EN.601.422 / EN.601.622

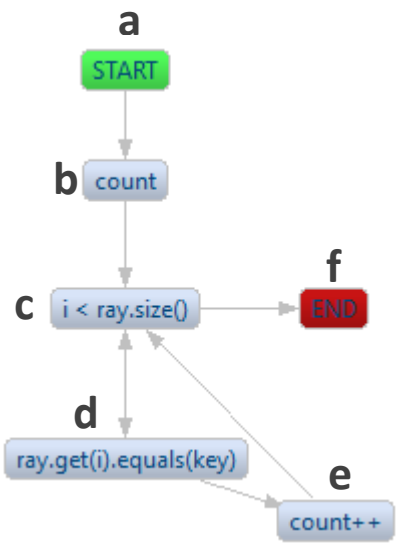
Software Testing & Debugging

Test Requirement, Coverage Criteria, and Coverage

- **Test Requirement (TR):** a specific element of software artifact that a test case must satisfy/cover.
  - Software artifact can be source code, user manual, API doc, design components (e.g., UML diagrams), GUI, input space, etc.
- **Coverage Criterion:** a rule or collection of rules to generate test requirements from a software artifact.
- **Coverage:** Given the set of TRs for a coverage criterion *C*, a test set *T* satisfies *C* (i.e., achieves coverage *C*) if and only if for any test requirement *tr* in *TR*, there is a test case *t* in *T* that covers/satisfies *tr*.

Example: Path Coverage Criterion

```
public static int countOf(ArrayList<Integer> ray, int key) {
   int count = 0;
   for (int i = 0; i < ray.size(); ++i) {
      if (ray.get(i).equals(key))  {
         count++;
      }
   }
   return count;
}
// Tests
List ray0 = new ArrayList<Integer>();
List ray1 = new ArrayList<Integer>();
ray1.add(2);
countOf(ray0, 2); // abcf
countOf(ray1, 1); // abcdcf
countOf(ray1, 2); // abcdecf
```



Paths to cover:

abcf
abcdcf
abcdecf

Test requirements

More Definitions

- **Minimal Test Set:** Given a set of test requirements *TR* and a test set *T*, *T* is minimal if removing any single test case from *T* will cause *T* to no longer satisfy all test requirements in *TR*.
- **Minimum Test Set:** A test set *T* that satisfies all test requirements in *TR* is minimum if there exists no smaller test set that can also satisfy all test requirements in *TR.*
- **Coverage Level:** Given test set *T* and test requirements set *TR*, assume that *T* covers/satisfies *x* number of test requirements in *TR*. The coverage level of *T* is: $\frac{x}{|TR|}$

Test Requirement Infeasibility

- Some test requirements may be infeasible, i.e., can never be satisfied.
- **Example:** unreachable code, thus not possible no achieve statement or branch coverage ➔ infeasible test requirement

```
if (x < 0) {
   // do some stuff
}
else if (x < -2) {
   // do some other stuff
}
```

Ways to Leverage Coverage Criteria

- 1: directly derive test cases to achieve coverage
  - Systematic
  - Not always easy/straightforward/possible
  - May not be always possible to automate
- 2: derive test cases and measure coverage level
  - Less systematic
  - Easier/more straightforward to perform
  - (The more) common practice in the industry

How to Utilize Coverage Criteria?

- 1: directly derive test cases to achieve coverage

Generator, i.e., “Test Generation” tools

- 2: derive test cases and measure coverage level:

Recognizer, i.e., “Coverage Analysis” tools

Important Question

Given a coverage criterion C, how do you compare 90% coverage level to 100% coverage level ? Does this mean the former is 10% less effective in revealing faults?

Subsumption

How to compare different coverage criteria against each other?

How to decide if one coverage criterion is stronger/weaker than another one?

Subsumption: A test criterion C1 subsumes C2 if and only if every set of test cases that satisfies criterion C1 also satisfies C2

Example: Branch Coverage (BC) subsumes Statement Coverage (SC)

Question

Assume coverage criterion C1 subsumes coverage criteria C2. T1 is a test set that satisfies C1 on program P and T2 is another test set that satisfies C2 on P. It can be concluded that T1 detects at least as many faults in P as T2 does.

True of False? Justify your answer.

Advantages of Criteria-based Testing

- Maximize the “bang for the buck”
  - Fewer tests that are more effective at finding faults
- Comprehensive test set with minimal overlap
- “Traceability” from software artifacts to tests
- A “stopping rule” for testing
- Lend themselves well to “automation”

Question

Any downsides to (or concerns with!) *criteria-based testing?*

Limitations of Coverage Criteria

- Might not be easy/straightforward to generate all the test requirements
- Might not be easy/straightforward to generate test cases that satisfy the generated test requirements
- Might still be very costly to achieve
- **Most important:** *what is the correlation, if any, between a coverage criterion satisfaction and its fault detection ability?*

Question

Suppose that coverage criterion C1 subsumes coverage criterion C2. Further suppose that test set T1 satisfies C1 and on program P test set T2 satisfies C2, also on P.

1. Does T1 necessarily satisfy C2? Explain.
2. Does T2 necessarily satisfy C1? Explain.
3. If P contains a fault, and T2 reveals the fault, T1 does not necessarily also reveal the fault. Explain

Combinatorial Coverage Criteria

- We already discussed equivalence partitioning (EP) technique when we talked about Blackbox testing
- We learned how to do EP on a domain
  - e.g., domain is integer values → negative, zero, positive
- In practice though, oftentimes we have several input/output domain to work with
  - e.g., a function that has more than one input parameter

We need to work with combinations of equivalence blocks

Example

```
/**
* Count the number of occurrences of a target value in an ArrayList
* @param ray the ArrayList instance
* @param key the target value
* @return count of occurrences
*/
public static int countOf(ArrayList<Integer> ray, int key) {
   int count = 0;
   for (int i = 0; i < ray.size(); ++i) {
      if (ray.get(i).equals(key))  {
         count++;
      }
   }
   return count;
}
```

All Combinations Coverage

All Combinations (ACoC) : All combinations of blocks from all characteristics must be used.

`countOf` example with the following blocks:

A. ArrayList null
B. ArrayList empty
C. ArrayList size 1
D. ArrayList size > 1

1. key negative
2. key zero
3. key positive

Test requirements for ACoC would include all possible combinations of the blocks:

(A, 1), (A, 2), …, (B, 1), …, (D, 3)

Each Choice Coverage

Each Choice Coverage (ECC) : One value from each block for each characteristic must be used in at least one test case.

`countOf` example with the following blocks:

A. ArrayList null
B. ArrayList empty
C. ArrayList size 1
D. ArrayList size > 1

1. key negative
2. key zero
3. key positive

Test requirements for ECC would be:

(A, 1), (B, 2), (C, 3), (D, 1)

Pair-Wise Coverage

Pair-Wise Coverage (PWC) : A value from each block for each characteristic must be combined with a value from every block for each of the other characteristics.

Assume we have three partitions with the following blocks:

[A, B], [1, 2, 3], and [x, y]

Test requirements for PWC would be:

(A, 1, x) (B, 1, y)
(A, 2, x) (B, 2, y)
(A, 3, x) (B, 3, y)
(A, 1, y) (B, 1, x)

t-Wise Coverage

t-Wise Coverage (TWC) : A value from each block for each group of t characteristics must be combined.

Assume we have three partitions with the following blocks:

[A, B], [1, 2, 3], and [x, y]

Test values for 3-wise coverage would be the same as ACoC, why?

Base Choice Coverage

Base Choice Coverage (BCC) : A base choice block is chosen for each characteristic, and a base test is formed by using the base choice for each characteristic. Subsequent tests are chosen by holding all but one base choice constant and using each non-base choice in each of the other characteristics.

Assume we have three partitions with the following blocks where A, 1, and x are base choices:

[A, B], [1, 2, 3], and [x, y]

Test values for BCC coverage would be:

(A, 1, x) **base test**

(B, 1, x) ←

(A, 2, x)

(A, 3, x)

(A, 1, y)

Base Choice Selection

- “base choice” essentially corresponds to the “default” choice for a block
- Base choice should be feasible (i.e., executable)
- Base choices usually take the “happy path”
  - E.g., base choice block for “factor” is positive values

```
void multiples(int factor) {
    if (factor <= 0) {
        System.out.println(“Provide a positive value!");
    } else {
        // Display consecutive positive factors until 100
        int value = factor;
        while (value <= 100) {
            System.out.print(value + " ");
            value = value + factor;
        }
    }
}
```

Relevant Reads & Resources

- Recommended Textbooks:
  - Introduction to Software Testing, 2nd Edition: ch5 and ch6