

Test Adequacy Measurement & Enhancement

W. Eric Wong
Department of Computer Science
The University of Texas at Dallas
ewong@utdallas.edu
<http://www.utdallas.edu/~ewong>

Speaker Biographical Sketch

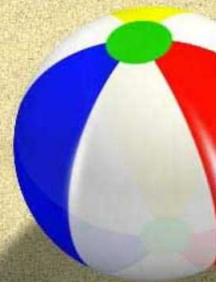
- Professor & Director of International Outreach
Department of Computer Science
University of Texas at Dallas
- Guest Researcher
Computer Security Division
National Institute of Standards and Technology (NIST)
- Vice President, IEEE Reliability Society
- Secretary, ACM SIGAPP (Special Interest Group on Applied Computing)
- Principal Investigator, NSF TUES (Transforming Undergraduate Education in Science, Technology, Engineering and Mathematics) Project
– *Incorporating Software Testing into Multiple Computer Science and Software Engineering Undergraduate Courses*
- Founder & Steering Committee co-Chair for the SERE conference
(*IEEE International Conference on Software Security and Reliability*)
(<http://paris.utdallas.edu/sere13>)



Outline

- Test Adequacy Measurement
- Test Set Enhancement

What Is Test Adequacy?



What Is Adequacy?

- Consider a program P written to meet a set of functional requirements R . We notate such a P and R as (P, R) . Let R contain n requirements labeled R_1, R_2, \dots, R_n .
- Suppose now that a set T containing k test cases has been constructed to test P to determine whether or not it meets all the requirements in R
 - Assume also P has been executed against each test case 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?*

Measurement of Adequacy

- In the context of software testing, the terms “*thorough*,” “*good enough*,” and “*adequate*,” used in the questions above, *have the same meaning*.
- **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 a criterion C when it satisfies C .
 - The determination of whether or not a test set T for program P satisfies a criterion C depends on the criterion itself and is explained later.

Black-box and White-box Criteria

- 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*.

Coverage

- We want 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 $C, P, and R$* .

Example I (1)

- Program `sumProduct` must meet 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 $x \geq y$

Example I (2)

- Suppose now that the **test adequacy criterion** C is specified as

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 .

- In this case the coverage domain $C_e = \{R_1, R_{2,1}, R_{2,2}\}$
- $T = \{t: \langle x = 2, y = 3 \rangle\}$ (which has $x < y$) covers R_1 and $R_{2,1}$ but not $R_{2,2}$
 - T is **not adequate** with respect to C
 - The coverage of T with respect to $C, P,$ and R is 0.66

Example II (1)

- Consider the following criterion

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

- Assume that P has **exactly two paths**, one corresponding to condition $x < y$ and the other to $x \geq y$.
 - We refer to these as p_1 and p_2 , respectively.
 - For the given adequacy criterion C we have the coverage domain $C_c = \{p_1, p_2\}$.
- We assume that P has exactly two paths. This assumption is based on the knowledge of the requirements. However, **when the coverage domain contains elements from the code, such elements should be derived from the program directly and not by an examination of its requirements.**

Example II (2)

- To measure the adequacy of T of sumProduct against C , we execute P against each test case in T .
- As $T = \{ \langle x = 2, y = 3 \rangle \}$ contains only one test for which $x < y$, only the path p_1 is executed.
 - The coverage of T with respect to C , P , and R is 0.5.
 - T is not adequate with respect to C .
 - We can also say that p_1 is tested and p_2 is not tested.

Example III

- The following program is incorrect as per the requirements of sumProduct
 - Using the path-based coverage criterion C , we have the coverage domain $C_c = \{p_1\}$
 - This path traverses all the statements
 - $T = \{ \langle x = 2, y = 3 \rangle \}$ is *adequate* with respect to C but does not reveal the bug

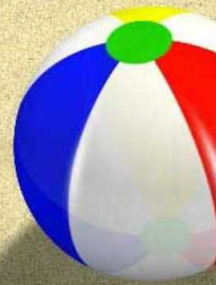
sumProduct-1

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

Lessons Learned

- An adequate test set might not reveal even the most obvious bug in a program.
- However, this does not diminish in any way the need for the measurement of test adequacy *as increasing coverage might reveal a bug!*

Test Enhancement



Test Enhancement

- While a test set adequate with respect to some criterion does not guarantee a bug-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 is likely to test the program in ways it has not been tested before such as testing an 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 hidden bugs.

Example IV (1)

- The following program is correct as per the requirements of sumProduct
 - It has two paths denoted by p_1 and p_2
 - $T = \{ \langle x = 2, y = 3 \rangle \}$ is *inadequate* with respect to the path-based coverage criterion C

sumProduct-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
```

Example IV (2)

- For sumProduct-2, to make T adequate with respect to the path coverage criterion we need to add a test case that covers p_2
 - One test case that does so is $\{ \langle x = 3, y = 1 \rangle \}$
 - Adding this test case to T and denoting the expanded test set by T' we have $T' = \{ \langle x = 3, y = 4 \rangle, \langle x = 3, y = 1 \rangle \}$
- Executing sumProduct-2 against the two test cases in T' will have both p_1 and p_2 traversed
 - T' is adequate with respect to the path coverage criterion

Example V (1)

- Consider a 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.

```
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
```

Example V (2)

- Suppose that a test set T is considered adequate if it tests the program on the previous slide for *at least one zero and one non-zero value of each of the two inputs x and y* .
- The *coverage domain* for C can be determined without any inspection of the program.
 - $C_e = \{x = 0, y = 0, x \neq 0, y \neq 0\}$
 - We can derive an adequate test set for the program by an examination of C_e
 - $T = \{t_1: \langle x = 0, y = 1 \rangle, t_2: \langle x = 1, y = 0 \rangle\}$ is adequate with respect to C

Example VI (Path Coverage) (1)

- Criterion C of the previous example is a **black-box coverage criterion** as it does not require an examination of the program under test for the measurement of adequacy.
- Let us now consider the *path coverage criterion*.
- An examination of the exponentiation program reveals that it has an indeterminate number of paths due to the *while* loop.
 - The number of paths depends on the value of y and hence that of count.

Example VI (Path Coverage) (2)

- Given that y is any non-negative integer, the number of paths can be arbitrarily large. This simple analysis of paths in the exponentiation program reveals that *for the path coverage criterion we cannot determine the coverage domain*.
- The usual approach in such cases is to simplify C and reformulate it as follows:

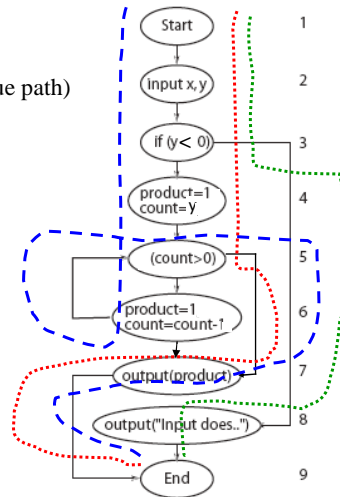
A test T is considered adequate if it tests all paths. In case the program contains a loop, then it is adequate to traverse the loop body zero times and once. (boundary-interior)

Example VI (Path Coverage) (3)

- The coverage domain of the modified path coverage criterion is $\{p_1, p_2, p_3\}$

- p_1 : $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 7 \rightarrow 9$ (red path)
- p_2 : $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 5 \rightarrow 7 \rightarrow 9$ (blue path)
- p_3 : $1 \rightarrow 2 \rightarrow 3 \rightarrow 8 \rightarrow 9$ (green path)

- Let $T = \{t_1: \langle x = 0, y = 1 \rangle,$
 $t_2: \langle x = 1, y = 0 \rangle,$
 $t_3: \langle x = 5, y = -1 \rangle\}$
 - As T does not contain any test with $y < 0$, p_3 (the green path) remains uncovered.
 - The coverage is $2/3 = 0.66$



Example VI (Path Coverage) (4)

- Any test case with $y < 0$ will cause p_3 to be traversed.
- Let t_3 be $\langle x = 5, y = -1 \rangle$
 - Test case t_3 covers path p_3 and P behaves correctly.
 - After adding t_3 to T , we have covered all feasible elements of the coverage domain.
 - The enhanced test set is $T = \{t_1: \langle x = 0, y = 1 \rangle, t_2: \langle x = 1, y = 0 \rangle, t_3: \langle x = 5, y = -1 \rangle\}$

Infeasibility and Test Adequacy

- An element of the coverage domain is *infeasible* if it cannot be covered by any test in the input domain of the program under test.
- **There does not exist an algorithm** that would analyze a given program and determine if a given element in the coverage domain is infeasible or not. Thus it is usually *the tester who determines* whether or not an element of the coverage domain is infeasible.

Demonstrating Feasibility

- **Feasibility can be demonstrated** by executing the program under test against a test case and showing that indeed the element under consideration is covered.
- **Infeasibility cannot be demonstrated** by program execution against a finite number of test cases.
 - In some cases *simple arguments* can be constructed to show that a given element is infeasible.
 - For more complex programs the problem of determining infeasibility could be *difficult*. Thus, an attempt to enhance a test set by executing a test *t* aimed at covering element *e* of program *P*, might fail.

Adequacy and Infeasibility

- In the presence of one or more infeasible elements in the coverage domain, a test is considered adequate when all feasible elements in the domain have been covered.
- While programmers might not be concerned with infeasible elements, testers attempting to obtain code coverage are.
- Prior to test enhancement, a tester usually does not know which elements of a coverage domain are infeasible.
 - Unfortunately, it is **only during an attempt to construct a test case to cover an element that one might realize the infeasibility of an element**

Fault Detection and Test Enhancement

- The purpose of test enhancement is to determine test cases that test the untested parts of a program.
 - Even the most carefully designed tests based exclusively on requirements can be enhanced.
- The *more complex* the set of requirements, the *more likely* it is that a test set designed using requirements is *inadequate* with respect to *even the simplest* of various test adequacy criteria.

Example VII (1)

- A program to meet the following requirements is to be developed.
- R_1 : Upon start the program offers the following three options to the user:
 - Compute x^y for integers x and $y \geq 0$.
 - Compute the factorial of integer $x \geq 0$.
 - Exit.
- $R_{1.1}$: If the “Compute x^y ” option is selected then the user is asked to supply the values of x and y , x^y is computed and displayed. The user may now select any of the three options once again.
- $R_{1.2}$: If the “Compute factorial x ” option is selected then the user is asked to supply the value of x and factorial of x is computed and displayed. The user may now select any of the three options once again.
- $R_{1.3}$: If the “Exit” option is selected the program displays a goodbye message and exits.

Example VII (2)

- Consider this program written to meet the above requirements.

```
1 begin
2   int x,y;
3   int product, request;
4   #define exp=1
5   #define fact=2
6   #define exit=3
7   get_request(request); // Get user request (one of three possibilities).
8   product=1; // Initialize product.
9   // Set up the loop to accept and execute requests.
10  while (request != exit)
11    // Process the "exponentiation" request.
12    if (request == 1)
13      input(x,y); count=y;
14      while (count > 0){
15        product=product * x; count=count-1;
16      }
17    // End of processing the "exponentiation" request.
18    // Process "factorial" request.
19    else if (request == 2)
20      input(x); count=x; product = 1;
21      while (count > 0){
22        product=product * count; count=count-1;
23      }
24    // End of processing the "factorial" request.
25    output(product); // Output the value of exponential or factorial and re-enter the loop.
26    input(request); // Get user request once again and jump to loop begin.
27  }
28 end
```

Example VII (3)

- Suppose now that the following test set has been developed to test whether or not our program meets its requirements.
- $T = \{ \langle \text{request} = 1, x = 2, y = 3 \rangle, \langle \text{request} = 2, x = 4 \rangle, \langle \text{request} = 3 \rangle \}$
- For the first two requests (*exponential followed by factorial*), the program correctly outputs 8 and 24. The program exits when executed against the third request. This program's behavior is correct and hence one might conclude that the program is correct.
- *Is this conclusion correct?*

Example VII (4)

- Let us now evaluate T against the path coverage criterion.
- *Construct the control flow graph of the example program and identify the paths not covered by T .*
- The coverage domain consists of all paths that traverse *each of the three loops zero and once in the same or different executions of the program.*

Example VII (5)

- Consider the path p that begins execution at line 1, reaches the outermost **while** at line 10, then the first **if** at line 12, followed by the statements that compute the factorial starting at line 20, and then the code to compute the exponential starting at line 13.
- p is traversed when the program is launched and *the first input request is to compute the factorial of a number*, followed by *a request to compute the exponential*. It is easy to verify that the sequence of requests in T (on slide 34) does not exercise p . Therefore, *T is inadequate with respect to the path coverage criterion.*

Example VII (6)

- To cover p we construct the following test:
- $T' = \{ \langle \text{request} = 2, x = 4 \rangle, \langle \text{request} = 1, x = 2, y = 3 \rangle, \langle \text{request} = 3 \rangle \}$
- When the values in T' are input to our example program in the sequence given, the program correctly outputs 24 as the *factorial* of 4 but incorrectly outputs 192 ($24 * 2 * 2 * 2$) as the value of 2^3 .
- This happens because T' traverses our “tricky” path *which makes the computation of the exponentiation begin without initializing product*. In fact the code at line 14 begins with the value of **product** set to 24.

Example VII (7)

- In our effort to increase the path coverage we constructed T' . *Execution of the test program on T' did cover a path that was not covered earlier and revealed a bug in the program.*
- This example has illustrated *a benefit of test enhancement based on code coverage.*

Multiple Executions (1)

- In the previous example we constructed two test sets T and T' . Notice that both T and T' *contain three tests one for each value of the variable request. Should T (or T') be considered a single test or a sequence of three tests?*
- $T = \{ \langle \text{request} = 2, x = 4 \rangle, \langle \text{request} = 1, x = 2, y = 3 \rangle, \langle \text{request} = 3 \rangle \}$

Multiple Executions (2)

- We assumed that all three tests, one for each value of **request**, are *input in a sequence during a single execution* of the test program. Hence we consider T as a test set containing one test case and write it as follows:

$$T = \left\{ \begin{array}{l} t_1 : \langle \langle \text{request} = 1, x = 2, y = 3 \rangle \rightarrow \\ \langle \text{request} = 2, x = 4 \rangle \rightarrow \langle \text{request} = 3 \rangle \rangle \end{array} \right\}$$

$$T' = \left\{ \begin{array}{l} t_1 : \langle \langle \text{request} = 1, x = 2, y = 3 \rangle \rightarrow \langle \text{request} = 2, x = 4 \rangle \rightarrow \\ \langle \text{request} = 3 \rangle \rangle \\ t_2 : \langle \langle \text{request} = 2, x = 4 \rangle \rightarrow \\ \langle \text{request} = 1, x = 2, y = 3 \rangle \rightarrow \langle \text{request} = 3 \rangle \rangle \end{array} \right\}$$

$$T'' = T \cup T'$$