CHAPTER IV

ADEQUACY CRITERIA

4.1 Overview

In this research, a set of adequacy criteria for testing polymorphic interactions is defined. issues related to inheritance, polymorphism, and dynamic binding are the major concern of these criteria. Each criterion is designed to enforce a particular level of test coverage. They range from covering just base class, completely ignoring polymorphism, to covering all combinations of subclasses.

Two important inputs for this process are interactions, in this case UML sequence diagrams, and class inheritance hierarchies, possibly UML class diagrams. Interactions are required to identify polymorphic entities, to identify which objects in a particular interaction must be considered while evaluating the adequacy criteria. Class inheritance hierarchies are important for identifying which classes are subclasses of a class of a particular polymorphic entity, hence must be covered by test case.

Five polymorphic adequacy criteria are defined in this research. A particular criterion may be superior to some, as a set of test cases which satisfies the criterion always satisfies those criteria. The subsumption hierarchy of the adequacy criteria is shown in figure 4.1.

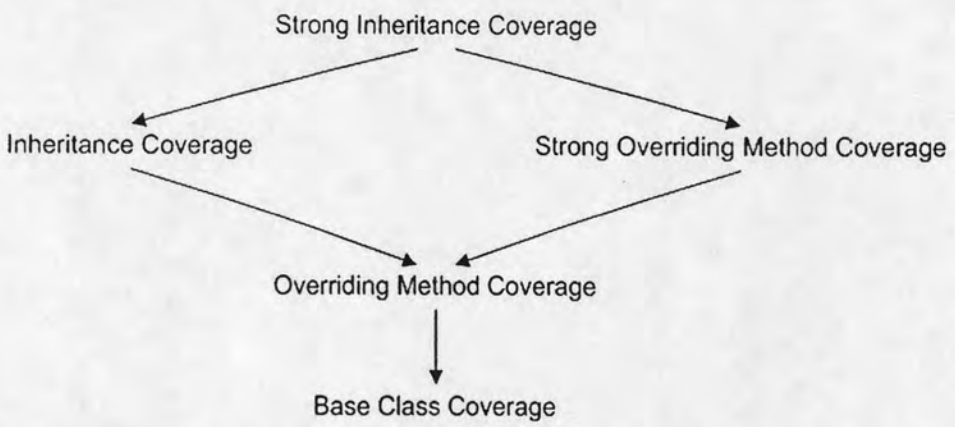

Figure 4.1 Subsumption hierarchy of the criteria

4.2 The Criteria

The adequacy criteria presented here describe how polymorphic interactions must be tested. In a polymorphic interaction, a polymorphic entity can actually be substituted by an object of one of the subclasses of the class of the entity, including the class itself, during the execution of the implementation of the interaction. To adequately test the interaction must also test the interaction with those subclasses. The adequacy criteria help dictating which superclasses/subclasses must be covered in test. They describe how to find out which classes must be included for test for a particular polymorphic interaction.

Presented in the subsequent subsections are the five adequacy criteria. For convenience, an interaction in UML sequence diagram in figure 4.2 is used for demonstration of how to apply each adequacy criterion. There are 4 roles in the interaction. The actor is the activator of the interaction. During test, it is replaced by a test driver. The other three roles are objects that collaborate in the interaction. There are represented by objects of classes: A, B, and C. Their structures are illustrated in UML class diagram in figure 4.3.

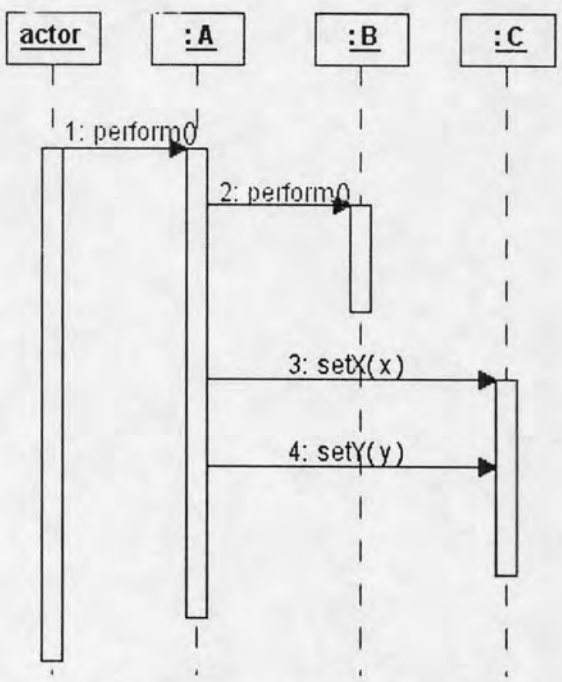

Figure 4.2 Example of a UML sequence diagram

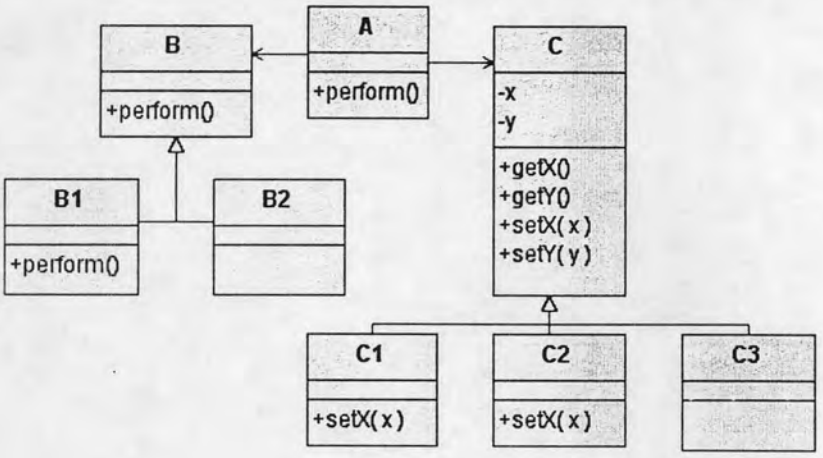

Figure 4.3 Example of a UML class diagram

From the example, there are two polymorphic entities: on objects of class B and C. Although there is a specific way for recognizing polymorphic entities presented in the next chapter, here they are recognized by checking if the inheritance hierarchies of the classes presented in the interaction have subclasses.

4.2.1 Base Class Coverage

Base class coverage is a criterion which totally ignores polymorphic behavior in interactions. This criterion requires no superclass-subclass substitution. Classes that must be covered under test are only those defined in the interaction.

From the example, Base class coverage requires just one test case to cover all classes presented in the UML sequence diagram. In this case, a test scenario which includes class A, B, and C must be executed to satisfy the criterion. Table 4.1 shows classes used for each object in the interaction for the test case.

Table 4.1 Test scenario for Base class coverage from the example

No.	Entity A	Entity B	Entity C
1	A	B	C

4.2.2 Overriding Method Coverage

Overriding method coverage is designed based on the assumption that overriding methods in subclasses are new parts of code introduced to the program under test; hence, they must be tested regardless of their parent overridden method. This means that an interaction, which has a message which represents a call to a method which is overridden, must also be tested with all of the overriding methods, as well as the overridden method. In other words, it is required that each class in an interaction must be substituted for test by its subclasses which override the method involved in the interaction. This criterion subsumes Base class coverage.

The criterion is very intuitive and common in traditional software testing. If some traditional test coverage criteria, for example statement coverage, are applied, they may result in a similar fashion as this criterion, since they also enforce the overriding methods, as uncovered code, to be tested. However, there is a small, but significant, difference in the context of test. Although coverage criteria, such as statement coverage, may enforce also the coverage of overriding methods, they do not state under which context the methods must be tested. The adequacy criteria presented in this research are more specific to context. Since a particular interaction is taken into account when adequacy criteria are considered, the criteria enforce testing of subclasses under the interaction. In other words, the subclasses must be tested in the place of its superclass in the interaction.

From the example in figure 4.2 and 4.3, method "perform" on class B, and method "setX" and "setY" on class C are used on the interaction. In addition to base classes, class B1, C1, and C2 must be included in test, since they all override at least one of the methods used in the interaction under test. The test set then must cover class A, B, C, B1, C, C1, and C2. The criterion does not enforce which classes from an inheritance hierarchy must be tested against which classes from other hierarchies; as a consequence, test scenarios may be arranged in several different ways to satisfy the criterion. Table 4.2 shows an example of a set of test scenarios which satisfies Overriding method coverage. Note that it is possible that another different set of test scenarios may also satisfy the criterion as well.

Table 4.2 Test scenario for Overriding method coverage from the example

No.	Entity A	Entity B	Entity C
1	A	B	C
2	A	B1	C1
3	A	B	C2

4.2.3 Inheritance Coverage

Inheritance coverage requires that a polymorphic entity in an interaction must be substituted with all of its subclasses for testing. The criterion is designed based on assumption that an inherited method, although already tested in the context of the superclass, must also be tested in the context of the subclasses as well. Perry and Kaiser stated in their work [1] that a component adequately tested under a context is not adequately tested by itself, which means a method/operation adequately tested in the context of the superclass is not adequately tested by itself, hence not adequately tested in the context of the subclasses.

The criterion, therefore, takes into account both inherited methods and overriding methods. It requires all subclasses to be tested in the context of its superclass in the interaction regardless of whether they override the method used in the interaction. Base class coverage and Overriding method coverage are subsumed by this criterion.

From the example, class A, B, B1, B2, C, C1, C2, and C3 must be included in test. Similar to Overriding method coverage, the criterion does not require particular arrangement of test scenarios. As long as the required classes are covered by the test set, the criterion is satisfied. Table 4.3 shows an example of a set of test scenarios which satisfies Inheritance coverage.

Table 4.3 Test scenario for Inheritance coverage from the example

No.	Entity A	Entity B	Entity C
1	A	B	C
2	A	B1	C1
3	A	B2	C2
4	A	B	C3

4.2.4 Strong Overriding Method Coverage

Previous three criteria do not take into account the possibility of failures occurred from more than one polymorphic entity. They entirely rely on single-fault assumption, which assumes that a failure only happens from a single fault in the program under test. On the contrary, multiple-fault assumption assumes that a failure may be a result from more than one fault in the program under test. Therefore, test techniques based on this assumption must take into account detection of failures from multiple faults.

Since it is possible that there is more than one polymorphic entity in an interaction, it is possible that a fault may be a result from a particular combination of subclasses on more than one polymorphic entity. Testing combinations of subclasses is then necessary. Strong overriding method coverage is designed based on Overriding method coverage. Similar to Overriding method coverage, it requires that subclasses with overriding methods must be tested. It additionally requires that all combinations of those subclasses must be tested. This criterion subsumes Overriding method coverage.

Table 4.4 shows the set of test scenarios which satisfies the criterion for the example in figure 4.2 and 4.3.

Table 4.4 Test scenario for Strong overriding method coverage from the example

No.	Entity A	Entity B	Entity C
1	A	B	C
2	A	B	C1
3	A	B	C2
4	A	B1	C
5	A	B1	C1
6	A	B1	C2

4.2.5 Strong Inheritance Coverage

As the strongest criterion among the adequacy criteria presented in this research, Strong inheritance coverage requires that all subclasses, regardless of whether they override methods used in the interaction, must be tested in the place of their corresponding polymorphic entity in the interaction in all combinations. It is the strong version of Inheritance coverage. The criterion is the strongest one; hence, the other criteria are subsumed by the criterion.

For the example in figure 4.2 and 4.3, the set of test scenarios which satisfies the criterion is shown in table 4.5. For both strong adequacy criteria, there would be only one set of test scenarios which satisfies each criterion.

Table 4.5 Test scenario for Strong inheritance coverage from the example

No.	Entity A	Entity B	Entity C
1	A	B	C
2	A	B	C1
3	A	B	C2
4	A	B	C3
5	A	B1	C
6	A	B1	C1
7	A	B1	C2
8	A	B1	C3
9	A	B2	C
10	A	B2	C1
11	A	B2	C2
12	A	B2	C3

4.3 Discussion

Base class coverage is the weakest criterion among the criteria presented here. It should only be suitable for a case where polymorphism does not actually take place in execution of the interaction even though it looks like it possibly could. This is the case of subclass inheritance. Opposed to subtype inheritance which allows an object of a subclass to substitute and play a role of its superclass, subclass inheritance does not have this substitutable property. Its major purpose is to apply inheritance for reusability of attributes and methods of superclass; therefore, superclass-subclass substitution does not occur.

Another situation where Base class coverage can be applied is when subclasses of the class of the polymorphic entity in the interaction are pure extension to the superclass. In other words, the subclasses do not override any feature on the superclass, and their new features do not affect the features on the superclass. Offutt et

al. showed that this situation is free from ITU (Inconsistent Type Use) fault [15], since the extension methods, which can cause failures, are not accessible in the context of the superclass, and the absence of multiple context scenario also keeps them from being accessed in the context of the subclass. In this situation, it may be more cost-effective to test only the superclass in the interaction. However, this requires human judgement. Unless there is a systematic way to perform semantic analysis on the structures of the superclass and subclasses, this decision must always rely on human.

When a method is overridden, intuitively the overriding version of the method should be tested in the context of the overridden method to ensure that the overriding method is correct. According to this belief, Overriding method coverage criterion seems to be appropriate for testing polymorphic interaction. However, showing in the studies, the absence of overriding method does not guarantee total correctness. While a subclass inherits a method from its superclass, correctness of the method is not inherited to the context of the subclass. There may be side effects from situations outside the scenario of the interaction under test, for example usage of multiple context (casting) and difference in initialization (or constructors) of superclass and subclass. Testing is usually required for the entire inheritance hierarchies to ensure complete correctness, whether overriding or not; therefore, Inheritance coverage criteria is considerably better in terms of reliability and thoroughness. However, Overriding method coverage criterion can provide a reasonable trade off in the situation where Base class coverage criterion is inappropriate, and Inheritance coverage criterion requires an unacceptable amount of effort.

Enhancing the regular criteria by exercising all combinations of substitutable classes, Strong overriding method coverage and Strong inheritance coverage criteria are designed to cope with an interaction, which includes more than one class with inheritance hierarchy. While they are very effective to reveal faults which occur in the situation of a particular combination of class substitution, they may require a tremendous test set. In the situation where, in an interaction, two objects with inheritance hierarchies do not interact with each other directly, there is a very little chance that faults could occur from the combinations of class substitution. In such a case, either one of the

regular criteria is possibly more cost-effective than its strong version criterion, as the excessive test cases are eliminated while comparable test effectiveness is achieved.

There is one issue regarding applying the adequacy criteria. In a certain programming language, it is possible that there are one or more abstract types in an inheritance hierarchy. An abstract type is a class-like construct which contains one or more abstract methods/operations. Examples are interfaces (Java), abstract classes (Java), and classes containing one or more virtual methods which have empty body. These types are not instantiable; consequently, it is impossible to have an object of any of these types. Care must be taken when applying the adequacy criteria. Abstract types cannot be tested directly. If an abstract type is required to be tested by a particular criterion, one of its subtypes/subclasses must be tested in its place instead. This rule is applied to all adequacy criteria presented in this research.

One interesting topic regarding adequacy criteria is the minimum number of test cases required by each adequacy criterion. It is obvious that Base class coverage always requires only one test case. The minimum number of test cases for Overriding method coverage and Inheritance coverage is the maximum value of the numbers of classes to be tested for each role. For example, the class diagram and the sequence diagram in figure 4.2 and 4.3 respectively requires 4 test cases to satisfy Inheritance coverage criterion, as there are 1, 3, and 4 classes to be tested for object role A, B, and C respectively. The minimum number of test cases for Strong overriding method coverage and Strong inheritance coverage is equal to the multiplication of the numbers of classes to be tested from every object role in the interaction. For example, the number of test cases to satisfy Strong inheritance coverage is 12 test cases, since there are 1, 3, and classes to be tested for object role A, B, and C respectively. The minimum numbers of test cases of the adequacy criteria are summarized in table 4.6.

Table 4.6 Minimum number of test cases for the adequacy criteria

Adequacy Criteria	Minimum No. of Test Cases
Base Class Coverage	1
Overriding Method Coverage & Inheritance Coverage	Maximum value of the numbers of classes to be tested for each role
Strong Overriding Method Coverage & Strong Inheritance Coverage	Multiplication product of the numbers of classes to be tested from every role in the interaction