Chapter 15

Test Adequacy Assessment Using Control Flow and Data Flow

This chapter introduces methods for the assessment of test adequacy and test enhancement. Measurements of test adequacy using criteria based on control flow and data flow are explained. These code-based coverage criteria allow a tester to determine how much of the code has been tested and what remains untested.

15.1. Test Adequacy: Basics

15.1.1. What is test adequacy ?

Consider a program P written to meet a set R of functional requirements. We notate such a P and R as (P, R). Let R contain n requirements labeled $R_1, R_2, \ldots, R_n$. Suppose now that a set T containing k tests has been constructed to test P to determine whether or not it meets all the requirements in R. Also, P has been executed against each test in T and has produced correct behavior. We now ask: *Is T good enough ?* This question can be stated differently as: *Has P been tested thoroughly ?*, or as: *Is T adequate ?* Regardless of how the question is stated, it assumes importance when one wants to test P thoroughly in the hope that all errors have been discovered and removed when testing is declared complete and the program P declared usable by the intended users.

In the context of software testing, the terms "thorough," "good enough," and "adequate," used in the questions above, have the same meaning. We prefer the term "adequate" and the question *Is T adequate ?*. Adequacy is measured for a given test set designed to test P to determine whether or not P meets its requirements. This measurement is done against a given criterion C. A test set is considered adequate with respect to criterion C when it *satisfies* C. The determination of whether or not a test set T for program P satisfies criterion C depends on the criterion itself and is explained later in this chapter.

In this chapter we focus only on *functional* requirements, testing techniques to validate non-

functional requirements are dealt with elsewhere.

EXAMPLE 15.1. Consider the problem of writing a program named `sumProduct` that meets the following requirements:

R_1 : Input two integers, say `x` and `y`, from the standard input device.

$R_{2.1}$: Find and print to the standard output device the sum of `x` and `y` if $x < y$.

$R_{2.2}$: Find and print to the standard output device the product of `x` and `y` if $x \geq y$.

Suppose now that the test adequacy criterion C is specified as follows:

C: *A test T for program (P, R) is considered adequate if for each requirement r in R there is at least one test case in T that tests the correctness of P with respect to r.*

It is obvious that $T = \{t :< x = 2, y = 3 >\}$ is inadequate with respect to C for program `sumProduct`. The lone test case t in T tests R_1 and $R_{2.1}$, but not $R_{2.2}$. ■

15.1.2. Measurement of test adequacy

Adequacy of a test set is measured against a finite set of elements. Depending on the adequacy criterion of interest, these elements are derived from the requirements or from the program under test. For each adequacy criterion C, we derive a finite set known as the *coverage domain* and denoted as C_e.

A criterion C is a *white-box* test adequacy criterion if the corresponding coverage domain C_e depends solely on program P under test. A criterion C is a *black-box* test adequacy criterion if the corresponding coverage domain C_e depends solely on requirements R for the program P under test. All other test adequacy criteria are of a mixed nature and not considered in this chapter. This chapter introduces several white-box test adequacy criteria that are based on the flow of control and the flow of data within the program under test.

Suppose that it is desired to measure the adequacy of T. Given that C_e has $n \geq 0$ elements, we say that T *covers* C_e if for each element e' in C_e there is at least one test case in T that tests e'. T is considered adequate with respect to C if it covers all elements in the coverage domain. T is considered inadequate with respect to C if it covers k elements of C_e where $k < n$. The fraction k/n is a measure of the extent to which T is adequate with respect to C. This fraction is also known as the *coverage* of T with respect to C, P, and R.

The determination of when an element e is considered *tested* by T depends on e and P and is explained below through examples.

EXAMPLE 15.2. Consider the program P, test T, and adequacy criterion C of Example 15.1. In this case the finite set of elements C_e is the set $\{R_1, R_{2.1}, R_{2.2}\}$. T covers R_1 and $R_{2.1}$ but not $R_{2.2}$. Hence T is not adequate with respect to C. The coverage of T with respect to C, P, and R is 0.66. Element $R_{2.2}$ is not tested by T whereas the other elements of C_e are tested. ■

EXAMPLE 15.3. Next let us consider a different test adequacy criterion which is referred to as the *path coverage* criterion.

C: *A test T for program (P, R) is considered adequate if each path in P is traversed at least once.*

Given the requirements in Example 15.1 let us assume that P has exactly two paths, one corresponding to condition $x < y$ and the other to $x \geq y$. Let us refer to these two paths as p_1 and p_2, respectively. For the given adequacy criterion C we obtain the coverage domain C_e to be the set $\{p_1, p_2\}$.

To measure the adequacy of T of Example 15.1 against C, we execute P against each test case in T. As T contains only one test for which $x < y$, only the path p_1 is executed. Thus the coverage of T with respect to C, P, and R is 0.5 and hence T is not adequate with respect to C. We also say that p_2 is not tested. ■

In Example 15.3 we assumed that P contains exactly two paths. This assumption is based on a knowledge of the requirements. However, when the coverage domain must contain elements from the code, these elements must be derived by program analysis and not by an examination of its requirements. Errors in the program and incomplete or incorrect requirements might cause the program, and hence the coverage domain, to be different from what one might expect.

EXAMPLE 15.4. Consider the following program written to meet the requirements specified in Example 15.1; the program is obviously incorrect.

Program P15.1

```
1   begin
2     int x, y;
3     input (x, y);
4     sum=x+y;
5     output (sum);
6   end
```

The above program has exactly one path which we denote as p_1. This path traverses all statements. Thus, to evaluate any test with respect to criterion C of Example 15.3, we obtain the coverage domain C_e to be $\{p_1\}$. It is easy to see that C_e is covered when P is executed against the sole test in T of Example 15.1. Thus T is adequate with respect to P even though the program is incorrect.

Program P15.1 has an error that is often referred to as a "missing path" or a"missing condition" error . A correct program that meets the requirements of Example 15.1 follows.

Program P15.2

```
1   begin
2     int x, y;
3     input (x, y);
4     if(x<y)
5     then
6       output(x+y);
```

```
7      else
8        output(x*y);
9    end
```

This program has two paths, one of which is traversed when $x < y$ and the other when $x \geq y$. Denoting these two paths by p_1 and p_2 we obtain the coverage domain given in Example 15.3. As mentioned earlier, test T of Example 15.1 is not adequate with respect to the path coverage criterion.

The above example illustrates that an adequate test set might not reveal even the most obvious error in a program. This does not diminish in any way the need for the measurement of test adequacy. The next section explains the use of adequacy measurement as a tool for test enhancement.

15.1.3. Test enhancement using measurements of adequacy

While a test set adequate with respect to some criterion does not guarantee an error-free program, an inadequate test set is a cause for worry. Inadequacy with respect to any criterion often implies deficiency. Identification of this deficiency helps in the enhancement of the inadequate test set. Enhancement in turn is also likely to test the program in ways it has not been tested before such as testing untested portion, or testing the features in a sequence different from the one used previously. Testing the program differently than before raises the possibility of discovering any uncovered errors.

EXAMPLE 15.5. Let us reexamine test T for P15.2 in Example 15.4. To make T adequate with respect to the path coverage criterion, we need to add a test that covers p_2. One test that does so is $\{< x = 3, y = 1 >\}$. Adding this test to T and denoting the expanded test set by T', we get:

$T' = \{< x = 3, y = 4 >, < x = 3, y = 1 >\}$.

When P15.2 is excuted against the two tests in T', both paths p_1 and p_2 are traversed. Thus T' is adequate with respect to the path coverage criterion.

Given a test set T for program P, test enhancement is a process that depends on the test process employed in the organization. For each new test added to T, P needs to be executed to determine its behavior. An erroneous behavior implies the existence of an error in P and will likely lead to debugging of P and the eventual removal of the error. However, there are several procedures by which the enhancement could be carried out. One such procedure follows.

`Procedure for Test Enhancement Using Measurements of Test Adequacy.`

Step 1 Measure the adequacy of T with respect to the given criterion C. If T is adequate then go to Step 3, otherwise execute the next step. Note that during adequacy measurement we will be able to determine the uncovered elements of C_e.

Step 2 For each uncovered element $e \in C_e$, do the following until e is covered or is determined to be infeasible.

2.1 Construct a test t that covers e or will likely cover e.

2.2 Execute P against t.

2.2.1 If P behaves incorrectly then we have discovered the existence of an error in P. In this case t is added to T, the error is removed from P and this procedure repeated from the beginning.

2.2.2 If P behaves correctly and e is covered then t is added to T, otherwise it is the tester's option whether to ignore t or to add it to T.

Step 3 Test enhancement is complete.

`End of Procedure`

Figure 15.1 shows a sample test construction-enhancement cycle. The cycle begins with the construction of a non-empty set T of test cases. These tests cases are constructed from the requirements of program P under test. P is then executed against all test cases. P is corrected if it does not behave as per the requirements on any test case. The adequacy of T is measured with respect to a suitably selected adequacy criterion C after P is found to behave satisfactorily on all elements of T. This construction-enhancement cycle is considered complete if T is found adequate with respect to C. If not, then additional test cases are constructed in an attempt to remove the deficiency. The construction of these additional test cases once again makes use of the requirements that P must meet.

EXAMPLE 15.6. Consider the following program intended to compute x^y given integers `x` and `y`. For `y` < 0 the program skips the computation and outputs a suitable error message.

Program P15.3

```
1   begin
2     int x, y;
3     int product, count;
4     input (x, y);
5     if(y≥0) {
6       product=1; count=y;
7       while(count>0) {
8         product=product*x;
9         count=count-1;
10      }
11    output(product);
12    }
13    else
14      output ( “Input does not match its specification.”);
15  end
```

Next, consider the following test adequacy criterion.