Introduction to Software Testing
Input Space Partition Testing (Ch. 6.2)

Software Testing & Maintenance

SWE 437

http://go.gmu.edu/swe437

Dr. Brittany Johnson-Matthews

(Dr. B for short)

Modeling the input domain

Step 1: Identify testable functions

Step 2: Find all **inputs, parameters, & characteristics**

Move from imp level to design abstraction level

Step 3: Model the **input domain**

Step 4: Apply a test **criterion** to choose **combinations** of values (6.2)

Entirely at the design abstraction level

Step 5: Refine combinations of blocks into **test inputs**

Back to the implementation abstraction level

Modeling the input domain

Step 1: Identify testable functions

Step 2: Find all **inputs, parameters, & characteristics**

Move from imp level to design abstraction level

Step 3: Model the **input domain**

Step 4: Apply a test **criterion** to choose **combinations** of values (6.2)

Entirely at the design abstraction level

Step 5: Refine combinations of blocks into **test inputs**

Back to the implementation abstraction level

Step 4 – Choosing Combinations of Values

After partitioning characteristics into blocks, testers design tests by combining blocks from different characteristics

- 3 characteristics: A, B, C
- Three blocks each: A = a1, a2, a3; B = b1, b2, b3; C = c1, c2, c3

A test starts by combining one block from each characteristic

- Then values are chosen to satisfy the combinations

We use **criteria** to choose **effective** combinations

All Combinations Criterion (ACoC)

The most obvious criterion is to choose all combinations.

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

a1 b1 c1	a2 b1 c1	a3 b1 c1
a1 b1 c2	a2 b1 c2	a3 b1 c2
a1 b1 c3	a2 b1 c3	a3 b1 c3
a1 b2 c1	a2 b2 c1	a3 b2 c1
a1 b2 c2	a2 b2 c2	a3 b2 c2
a1 b2 c3	a2 b2 c3	a3 b2 c3
a1 b3 c1	a2 b3 c1	a3 b3 c1
a1 b3 c2	a2 b3 c2	a3 b3 c2
a1 b3 c3	a2 b3 c3	a3 b3 c3

All Combinations Criterion (ACoC)

Number of tests is the product of the number of blocks in each characteristic:

$$\prod_{i=1}^{Q} (B_i)$$

The syntax characterization of triang()

-Each side: >1, 1, 0, <1

-Results in 4*4*4 = **64 tests**

Most form invalid triangles

How can we get fewer tests?

In-class Exercise

All Combinations Criterion *(ACoC)*

Consider our previous example.

3 characteristics: A, B, C

Three blocks each: A = a1, a2, a3; B = b1, b2, b3; C = c1, c2, c3

How many tests do we need to satisfy ACoC?

In-class Exercise

All Combinations Criterion *(ACoC)*

Consider our previous example.

3 characteristics: A, B, C

Three blocks each: A = a1, a2, a3; B = b1, b2, b3; C = c1, c2, c3

How many tests do we need to satisfy ACoC?

3 * 3 * 3 = **27 tests**

ISP Criteria – Each Choice (ECC)

We should try **at least one** value from each block

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

Number of tests is the number of blocks in the **largest** characteristic:

$$\mathrm{Max}_{i=1}^{Q}(B_i)$$

In-class Exercise

Each Choice Criterion *(ECC)*

3 characteristics: A, B, C

Three blocks each: A = a1, a2, a3; B = b1, b2, b3; C = c1, c2, c3

1. *How many tests do we need (with ECC)?*
2. *Write the (abstract) tests.*

In-class Exercise

Each Choice Criterion *(ECC)*

3 characteristics: A, B, C

Three blocks each: A = a1, a2, a3; B = b1, b2, b3; C = c1, c2, c3

1. *How many tests do we need (with ECC)?*
   Max # of blocks is 3 → **minimum of 3 tests**
2. *Write the (abstract) tests.*
   (a1, b1, c1); (a2, b2, c2); (a3, b3, c3)

ISP Criteria – Base Choice (BCC)

ECC is **simple**, but very few tests

The **base choice criterion** recognizes that

-Some blocks are more **important** than others

-Using **diverse combinations** can strengthen testing

Let testers bring in **domain knowledge** of the program

> 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 other characteristic.

Number of tests is one base test + one test for each "non-base" other block:

$$1 + \sum_{i=1}^{Q} (B_i - 1)$$

Base choice considerations

The base test must be **feasible**

-That is, all base choices must be **compatible**

Base choices can be

-Most likely from an end-use point of view

-Simplest

-Smallest

-First in some ordering

Happy path tests often make good base choices

The base choice is a **crucial design** decision

-Test designers should **document** why the choices were made

In-class Exercise

Base Choice Criterion *(BCC)*

Write BCC tests for our previous example.

3 characteristics: A, B, C

Three blocks each: A = a1, a2, a3; B = b1, b2, b3; C = c1, c2, c3

1. *How many tests do we need?*
2. *Pick base values and write one base test*
3. *Write remaining tests*

In-class Exercise

Base Choice Criterion *(BCC)*

3 characteristics: A, B, C

Three blocks each: A = a1, a2, a3; B = b1, b2, b3; C = c1, c2, c3

1. *How many tests do we need?*
   1 + 2 + 2 + 2 = **7 tests**
2. *Pick base values and write one base test*
   base values → **a2, b3, c1**
   base test → **(a2, b3, c1)**
3. *Write remaining tests*
   (a2,**b2**, c1); (a2, **b1**, c1); (a2, b3, **c2**);
   (a2, b3, **c3**); (**a1**, b3, c1); (**a3**, b3, c1)

ISP Criteria – Multiple Base Choice (MBCC)

We sometimes have **more than one** logical base choice

<u>Multiple Base Choice Coverage (MBCC)</u> At least one, and possibly more, base choice blocks are chosen for each characteristic, and base tests are formed by using each base choice for each characteristic at least once. Subsequent tests are chosen by holding all but one base choice constant for each base test and using each non-base choice in each other characteristic

If ***M*** base tests and $\boldsymbol{m_i}$ base choices for each characteristic:

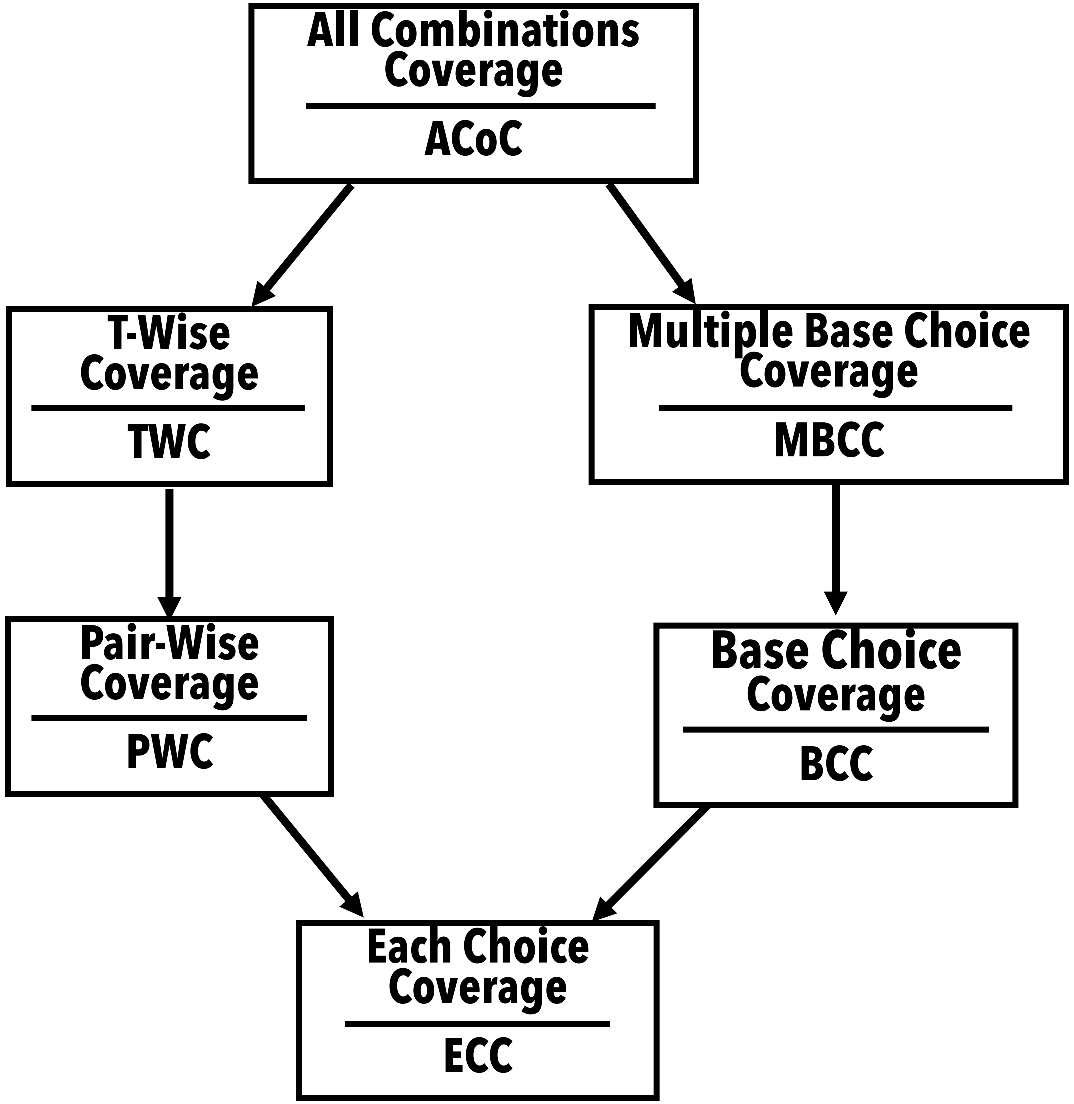
$$M + \sum_{i=1}^{Q} (M * (B_i - m_i))$$

For our example… two base tests: **a1, b1, c1** **a2, b2, c2**

Tests from **a1, b1, c1**: a1, b1, **c3**; a1, **b3**, c1; **a3**, b1, c1

Tests from **a2, b2, c2**: a2, b2, **c3**; a2, **b3**, c2; **a3**, b2, c2

ISP Coverage Criteria Subsumption



Input Space Partitioning Summary

Fairly easy to apply, even with **no automation**

Convenient ways to **add more or less** testing

Equally applicable to **all levels** of testing – unit, class, integration, system, etc.

Based only on the **input space** of the program, not the implementation

Simple, straight-forward, effective, and widely used.