Derived Classes Unit Testing

Inheritance is one of the bases of object-oriented programming. Inheritance is a hierarchical relationship that allows us to exploit the commonality between various classes. Since classes represent concepts, and concepts do not exist in isolation, it can be implied that there is a logical relationship between various classes[Str91]. Inheritance is one of many possible glues that logically bind two classes. Inheritance provides a simple, flexible, and efficient mechanism for defining a new class by adding or modifying the facilities provided by an existing class. When two classes are related by inheritance, a class is derived from another class, which is then called a base class. The derived class inherits the properties, including data members and member functions, of its base class[ES90].

This chapter deals with the unit testing of inherited, or derived, classes. There are two extreme options for testing a derived class. One is to test a derived class as a flattened[BB91] class. This, however, would require retesting of all attributes of a base class. The other extreme possibility is to do the obvious. Apparently, if a base class has been adequately tested, then its attributes in a derived class need not be retested. This intuition has been rebutted.[PK90].

We contest that not all attributes of a base class need retesting in a derived class. Our solution identifies only the attributes of a base class

\[1\] According to Meyer[Mey90], the “flattened form of a class is the text of the class with the same features as the original, but with no inheritance clause. Features inherited directly or indirectly are put in the flattened form at the same level as the features declared locally.”
that mandate retesting in a derived class scope. We assume that the base class has been adequately tested and the test history of the base class is available.

Here we deviate from our regular chapter format, i.e., objective, approach, activities, and so on. Although we are still addressing the unit testing of a class in an object-oriented or in an object-based language, the testing of derived classes is a significant topic and warrants serious consideration. In this chapter, we describe our approach toward testing derived classes. All recommendations and practices of chapter 6 are still applicable.

7.1 Test Procedure

In chapter 6, we examined a procedure for testing a stand-alone class. Generating the test cases included slicing the class and then generating test cases for each of the slices. In this chapter, we use the same test procedure to test a derived class $D_{\text{derived}}$. We assume that the base class $B_{\text{base}}$ of class $D_{\text{derived}}$ has already been tested as a stand-alone class using the approach described in chapter 6. We introduce a notation by which an attribute of a class is subscripted with the base or derived keywords to identify the context in which it is being used.

We apply our SampleStatistic class test procedure to SampleHistogram class, a derived class of SampleStatistic class, in libg++[Lea92]. The interface of the class SampleHistogram is given in figure 7.1.

7.1.1 Draw Derived Enchanced Call Graph ($ECG_{\text{derived}}$)

The first step in testing a derived class is to explicitly determine the inter- and intra-relationships between its local and inherited attributes. An ECG gives us a visual feeling of these relationships. We use the notation given in figure 7.2 to distinguish between the attributes of base and derived classes. An ECG of SampleHistogram class is shown in figure 7.3.

7.1.2 Draw $MaDUM_{\text{derived}}$

The testing of a derived class starts with extending the MaDUM of its base class. Hence, the next step in testing a derived class is to draw its minimal data members usage matrix ($MaDUM$).

- Take the MaDUM of the base class $B_{\text{base}}$. 