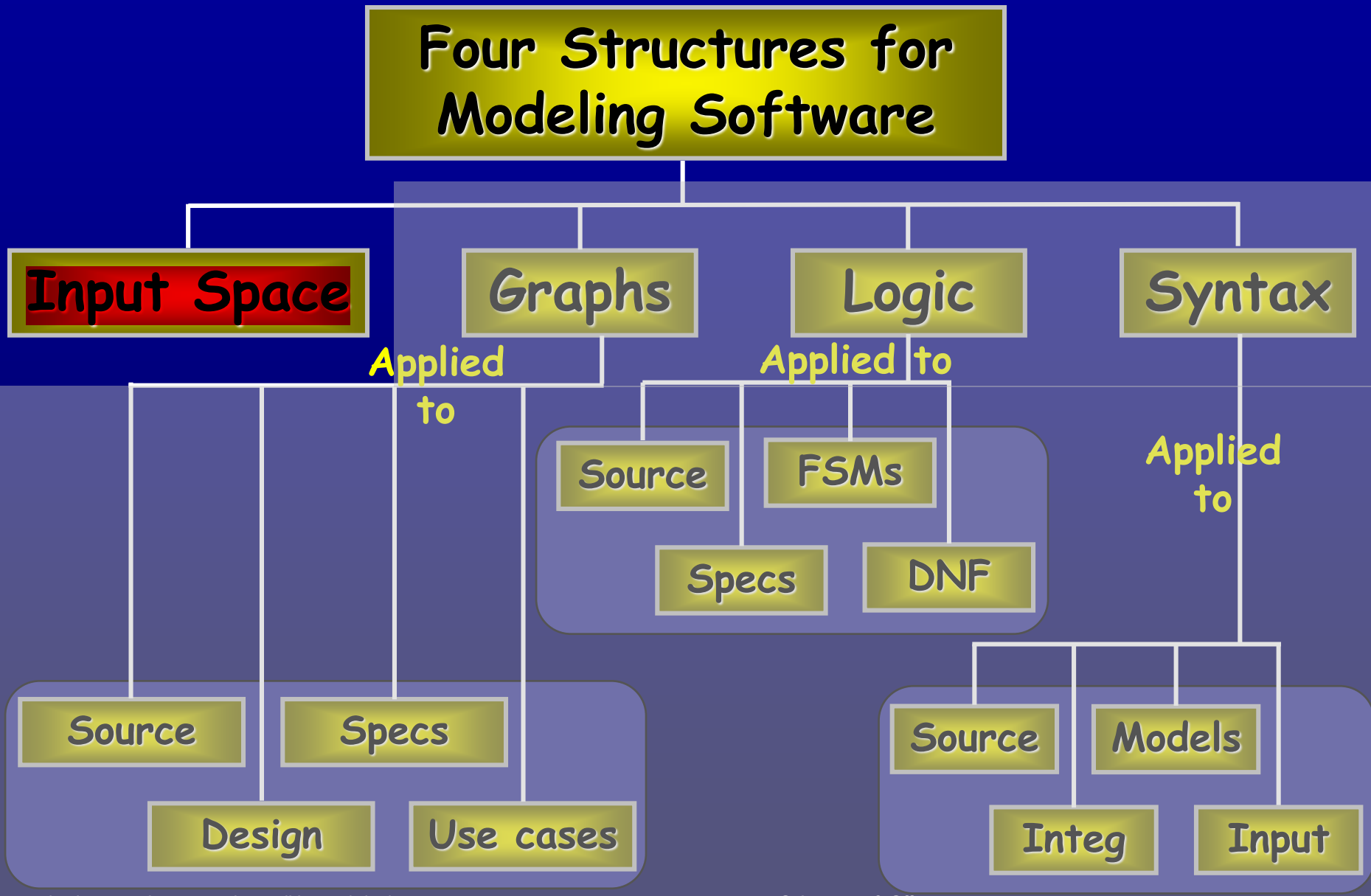
Introduction to Software Testing
Chapter 6
Input Space Partition Testing

Paul Ammann & Jeff Offutt

http://www.cs.gmu.edu/~offutt/softwaretest/

Ch. 6 : Input Space Coverage



Modeling the Input Domain

- Step 1 : Identify testable functions
- Step 2 : Find all the parameters that can affect the behavior of a given testable function.
- Step 3 : Model the input domain
  - The domain is scoped by the parameters
  - The structure is defined in terms of characteristics
  - Each characteristic is partitioned into sets of blocks
  - Each block represents a set of values
  - This is the most creative design step in using ISP

Modeling the Input Domain (*cont*)

- Step 4 : Apply a test criterion to choose combinations of values
  - A test input has a value for each parameter
  - One block for each characteristic
  - Choosing all combinations is usually infeasible
  - Coverage criteria allow subsets to be chosen
- Step 5 : Refine combinations of blocks into test inputs
  - Choose appropriate values from each block

Step 4 – Choosing Combinations of Values (6.2)

- Once characteristics and partitions are defined, the next step is to choose test values
- We use criteria – to choose effective subsets
- The most obvious criterion is to choose all combinations

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

- Number of tests is the product of the number of blocks in each characteristic : $\prod_{i=1}^{Q}(B_i)$
- The second characterization of triang() results in 4*4*4 = 64 tests
  - Too many ?

ISP Criteria – All Combinations

- Consider the "second characterization" of Triang as given before:

Characteristic	b_1	b_2	b_3	b_4
q_1 = "Refinement of q_1"	greater than 1	equal to 1	equal to 0	less than 0
q_2 = "Refinement of q_2"	greater than 1	equal to 1	equal to 0	less than 0
q_3 = "Refinement of q_3"	greater than 1	equal to 1	equal to 0	less than 0

- For convenience, we relabel the blocks using abstractions:

Characteristic	b_1	b_2	b_3	b_4
A	A1	A2	A3	A4
B	B1	B2	B3	B4
C	C1	C2	C3	C4

ISP Criteria – ACoC Tests

A1 B1 C1	A2 B1 C1	A3 B1 C1	A4 B1 C1
A1 B1 C2	A2 B1 C2	A3 B1 C2	A4 B1 C2
A1 B1 C3	A2 B1 C3	A3 B1 C3	A4 B1 C3
A1 B1 C4	A2 B1 C4	A3 B1 C4	A4 B1 C4
A1 B2 C1	A2 B2 C1	A3 B2 C1	A4 B2 C1
A1 B2 C2	A2 B2 C2	A3 B2 C2	A4 B2 C2
A1 B2 C3	A2 B2 C3	A3 B2 C3	A4 B2 C3
A1 B2 C4	A2 B2 C4	A3 B2 C4	A4 B2 C4
A1 B3 C1	A2 B3 C1	A3 B3 C1	A4 B3 C1
A1 B3 C2	A2 B3 C2	A3 B3 C2	A4 B3 C2
A1 B3 C3	A2 B3 C3	A3 B3 C3	A4 B3 C3
A1 B3 C4	A2 B3 C4	A3 B3 C4	A4 B3 C4
A1 B4 C1	A2 B4 C1	A3 B4 C1	A4 B4 C1
A1 B4 C2	A2 B4 C2	A3 B4 C2	A4 B4 C2
A1 B4 C3	A2 B4 C3	A3 B4 C3	A4 B4 C3
A1 B4 C4	A2 B4 C4	A3 B4 C4	A4 B4 C4

ACoC yields 4*4*4 = 64 tests for Triang!

This is almost certainly more than we need

Only 8 are valid (all sides greater than zero)

ISP Criteria – Each Choice

- 64 tests for triang() is almost certainly way too many
- One criterion comes from the idea that we should try at least one value from each block

Each Choice Coverage (ECC) : 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 : $\mathbf{Max}_{i=1}^{Q}(B_i)$

For *triang*() : A1, B1, C1
A2, B2, C2
A3, B3, C3
A4, B4, C4

Write down EC tests
Use the abstract labels (A1,A2, …)

Substituting values: 2, 2, 2
1, 1, 1
0, 0, 0
-1, -1, -1

Suggest values …

ISP Criteria – Pair-Wise

- Each choice yields few tests—cheap but maybe ineffective
- Another approach combines values with other values

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

- Number of tests is at least the product of two largest characteristics $(\text{Max}_{i=1}^{Q}(B_i)) * (\text{Max}_{j=1,\, j!=i}^{Q}(B_j))$

For *triang()* :

A1, B1, C1	A1, B2, C2	A1, B3, C3	A1, B4, C4
A2, B1, C2	A2, B2, C3	A2, B3, C4	A2, B4, C1
A3, B1, C3	A3, B2, C4	A3, B3, C1	A3, B4, C2
A4, B1, C4	A4, B2, C1	A4, B3, C2	A4, B4, C3

Write down PWC tests
Use the abstract labels
(Hint: Should be 16 tests)

ISP Criteria –T-Wise

- A natural extension is to require combinations of *t* values instead of *2*

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

- Number of tests is at least the product of *t* largest characteristics
- If all characteristics are the same size, the formula is

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

- If *t* is the number of characteristics *Q*, then all combinations
- That is … *Q-wise = AC*
- *t*-wise is expensive and benefits are not clear

ISP Criteria – Base Choice

- Testers sometimes recognize that certain values are important
- This uses 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 other block $1 + \sum_{i=1}^{Q} (B_i - 1)$

Q=3,Bi=4

For *triang*() : Base A1, B1, C1 A1, B1, C2 A1, B2, C1 A2, B1, C1
A1, B1, C3 A1, B3, C1 A3, B1, C1
A1, B1, C4 A1, B4, C1 A4, B1, C1

Write down BCC tests

Base Choice Notes

- 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

ISP Criteria – Multiple Base Choice

- We sometimes have more than one logical base choice

Multiple Base Choice Coverage (MBCC) : 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 m_i base choices for each characteristic:**

M=2, Q=3

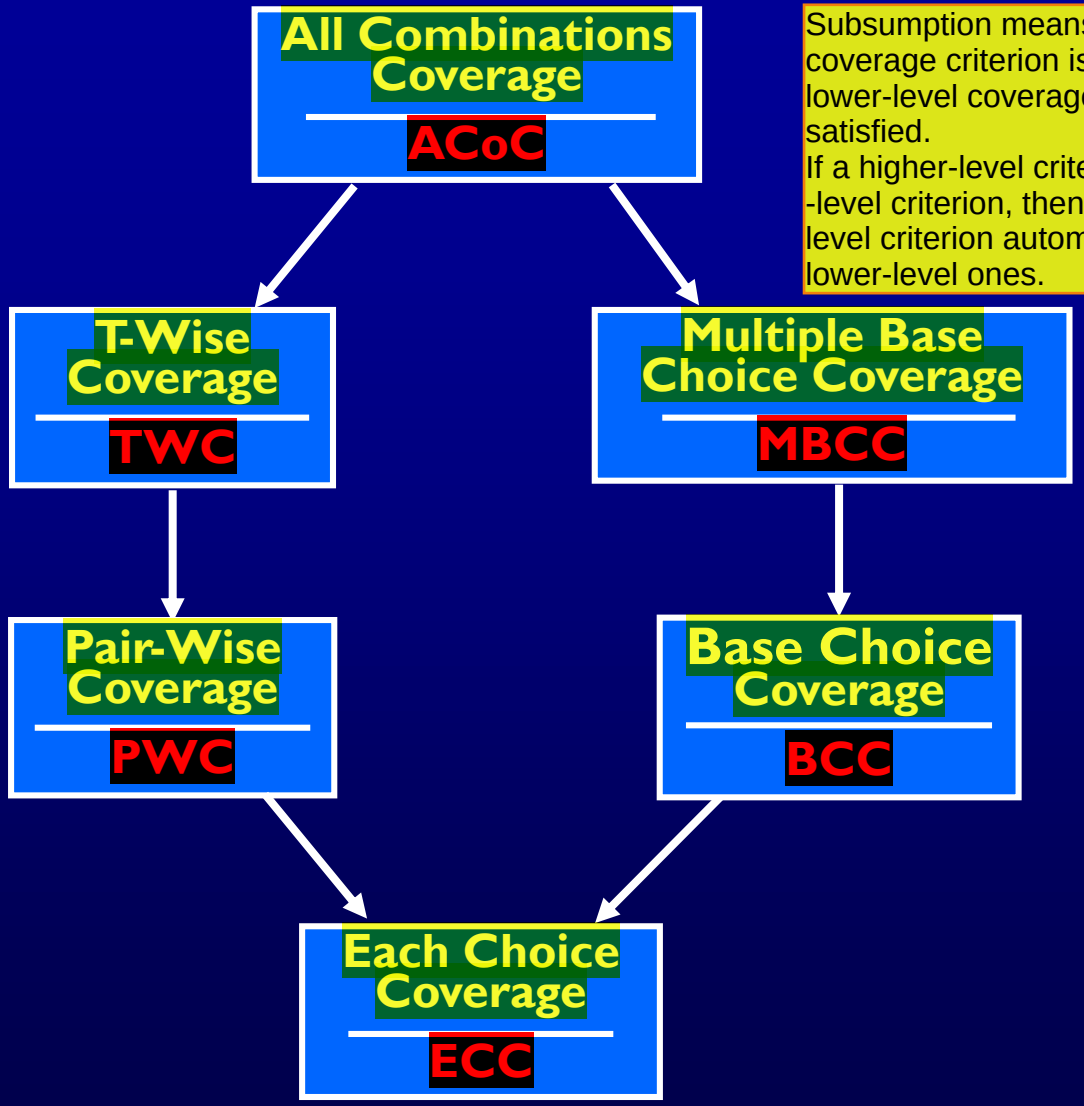
$$M + \sum_{i=1}^{Q} (M * (B_i - m_i))$$

Bi=4,mi=2

For *triang*() : Bases

A1, B1, C1	A1, B1, C3	A1, B3, C1	A3, B1, C1
	A1, B1, C4	A1, B4, C1	A4, B1, C1
A2, B2, C2	A2, B2, C3	A2, B3, C2	A3, B2, C2
	A2, B2, C4	A2, B4, C2	A4, B2, C2

ISP Coverage Criteria Subsumption



Subsumption means that if a higher-level coverage criterion is satisfied, then all lower-level coverage criteria are also satisfied.
If a higher-level criterion subsumes a lower -level criterion, then satisfying the higher-level criterion automatically satisfies the lower-level ones.

Constraints Among Characteristics (6.3)

- Some combinations of blocks are infeasible
  - "less than zero" and "scalene" … not possible at the same time
- These are represented as constraints among blocks
- Two general types of constraints
  - A block from one characteristic cannot be combined with a specific block from another
  - A block from one characteristic can ONLY BE combined with a specific block form another characteristic
- Handling constraints depends on the criterion used
  - ACC, PWC, TWC : Drop the infeasible pairs
  - BCC, MBCC : Change a value to another non-base choice to find a feasible combination

Example Handling Constraints

```
public boolean findElement (List list, Object element)
// Effects: if list or element is null throw NullPointerException
//          else return true if element is in the list, false otherwise
```

Characteristic	Block 1	Block 2	Block 3	Block 4
A : length and contents	One element	More than one, unsorted	More than one, sorted	More than one, all identical
B : match	element not found	element found once	element found more than once	
Invalid combinations : (A1, B3), (A4, B2)				

element cannot be in a one-element list more than once

If the list only has one element, but it appears multiple times, we cannot find it just once

Input Space Partitioning Summary

- Fairly easy to apply, even with no automation
- Convenient ways to add more or less testing
- Applicable to all levels of testing – unit, class, integration, system, etc.
- Based only on the input space of the program, not the implementation

Simple, straightforward, effective, and widely used