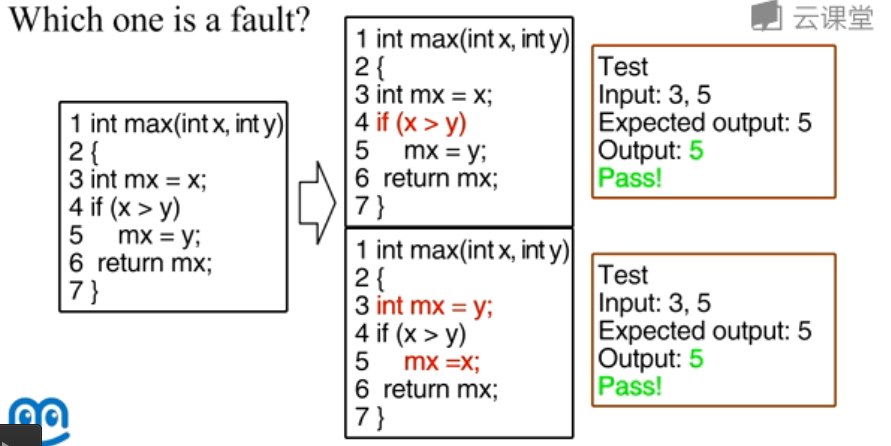
## Fault is defined by fixing ?





Fault似乎是通过修复来定义的——通过修复发现测试从failure到pass，这时才断定这是一个fault。

进一步举个例子：



一个程序员通过修改第四行通过测试，其认为第四行是一个fault。另一个通过修改3 5行也解决了问题，其认为3 5行是fault。所以到底哪一行才是fault？（由于不同的修复，带来了不同的fault）

Q：到底什么是fault，它还是静态存在于代码当中吗？还是需要通过测试、修复等一些动态的操作来确认？

通常程序员认为，一个极小的修复更像一个fault

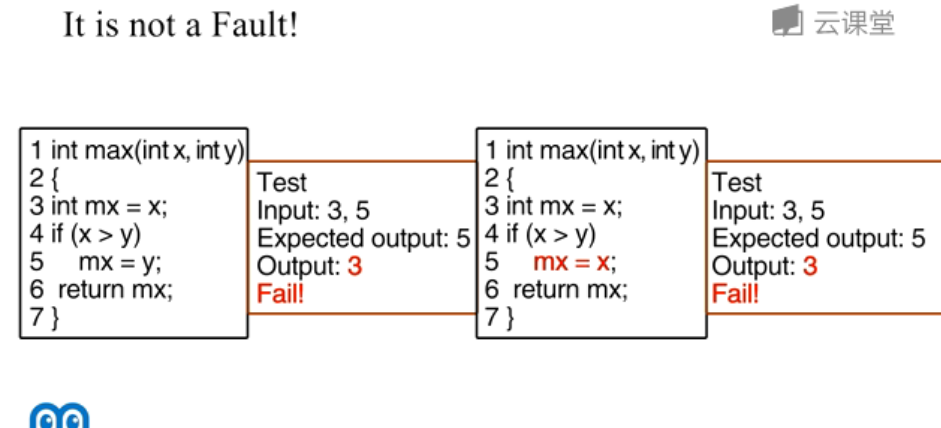
## Fault is defined by testing ?

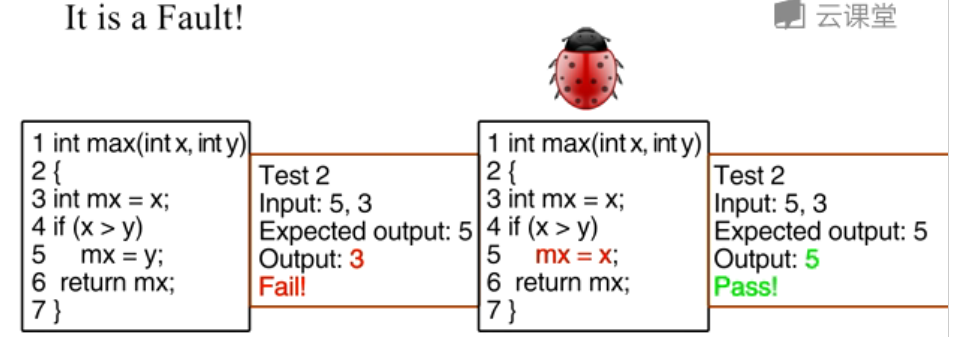
进一步：fault好像是通过测试来定义的

第一次修改第五行后发现依然错，因此程序员断定第五行不是一个fault

第二次的测试输入变了，修复后使原来的failure变成了pass，因此程序员可能会判定第五行是一个fault

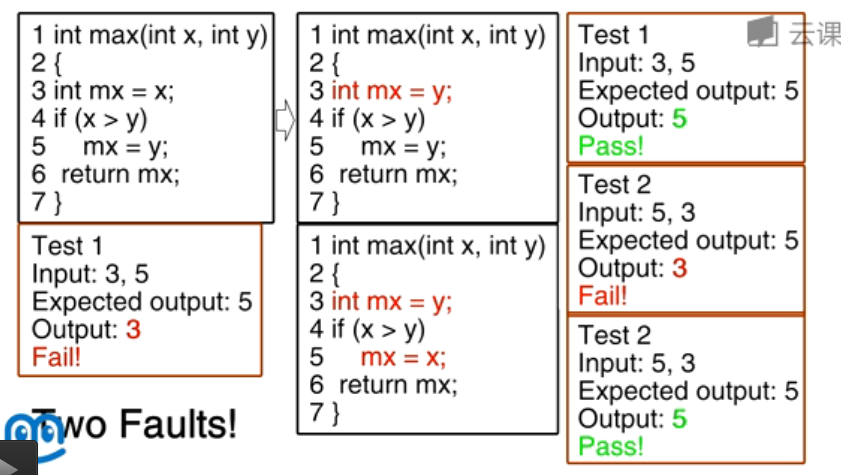
——即使是一个相同的修复，由于不同的测试，也可能使得测试人员断定fault与否的结果不一样。



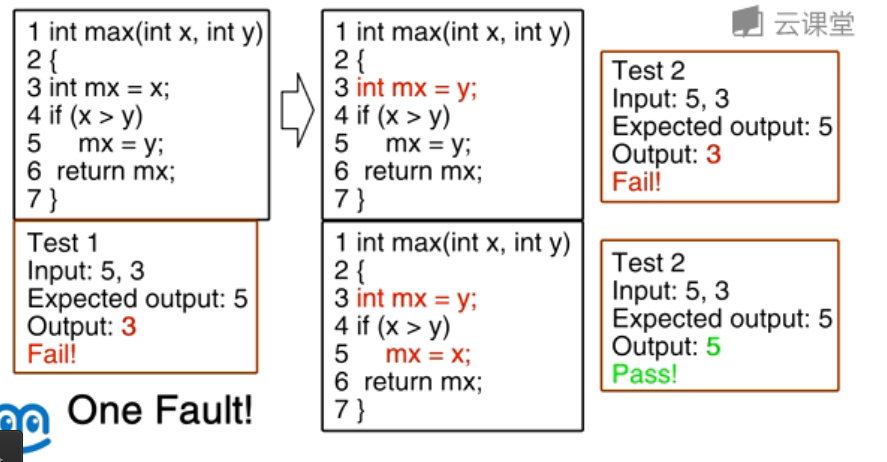


## One or two ?

有时候很难判定到底是一个fault还是两个fault：修改第三行后第一个测试通过，第二个测试failure——是否还存在其他的fault？于是修改了第五行，发现两个都pass了。



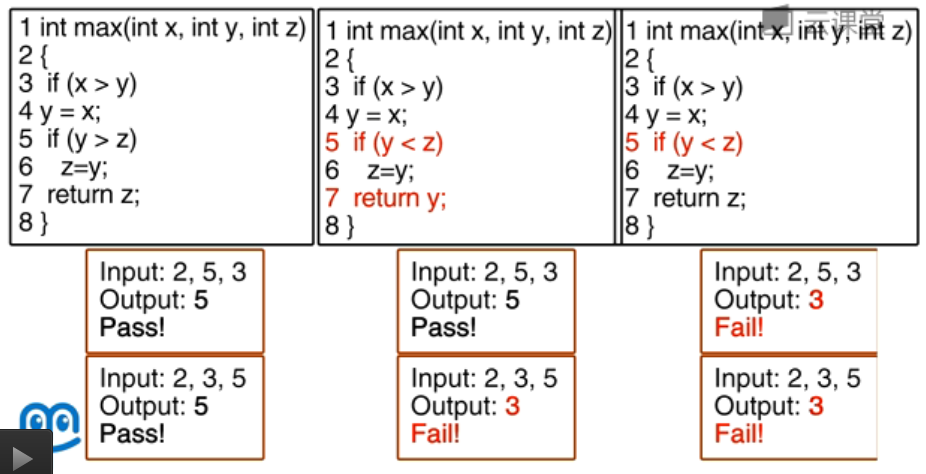
而对于另外一个程序员：修改第三行，发现依然错误，程序员可能认为自己修改对了只是修改的不够完整，因此继续修改了第五行代码，发现pass——断定3.5两行代码共同构成了一个fault。——会判断为这是一个fault。



## Fault interference

Fault之间的干扰

第二个例子中，第二个fault干扰了第一个fault，使得原来的failure变成了pass



## 总结

我们并不明白什么叫bug（这里的fault）——nobody knows.