**基本介绍**

在一种传统的结构化编程语言中，比如C，要进行测试的单元一般是函数或子过程。在像C++这样的面向对象的语言中，要进行测试[2]的基本单元是类。对Ada语言来说，开发人员可以选择是在独立的过程和函数，还是在Ada包的级别上进行单元测试。单元测试的原则同样被扩展到第四代语言（4GL)的开发中，在这里基本单元被典型地划分为一个菜单或显示界面。

经常与单元测试联系起来的另外一些开发活动包括代码走读（Codereview)，静态分析（Staticanalysis)和动态分析（Dynamicanalysis)。静态分析就是对软件的源代码进行研读，查找错误或收集一些度量数据，并不需要对代码进行编译和执行。动态分析就是通过观察软件运行时的动作，来提供执行跟踪，时间分析，以及测试覆盖度方面的信息。

**详细步骤**

单元测试（模块测试）是开发者编写的一小段代码，用于检验被测代码的一个很小的、很明确的功能是否正确。通常而言，一个单元测试是用于判断某个特定条件（或者场景）下某个特定函数的行为。例如，你可能把一个很大的值放入一个有序list中去，然后确认该值出现在list的尾部。或者，你可能会从字符串中删除匹配某种模式的字符，然后确认字符串确实不再包含这些字符了。 单元测试是由程序员自己来完成，最终受益的也是程序员自己。可以这么说，程序员有责任编写功能代码，同时也就有责任为自己的代码编写单元测试。执行单元测试，就是为了证明这段代码的行为和我们期望的一致。 工厂在组装一台电视机之前，会对每个元件都进行测试，这，就是单元测试。 其实我们每天都在做单元测试。你写了一个函数，除了极简单的外，总是要执行一下，看看功能是否正常，有时还要想办法输出些数据，如弹出信息窗口什么的，这，也是单元测试，把这种单元测试称为临时单元测试。只进行了临时单元测试的软件，针对代码的测试很不完整，代码覆盖率要超过70%都很困难，未覆盖的代码可能遗留大量的细小的错误，这些错误还会互相影响，当BUG暴露出来的时候难于调试，大幅度提高后期测试和维护成本，也降低了开发商的竞争力。可以说，进行充分的单元测试，是提高软件质量，降低开发成本的必由之路。 对于程序员来说，如果养成了对自己写的代码进行单元测试的习惯，不但可以写出高质量的代码，而且还能提高编程水平。 要进行充分的单元测试，应专门编写测试代码，并与产品代码隔离。我认为，比较简单的办法是为产品工程建立对应的测试工程，为每个类建立对应的测试类，为每个函数（很简单的除外）建立测试函数。首先就几个概念谈谈我的看法。 一般认为，在结构化程序时代，单元测试所说的单元是指函数，在当今的面向对象时代，单元测试所说的单元是指类。以我的实践来看，以类作为测试单位，复杂度高，可操作性较差，因此仍然主张以函数作为单元测试的测试单位，但可以用一个测试类来组织某个类的所有测试函数。单元测试不应过分强调面向对象，因为局部代码依然是结构化的。单元测试的工作量较大，简单实用高效才是硬道理。 有一种看法是，只测试类的接口（公有函数），不测试其他函数，从面向对象角度来看，确实有其道理，但是，测试的目的是找错并最终排错，因此，只要是包含错误的可能性较大的函数都要测试，跟函数是否私有没有关系。对于C++来说，可以用一种简单的方法区隔需测试的函数：简单的函数如数据读写函数的实现在头文件中编写（inline函数），所有在源文件编写实现的函数都要进行测试（构造函数和析构函数除外）。

**测试效果**

我们编写代码时，一定会反复调试保证它能够编译通过。如果是编译没有通过的代码，没有任何人会愿意交付给自己的老板。但代码通过编译，只是说明了它的语法正确；我们却无法保证它的语义也一定正确，没有任何人可以轻易承诺这段代码的行为一定是正确的。 幸运的是，单元测试会为我们的承诺做保证。编写单元测试就是用来验证这段代码的行为是否与我们期望的一致。有了单元测试，我们可以自信的交付自己的代码，而没有任何的后顾之忧。 什么时候测试？单元测试越早越好，早到什么程度？XP开发理论讲究TDD，即测试驱动开发，先编写测试代码，再进行开发。在实际的工作中，可以不必过分强调先什么后什么，重要的是高效和感觉舒适。从老纳的经验来看，先编写产品函数的框架，然后编写测试函数，针对产品函数的功能编写测试用例，然后编写产品函数的代码，每写一个功能点都运行测试，随时补充测试用例。所谓先编写产品函数的框架，是指先编写函数空的实现，有返回值的随便返回一个值，编译通过后再编写测试代码，这时，函数名、参数表、返回类型都应该确定下来了，所编写的测试代码以后需修改的可能性比较小。 由谁测试？单元测试与其他测试不同，单元测试可看作是编码工作的一部分，应该由程序员完成，也就是说，经过了单元测试的代码才是已完成的代码，提交产品代码时也要同时提交测试代码。测试部门可以作一定程度的审核。 关于桩代码，单元测试应避免编写桩代码。桩代码就是用来代替某些代码的代码，例如，产品函数或测试函数调用了一个未编写的函数，可以编写桩函数来代替该被调用的函数，桩代码也用于实现测试隔离。采用由底向上的方式进行开发，底层的代码先开发并先测试，可以避免编写桩代码，这样做的好处有：减少了工作量；测试上层函数时，也是对下层函数的间接测试；当下层函数修改时，通过回归测试可以确认修改是否导致上层函数产生错误。

**行为优点**

1、它是一种验证行为。 程序中的每一项功能都是测试来验证它的正确性。它为以后的开发提供支援。就算是开发后期，我们也可以轻松的增加功能或更改程序结构，而不用担心这个过程中会破坏重要的东西。而且它为代码的重构提供了保障。这样，我们就可以更自由的对程序进行改进。 2、它是一种设计行为。 编写单元测试将使我们从调用者观察、思考。特别是先写测试（test-first），迫使我们把程序设计成易于调用和可测试的，即迫使我们解除软件中的耦合。 3、它是一种编写文档的行为。 单元测试是一种无价的文档，它是展示函数或类如何使用的最佳文档。这份文档是可编译、可运行的，并且它保持最新，永远与代码同步。 4、它具有回归性。 自动化的单元测试避免了代码出现回归，编写完成之后，可以随时随地的快速运行测试。