一个自动构造类测试驱动程序的框架*)

姜 文 缪淮扣 刘 玲
(上海大学计算机工程与科学学院 上海200072)

摘 要 在类测试时,需要驱动程序来运行可执行的类测试用例并收集测试结果,因此驱动程序的开发工作量将直接影响类测试的效率。本文给出了一种类测试驱动程序的构造框架。利用面向对象技术的设计思想,设计了一个驱动基类,通过继承驱动基类,并覆盖驱动基类中的虚函数而生成一个被测类的驱动类。同时在主控程序中注册被测试类的驱动类,从而得到被测试类的驱动程序。最后通过一个实例,说明该驱动构造框架的可行性。

关键词 测试驱动,软件测试,类测试

An Framework of Automated Class Test Driver Program Construction

JIANG Wen MIAO Huai-Kou LIU Ling
(School of Computer Engineering and Science, Shanghai University, Shanghai 200072)

Abstract The test driver must be constructed to run test case and collect test result at the class test level. So the cost of constructing a driver will have direct influence on the class test. This paper presents a framework to construct class test driver automatically. The design of the framework adopts the Object oriented Polymorphism. In the framework, firstly, by inheriting the TestDriver and overriding the virtual function of the TestDriver, the driver class of CUT(class under test)will be produced; and then the driver of the CUT will be registered in the main control program. Finally, the driver of the CUT will be constructed correctly. In the end, this paper presents an instance to show the feasibility of the framework.

Keywords Test driver, Software test, Class test

1 引言

软件复杂度越来越高,软件可靠性和软件质量的控制也越来越难。长期的软件生产实践证明,软件测试仍是软件质量保障中的关键技术。软件测试的策略是从"小型测试"开始,逐步走向"大型测试。"目前软件测试大致分为四个测试阶段:单元测试、集成测试、确认测试、系统测试。面向对象软件的类测试是和传统软件的单元测试相对应的。和传统的单元测试不一样,类测试主要考察封装在类中的属性和方法的相互作用,因此对类的测试实际上是测试类的实例对象的状态和动态行为。在类测试的过程中,常常需要设计测试驱动程序来运行测试用例。大多数的测试生成方案都会产生巨大数量的测试用例,如果没有测试驱动程序,那么这些测试用例根本无法运行。另外,要执行回归测试也必须要有测试驱动程序,因为回归测试需要一次又一次地来运行被测软件。事实上无论是白盒测试还是黑盒测试,都需要运行测试用例并给出测试结果,因而测试驱动程序的设计是软件测试的一个极其重要的部分。驱动程序的设计是一项耗时的工作,甚至在某些情况下,为某个类构造一个测试驱动程序所需要的工作量可能比开发一个类所需要的工作量还要大,这将大大影响类测试的效率。针对这一问题,本文提出了一种类测试驱动程序的生成框架。通过该驱动程序生成框架可以高效、准确地生成测试驱动程序,从而提高类测试执行的效率,节省测试的时间。

2 测试用例生成与测试驱动

一个测试过程可细分为四个独立的步骤,如图1所示。

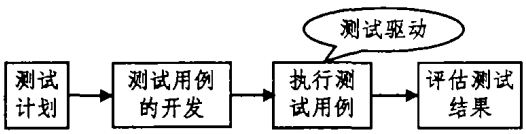

图1 测试过程模型

2.1 测试用例的生成

测试用例的生成位于图中的第二个步骤。测试用例通常指测试输入数据和期望的输出。测试用例是在一定的测试准则指导下而产生的。

我们研究的是基于Object-Z形式化规格说明的测试方法。其主要工作是根据测试(覆盖)准则,从基于模型的形式化的类规格说明自动产生类内部测试用例。类内部测试准则本质上是一种利用方法内前、后置条件的关系构造类内部测试用例的技术。这种技术不需要类的状态图,因而易于自动构造类测试用例,从而提高测试的效率。

在文[3]中,我们定义了一种方法内前后置条件的关系—相连关系,即前一个方法内测试用例的后置条件蕴涵后一个方法内测试用例的前置条件。

我们提出了三个类内部测试准则:

全测试覆盖(ATC):类内部测试用例集包括所有的方法

*)该课题受到国家自然基金(编号:60173030)的资助。姜 文 硕士,主要研究方向为软件测试;缪淮扣 教授,博导,主要研究方向为软件工程、软件形式方法、自动推理;刘 玲 博士,主要研究方向为软件测试、软件形式方法。

内测试用例。

全无环有效序列覆盖(ALES):类内部测试用例集必须包含所有方法内测试用例的无环有效测试序列。

最大无环有效序列覆盖(MLES):类内部测试用例集必须包含所有的方法内测试用例的最大无环有效测试序列。其中的有效测试序列是一种特殊的相连测试序列$\langle t_1, t_2, \cdots, t_n \rangle$,其中$t_1$是一个类的 INIT 操作的测试用例,$t_n$不能被其它的测试用例相连或广义相连。

从形式规格说明推导用于测试多态关系的测试用例时,需要考虑两种情况:

1)一个操作中包含对多态性对象的操作。2)一个类的状态变量是多态对象。对于第一种情况,我们在范畴划分测试的基础上提出了所有有效多态选择组合标准、每个有效多态选择准则和基本有效多态选择覆盖准则来指导测试用例的生成。对于第二种情况,采用对三个类内部测试准则的改进而得到的多态关系的三个测试准则,分别为:全多态测试覆盖准则(APTC),全多态无环有效序列覆盖准则(APLESC)和最大多态无环有效序列覆盖准则(MPLESC)。

2.2 测试驱动

所谓测试驱动是指运行可执行的测试用例并给出运行测试结果的程序。

测试驱动则位于测试过程的第三步骤。从图中可以看出,在测试用例产生后,需要执行测试用例,这就需要开发测试驱动程序,否则,将无法运行测试用例。另外,要执行回归测试也必须设计测试驱动程序,因为回归测试需要一次又一次地来运行被测软件。由此可见,测试用例生成以后,测试驱动程序成为能否在规定的时间内准确地完成测试以及能否在有效的时间内进行足够而充分测试的一个关键因素。

本文对测试驱动程序的研究是在前面的测试准则指导下产生的测试用例的基础之上进行的。本文所提出的测试驱动的生成框架,不仅适用于用文[3]中的方法构造的类测试用例,对于按照其它准则所生成的类测试用例也同样适用。

3 已有的工作

测试驱动分为单元测试驱动和系统测试驱动,它们分别表示运行单元测试用例和运行系统测试用例。

目前,系统级的测试用例的执行是通过测试工具来自动完成的。如,Rational 公司的 VisualTest 可以根据测试人员编写的测试脚本自动运行指定的测试用例并记录测试结果;还有人们所熟悉的捕获/回放工具,它把用户在进行测试时的操作记录下来,同时把软件的响应记录下来,然后通过回放功能实现测试的自动执行。

但如何自动运行面向对象的类测试(单元测试)用例,现在还没有一个理想的解决方案。Alkadi 和 Doris[2]提出了一种面向对象软件测试驱动生成的方法。该方法是针对类中的每一个方法自动地产生相应的驱动。首先将软件中的方法进行分类,分别分成对象级方法、继承方法、接口方法。然后针对每一类方法产生相应的驱动。John D[4]提到测试驱动程序的设计应该相对简单,因为很少有时间和资源来对驱动程序软件进行基于执行的测试,而主要依据代码检查来测试驱动程序。针对这种需求,他总结了三种编写驱动程序的方法。第一种方法是采用有条件编译驱动程序,即将驱动的代码写在一个类的实现函数中,并有条件地编译驱动程序的代码,从而使之执行,这种方法能够比较容易地控制驱动程序的执行。第二种方法是,在类中实现一个静态成员函数,该静态成员函数完成用例的执行,然后调用这个函数,使之执行。第三种方法,是实现单独的类,在主函数中,将这个类实例化,并向其发送一个消息使之执行,这种方法使得驱动代码容易被复用。

4 测试驱动程序构造框架的设计

为了能够快速有效地生成驱动程序,使生成的驱动程序简单并且易于维护,本文在已有的研究基础之上,提出了一种驱动程序构造框架(DriverFramework)。并结合实例说明了如何利用该框架构造驱动程序。

该驱动程序构造框架的主要设计思想采用了面向对象技术中的多态性思想。将类的驱动用一个单独的类来实现。在该框架中,首先,测试驱动基类(TestDriver)能够为所有(具体的)测试人员共用的操作提供默认的实现,其次,通过继承测试基类并覆盖其中的虚函数而产生一个被测类的驱动类,然后在主控程序中注册被测类的驱动类,最后,将被测类、被测类的驱动类和主控程序一起编译产生可执行的应用程序从而驱动测试用例的运行。

下面将详细介绍测试驱动程序生成框架的设计。本文提出的驱动程序构造框架模型如图2所示。

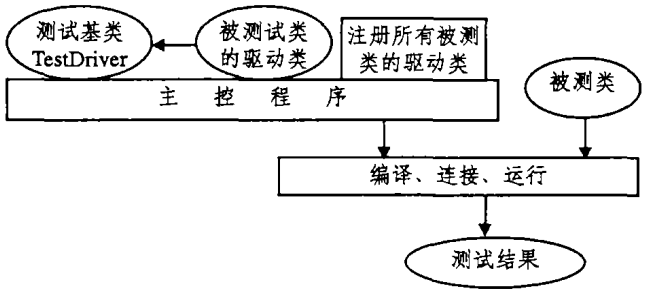

图2 驱动构造框架模型

4.1 测试基类 TestDriver

利用面向对象中的多态性的设计思想,我们设计了一个测试基类 TestDriver。TestDriver 类模型如图3所示。TestDriver 类是抽象的,这个类的代码能够为所有测试人员共用的操作提供默认的实现。这些操作包括:打开用例文件,分析用例文件,创建保存测试结果的文件等功能,同时提供了运行测试用例的界面 *RunTest*()。

其中的 *ResultFile*,*CaseFile* 分别表示了测试结果的文件和测试用例文件。MethodTests 用于存放每个测试用例的内容,*iter* 表示指向每个测试用例内容的指针。

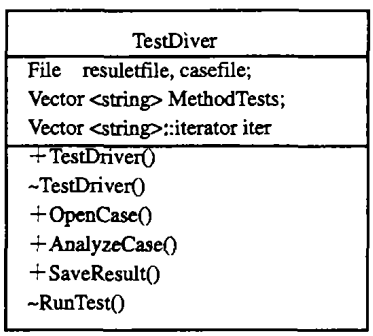

图3 TestDriver 类模型

4.2 被测类的驱动类

对于不同的被测试类(Class Under Test,简称 CUT),我们可以通过继承基类 TestDriver 来驱动它,并通过覆盖

RunTest()来实现之(CUT 的驱动模型如图4)。对于 *RunTest*()主要由两个部分组成:一个是运行测试用例,一个是运行测试预测程序(Testoracle)。对于一个具体的测试驱动类(驱动类在本文中是指驱动程序的一种存在形式),它的 *RunTest*()的实现是不同的,因为不同的 CUT,测试用例是不同的,当然它的测试预测结果也是不同的。因此在图3的 CUT 的驱动模型中,CUT 的 *runtest*()和 TestDriver 的 *runtest*()的实现是不同的。

对于被测类的 *RunTest*()函数可用算法来实现。算法如下:

a. 从测试用例文件里读出第一个测试用例
b. 取出测试用例文件里每个测试用例中的被测类
c. 建立该类的实例
```
for (从测试用例的开始 to 测试用例的结束 )
    {
        执行当前所指的测试用例中的方法
        执行该方法的测试预测函数
    }
```
d. 重复 a,b,c 直到测试用例文件结束

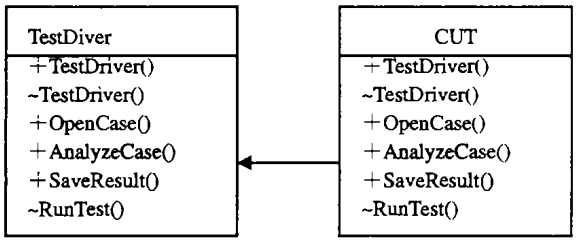

图4 CUT 的驱动模型

4.3 主控程序

主控程序是实现运行测试用例的入口。在这里,主控程序的实现是非常简单的。它主要采用有条件编译的方法来实现对被测试类(CUT)的注册。所谓类注册,其实质是创建一个被测类的驱动类实例。如果有多个 CUT,就会有多个相应的驱动类,并将为每个驱动类创建一个实例。我们用一个 C++ 程序段的例子来说明。假如有一个被测试类的驱动类为 TetrisTesterDriver,则注册后在主控程序中得到代码:

```
#ifdef TetrisTesterDriver
      estDriver *r1=new Tetris
      r->Runtest()
#end if
```

这样通过条件编译可以实现多个被测类(CUT)在一个主控程序下完成测试。

4.4 编译、连接、运行

编译、连接、运行主要完成的任务是将被测类、被测类的驱动类和主控程序一起编译连接产生可执行的应用程序,从而驱动测试用例的运行。因为,在类测试层次,被测类是不可执行的,所以要求主控程序、测试驱动类和被测类一起编译链接产生一个可执行的应用程序,从而驱动测试用例运行,完成测试。例如:在 C++ 中,我们可以调用 CL 编译器,通过设置参数,灵活地实现编译连接。

其中我们将编译连接的参数设置和连接参数的设置以 compileset.txt,linkset.tx 的形式存放,我们只需要执行 cl @compileset.txt,link @linkset.txt 命令就可以编译连接产生一个可执行的程序,也即是测试驱动程序。

5 实例研究

本文采用 C++ 来说明测试驱动生成框架。

本文以"俄罗斯方块"游戏为例,来说明类的驱动程序的生成。*Tetris* 类用于说明这个游戏系统。该类的接口包括三个状态变量和六个操作。其中操作 *MoveRight*、*MoveLeft* 和 *Rotate* 说明了操作者右、左和旋转下降的方块动作;另外三个操作 *MoveDown*、*AddBlock* 和 *RemoveRow* 分别说明了一个方块向下移动一行、在屏幕上新添一个方块和从屏幕上消去完全被覆盖的一行的行为。这些操作的定义都是借助于一个多态的状态变量 *block* 来完成的。在 *Tetris* 类中,多态对象 *block* 的可能取值有七种,分别是:*Square*、*Rectangle*、*T*、*S*、*Z*、*L*、*ReverseL*。因而,*Context*(*Tetris*)中应该包含八个类定义。

在 Tetris 系统[3]中,共有11个类,这些类之间的继承关系如图5所示。

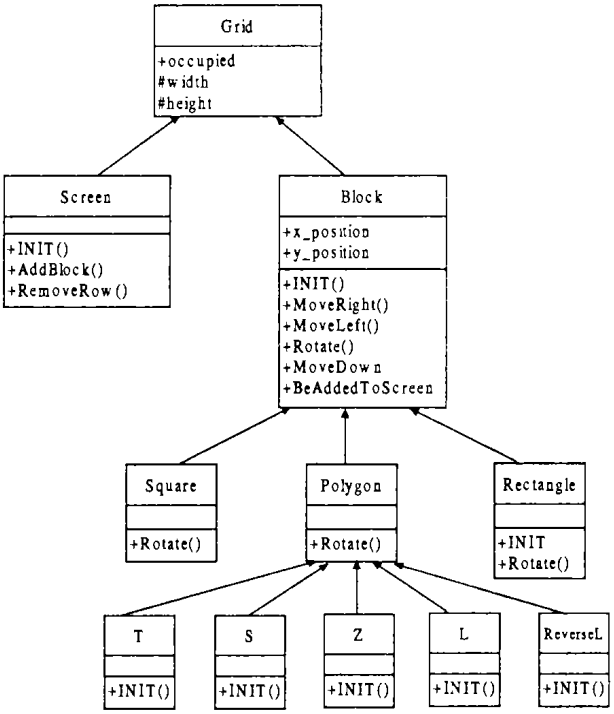

图5

5.1 测试用例

测试用例通常指测试输入数据和期望的输出。测试用例是在一定的测试准则指导下而产生的。在该实例中利用三个测试准则[1] MPLESC(最大多态无环有效序列覆盖)、APLESC(全多态无环有效序列覆盖)和 APTC(全多态测试覆盖)推导出测试用例[1]。

APTC 准则:对于被测类 *C* 的上下文中的每个类定义,测试用例集必须包含所有可能的方法内测试用例。

APLESC 准则:对于被测类 *C* 的上下文中的每个类定义,测试用例集必须包含所有的方法内测试用例的无环有效测试序列。

MPLESC 准则:对于被测类 *C* 的上下文中的每个类定义,测试用例集必须包含所有的方法内测试用例的最大无环有效测试序列

测试用例如下所示(本文中为了说明方便,只列出了部分用例)

```
<INIT,MoveLeft,MoveRight>block=T,
<INIT,MoveLeft,MoveDown,MoveRight>block=Square,
<INIT,MoveLeft,MoveDown,MoveRight>block=S,
<INIT,MoveLeft,MoveRight>block=Rectangle,
<INIT,MoveLeft,MoveDown,MoveRight>block=Rectangle,
<INIT,MoveLeft,MoveDown,AddBlock,Rotate>block=Square,
```

并将该测试用例以文件(casefile.txt)的形式保存。

关于该例的 object-Z 规格说明,C++ 实现,相应的测试用例,请参见文[3]。

(下转第185页)

12 Sabater J,Sierra C. Reputation and social network analysis in multi-agent systems. AAMAS,2002. 475～482
13 Barber K S,Kim J. Belief Revision Process Based on Trust: Agents Evaluating Reputation of Information Sources. Trust in Cyber-societies LNAI2246,Spring-Verlag Berlin H,2001. 73～82
14 Esfandiari B,Chandrasehharan S. On how Agents Make Friends: Mechanisms for Trust Acquisition. In:Proc. of the 4th workshop on Component-Oriented Programming,2001
15 Sen S. believing others: Pros and cons. Artificial Intelligence, 2002:142:179～203
16 Yu B,Singh M P. Detecting Deception in Reputation management. In:Proc. of 2nd Intl. Joint Conf. on Autonomous Agents and Multi-Agent Systems,2003. 73～80
17 Yu B, Singh M P. Searching Social Networks. In: Proc. of 2nd Intl. Joint Conf. on Autonomous Agents and Multi-Agent Systems,2003. 65～72
18 Watts D J,Strogatz S H. Collective dynamics of small-world networks. Nature,1998,393:440～442

(上接第161页)

5.2 测试驱动程序的构造

利用测试驱动程序框架来构造 *Tetris* 类的驱动程序的步骤如下。

步骤一：TetrisTesterDriver：public TestDriver，表示TesrisTesterDriver 从 TestDriver 继承而来。

步骤二：覆盖 TesterDriver 中的 *RunTest*()。其中 *RunTest*()的设计按照前面所提到的算法来实现(为了说明的方便本文以伪代码的形式给出)。

```
RunTest(){
tetris. block = I;
while casefile not end
    {
      for(casepoint from testcase. first to testcase. end )
       { if( * casepoint="INIT")
                 { tetris. block->Init();
                   oracleblockinit()}
                      M
                   if( * casepoint="Rotate")
                 { tetris. Rotate();
                   oracle. rotate();}
       }
    }
}
```

说明：对于 *RunTest*()函数，对于不同的 CUT，*RunTest*()是不同的。其中的省略号部分表示的是测试用例中每个方法的调用，及其对应方法的测试预测程序(orcacle)的执行。

步骤三：注册该被测类驱动，在 *main*()函数中得到语句：

```
#ifdef TetrisTesterDriver
      TestDriver *r1=new Tetris
      r->Runtest()
#end if
```

步骤四：将被测类，被测类的驱动类和主控程序一起编译连接生成可执行的程序，从而运行测试用例。通过调用 CL 编译器完成。其中我们将编译连接的参数设置和连接参数的设置以 compileset. txt，linkset. tx 的形式存放。

```
cl@compileset. txt
link@linkset. txt
```

即可生成我们所需要的可执行的驱动程序，利用生成的可执行程序可以驱动测试用例运行，从而完成了 *Tetris* 类的测试。运行后，我们将会得到一个存放运行结果的文件 TestResult. txt。

运行结果(即 TestResult. txt 里面的内容)如下：

```
Test case1: INIT MoveLeft MoveRight block=T
    pass
Test case2: INIT MoveLeft MoveDown MoveRight block=Square
    pass
Test case3: INIT MoveLeft MoveDown MoveRight block=S
    pass
Test case4: INIT MoveLeft MoveRight block=Rectangle
    pass
Test case5: INIT MoveLeft MoveDown MoveRight block=Rectangle
    pass
Test case6: INIT MoveLeft MoveDown AddBlock Rotate block=
    Square
    pass
total test cases: 6
PassedTests: 6 passRatio: 100%
failedTests: 0 failedRatio: 0%
```

总结 本文提出了一种面向对象类测试驱动程序的构造框架，介绍了该框架的设计。将类驱动设计成了一个独立的类，使得驱动程序有了很好的复用性；利用条件编译实现被测类的驱动类注册，使得在一个主控程序下，能够同时测试多个类，并且条件编译可以比较灵活地控制驱动程序的执行。最后通过实例说明了这种框架的可行性。利用该驱动程序构造框架，可以方便快速地生成类测试驱动程序。

本文所提出的框架还存在有待于进一步研究和改进的地方，例如对于驱动中的测试结果的预测。因为对于测试结果预测的自动化，将会进一步简化驱动程序的设计。因此希望在以后的工作中在此方面取得进展。

从目前的研究现状来看，对于类测试驱动程序的构造，现在还没有一个完全的解决方案。但对于类测试驱动的研究将对提高类测试的效率，实现类测试的自动化，以及类的回归测试都有着十分重要的意义。

参考文献

1 Doong R K,Frankl P G. The ASTOOT Approach to Testing Object-oriented Programs [J]. ACM Transactions on Software Engineering and Methodology,1994,3(2):101～130
2 Ihssan A,Carver D. A Testing Assistant for Object-oriented Programs [J]. In:1998 IEEE Aerospace Conf.
3 刘玲. 基于面向对象形式规格说明的测试用例生成技术[D]. 上海大学,2004
4 McGrgor J D,Sykes D A 著. 面向对象的软件测试[M]. 北京：机械工业出版社,2002
5 Luo Gang,Probert R L,Dral H. Approach to construction software unit testing tools [J]. Software Engineering Journal,1995. 11
6 Liu Ling,Miao Huaikou,Zhan Xuede. A Framework for Specification-Based Class Testing. In:Eighth IEEE intl. conf. on engineering of complex computer systems,Dec. 2002. 153～162
7 Doong R K,Frankl P G. The ASTOOT approach to testing object-oriented programs [J]. ACM Transactions on Software Engineering and Methodology,1994,3(2):101～130
8 张雪萍，张慧档，庄雷. 面向对象软件的类测试技术[J]. 微机发展，2002,5:74～77
9 郑春一，宋雨，孙文婧. 面向对象类测试方法分析[J]. 微机发展，2003,1:57～59
10 陈站华. 软件单元测试[J]. 软件与测试,2003,5:50～51