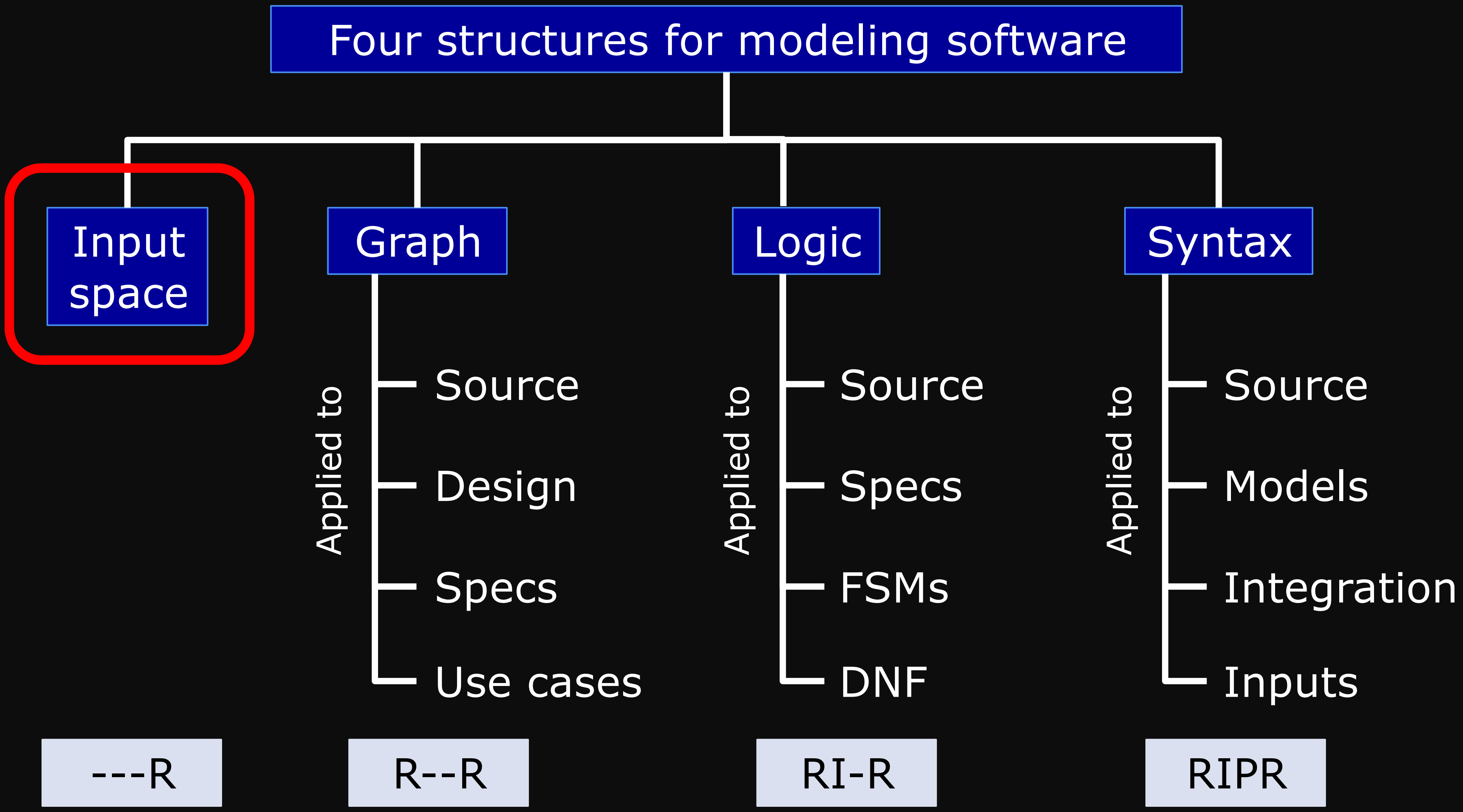
ISP Coverage Criteria

CS 3250
Software Testing

[Ammann and Offutt, "Introduction to Software Testing," Ch. 6.2-6.3]

Structures for Criteria-Based Testing



Today's Objectives

- How should we consider multiple partitions or IDMs at the same time?
- What combinations of blocks should we choose values from?
- How many tests should we expect?

Revisit

Applying ISP

Identify testable functions

⇓

Identify parameters, return types, return values, exceptional behavior

⇓

Model the input domain

The most creative design step in using ISP

⇓ Input Domain Model (IDMs)

Task I: Model input domain

Apply a test criterion to choose combinations of blocks

⇓ Test requirements (TRs)

Derive test inputs

⇓ Test cases

Task II: Choose combinations of values

Revisit

Modeling the Input Domain

- The domain is scoped by the parameters
- Characteristics define the structure of the input domain
  - Should be based on the input domain – not program source

- Two Approaches
  - Interface-based (simpler)
    - Develop characteristics from individual parameters
  - Functionality-based (harder)
    - Develop characteristics from a behavior view

Design characteristics

⇩

Partition each characteristic into blocks

⇩

Identify values of each block

Using Multiple Partitions or IDMs

- Some programs may have many parameters
- Typical to create several small IDMs – divide-and-conquer
- Some parameters may appear in more than one IDM
  - Leading to overlap IDMs
- Some IDMs may include specific constraints (such as invalid values)
- Multiple partitions or IDMs can be combined to create tests

How should we consider multiple partitions or IDMs at the same time?

Applying ISP

Identify testable functions

⇩

Identify parameters, return types, return values, exceptional behavior

⇩

Model the input domain

⇩ Input Domain Model (IDMs)

Task I: Model input domain

Apply a test criterion to choose combinations of blocks

⇩ Test requirements (TRs)

Derive test inputs

⇩ Test cases

Task II: Choose combinations of values

Today's focus

Running Example: triang()

- Partition characteristics

Characteristic	b1	b2	b3	b4
C1) = length of **Side1**	greater than 1	equal to 1	equal to 0	less than 0
C2) = length of **Side2**	greater than 1	equal to 1	equal to 0	less than 0
C3) = length of **Side3**	greater than 1	equal to 1	equal to 0	less than 0

- For convenience, let's relabel the blocks

Characteristic	b1	b2	b3	b4
A = length of **Side1**	A1	A2	A3	A4
B = length of **Side2**	B1	B2	B3	B4
C = length of **Side3**	C1	C2	C3	C4

- Possible values

Characteristic	b1	b2	b3	b4
A = length of **Side1**	2	1	0	-1
B = length of **Side2**	2	1	0	-1
C = length of **Side3**	2	1	0	-1

Choosing Combinations of Values

- Approaches to choose values
  - Select values randomly
    - Quality of tests depends on experience and expertise
  - Use coverage criteria – to choose effective subsets
    - Quality of tests depends on the strength of the criteria

- ISP Coverage criteria
  - All Combinations Coverage (ACoC)
  - Each Choice Coverage (EEC)
  - Pair-Wise Coverage (PWC)
  - ~~T-Wise Coverage (TWC) – expensive, unclear benefits~~
  - Base Choice Coverage (BCC)
  - Multiple Base Choice Coverage (MBCC)

All Combinations (ACoC)

All combinations of blocks from all characteristics must be used

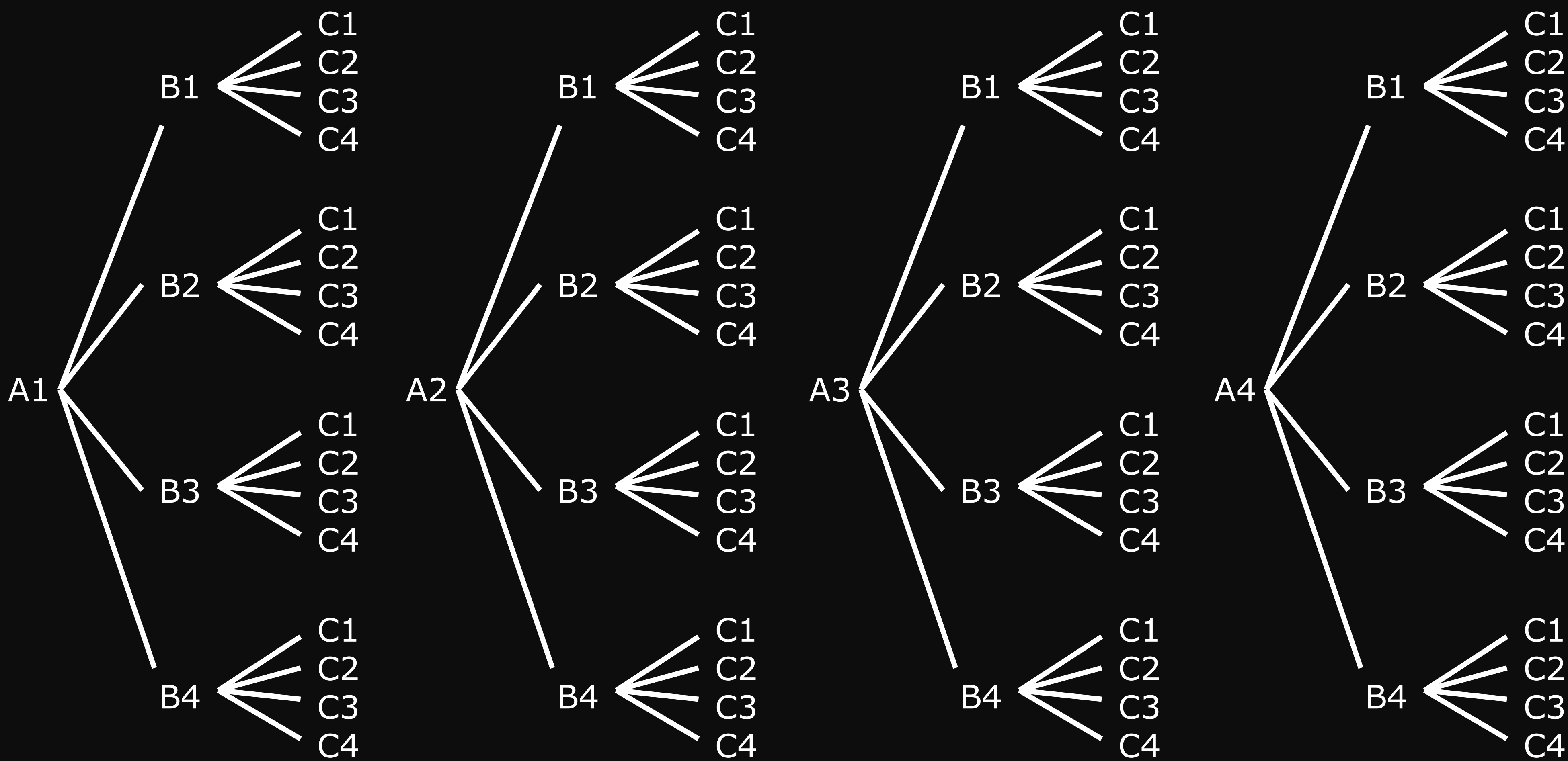
- Number of tests = $\prod_{i=1}^{Q} (B_i)$

  Q = number partitions (or characteristics), B = number blocks

- More tests → likely to find more faults
- More tests than necessary
- Impractical when more than two or three partitions are defined

ACoC - Example

- Applying ACoC to derive test requirements



ACoC – Example *(cont.)*

- Test requirements: 4*4*4 = 64 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)

This is almost certainly more than we need

Only 8 are valid (all sides greater than zero)

Zero length:
A3, B3, C3

Negative length:
A4, B4, C4

ACoC – Example *(cont.)*

- Substituting test input values

(2, 2, 2)	(1, 2, 2)	(0, 2, 2)	(-1, 2, 2)
(2, 2, 1)	(1, 2, 1)	(0, 2, 1)	(-1, 2, 1)
(2, 2, 0)	(1, 2, 0)	(0, 2, 0)	(-1, 2, 0)
(2, 2, -1)	(1, 2, -1)	(0, 2, -1)	(-1, 2, -1)
(2, 1, 2)	(1, 1, 2)	(0, 1, 2)	(-1, 1, 2)
(2, 1, 1)	(1, 1, 1)	(0, 1, 1)	(-1, 1, 1)
(2, 1, 0)	(1, 1, 0)	(0, 1, 0)	(-1, 1, 0)
(2, 1, -1)	(1, 1, -1)	(0, 1, -1)	(-1, 1, -1)
(2, 0, 2)	(1, 0, 2)	(0, 0, 2)	(-1, 0, 2)
(2, 0, 1)	(1, 0, 1)	(0, 0, 1)	(-1, 0, 1)
(2, 0, 0)	(1, 0, 0)	(0, 0, 0)	(-1, 0, 0)
(2, 0, -1)	(1, 0, -1)	(0, 0, -1)	(-1, 0, -1)
(2, -1, 2)	(1, -1, 2)	(0, -1, 2)	(-1, -1, 2)
(2, -1, 1)	(1, -1, 1)	(0, -1, 1)	(-1, -1, 1)
(2, -1, 0)	(1, -1, 0)	(0, -1, 0)	(-1, -1, 0)
(2, -1, -1)	(1, -1, -1)	(0, -1, -1)	(-1, -1, -1)

Substituting values before refining TRs → Useless tests

Refining TRs by eliminating redundant and infeasible tests

Always refine TRs before deriving test values

Different choices of values from the same block are equivalent from a testing perspective. Thus, we need only one value from each block

Each Choice (ECC)

One value from each block for each characteristic must be used in at least one test case

- Number of tests = $\mathit{Max}_{i=1}^{Q} (B_i)$

  Q = number partitions (or characteristics), B = number blocks

- Flexibility in terms of how to combine the test values
- Fewer tests → cheap but may be ineffective
- Not require values to be combined with other values
  → weak criterion

ECC – Example

- Applying ECC to derive test requirements

Blocks for characteristic A = {A1, A2, A3, A4}
Blocks for characteristic B = {B1, B2, B3, B4}
Blocks for characteristic C = {C1, C2, C3, C4}

Possible combination

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

Another possible combination

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

ECC – Example *(cont.)*

- Test requirements: Max number of blocks = 4

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

- Substituting test input values

  (2, 2, 2)
  (1, 1, 1)
  (0, 0, 0)
  (-1, -1, -1)

What are missing?

Testers sometimes recognize that certain values are important. To strengthen ECC, domain knowledge of the program must be incorporated.

What is the most important block for each partition?

Pair-Wise (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 = $\left(Max_{i=1}^{Q} (B_i) \right) * \left(Max_{j=1,\, j!=i}^{Q} (B_j) \right)$

  Q = number partitions (or characteristics), B = number blocks

- Allow the same test case to cover more than one unique pair of values

PWC – Example 1: triang()

- Applying PWC to derive test requirements

  Blocks for characteristic A = {A1, A2, A3, A4}
  Blocks for characteristic B = {B1, B2, B3, B4}
  Blocks for characteristic C = {C1, C2, C3, C4}

- Number of tests = 4 * 4 = 16

- Test requirements
  - It is simpler to list the combinations in a table (see next slide)

Pair-Wise – Example 1

TR	A	B	C
1	A1	B1	C1
2	A1	B2	C2
3	A1	B3	C3
4	A1	B4	C4
5	A2	B1	C1
6	A2	B2	C2
7	A2	B3	C3
8	A2	B4	C4
9	A3	B1	C1
10	A3	B2	C2
11	A3	B3	C3
12	A3	B4	C4
13	A4	B1	C1
14	A4	B2	C2
15	A4	B3	C3
16	A4	B4	C4

TR = Test requirement

- Order characteristics in columns, from max number of blocks
- Fill the first column, repeat as many times as the number of the next max blocks
- File the second column
- Ensure each block of A pairs with all possible blocks of B. Swap as needed
- Fill the third column

Pair-Wise – Example 1

TR	A	B	C
1	A1	B1	C1
2	A1	B2	C2
3	A1	B3	C3
4	A1	B4	C4
5	A2	B1	C2
6	A2	B2	C3
7	A2	B3	C4
8	A2	B4	C1
9	A3	B1	C3
10	A3	B2	C4
11	A3	B3	C1
12	A3	B4	C2
13	A4	B1	C4
14	A4	B2	C1
15	A4	B3	C2
16	A4	B4	C3

TR = Test requirement

- Order characteristics in columns, from max number of blocks
- Fill the first column, repeat as many times as the number of the next max blocks
- File the second column
- Ensure each block of A pairs with all possible blocks of B. Swap as needed
- Fill the third column
- Ensure each block of B pairs with all possible of blocks of C. Swap as needed
- Ensure each block of A pairs with all possible blocks of C. Swap as needed

Pair-Wise – Example 1

TC	A	B	C	Expected output
1	2	2	2	
2	2	1	1	
3	2	0	0	
4	2	-1	-1	
5	1	2	1	
6	1	1	0	
7	1	0	-1	
8	1	-1	2	
9	0	2	0	
10	0	1	-1	
11	0	0	2	
12	0	-1	1	
13	-1	2	-1	
14	-1	1	2	
15	-1	0	1	
16	-1	-1	0	

Reminder: two mandatory components of a test case are test inputs and expected output

TC = Test case

Substituting test input values

Pair-Wise – Example 2

- Applying PWC to derive test requirements

  Blocks for characteristic C1 = {A, B}
  Blocks for characteristic C2 = {1, 2, 3}
  Blocks for characteristic C3 = {x, y}

- Number of tests = 3 * 2 = 6

TR	C2	C1	C3
1	1	A	x
2	1	B	y
3	2	A	x
4	2	B	y
5	3	A	~~x~~ y
6	3	B	~~y~~ x

Base Choice (BCC)

A base choice block is chosen for each characteristic.

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

  Q = number partitions (or characteristics), *B* = number blocks

- Use domain knowledge of the program
  - *What is the most important block for each partition?*
- Pick the base choice test, then add additional tests
- Test quality depends on the selection of the base choice

BCC – Example

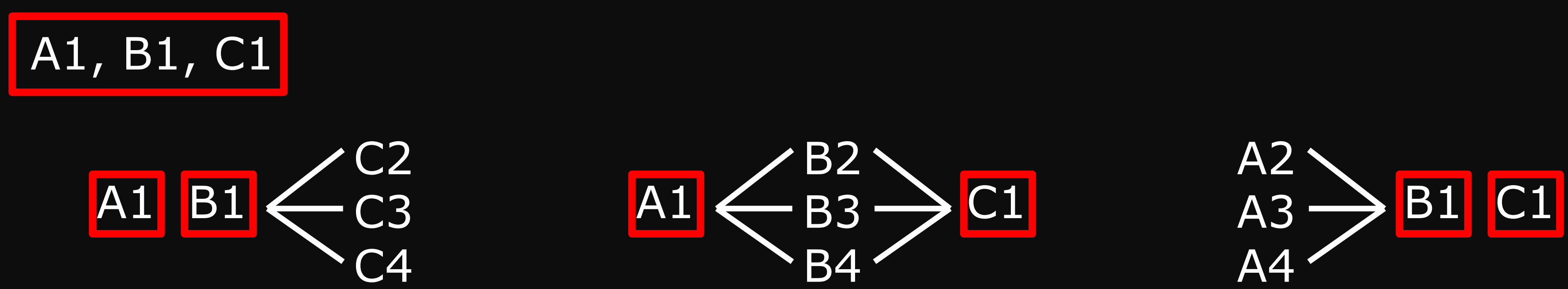
- Applying BCC to derive test requirements

  Blocks for characteristic A = {A1, A2, A3, A4}
  Blocks for characteristic B = {B1, B2, B3, B4}
  Blocks for characteristic C = {C1, C2, C3, C4}

  Suppose base choice blocks are A1, B1, and C1

  Then the base choice test is (A1, B1, C1)

  Hold all but one base choice constant, use each non-base choice in each other characteristic



BCC – Example *(cont)*

- Test requirements: 1 + 3 + 3 + 3 = 10

(A1, B1, C1) Base	(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)

- Substituting test input values

(2, 2, 2) Base	(2, 2, 1)	(2, 1, 2)	(1, 2, 2)
	(2, 2, 0)	(2, 0, 2)	(0, 2, 2)
	(2, 2, -1)	(2, -1, 2)	(-1, 2, 2)

BCC – Notes

- The base test must be feasible
- Base choices can be
  - From an end-user point of view
  - Simplest
  - Smallest
  - First in some order
  - Happy path test
- The base choice is a crucial design decision as it affects the quality of testing
  - Test designers should always document why the choices were made

Testers sometimes have multiple logical base choices

Multiple Base Choice (MBCC)

At least one, and possible more, base choice blocks are chosen from each characteristic.

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.

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

  M = number base tests
  m_i = number base choices for each characteristic
  Q = number partitions (or characteristics)
  B = number blocks

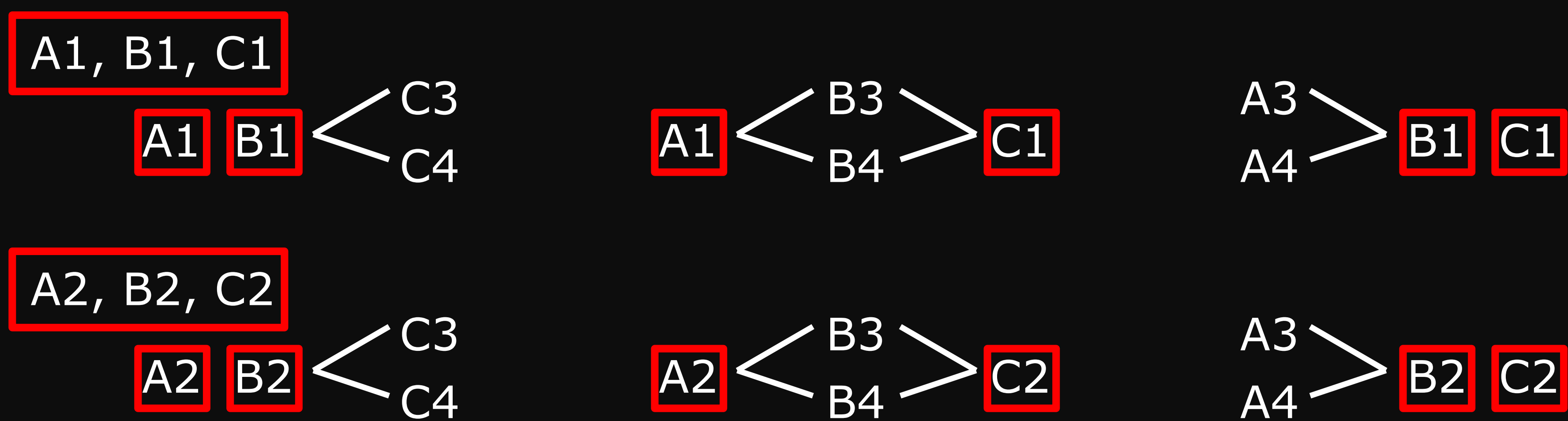
MBCC – Example

- Applying MBCC to derive test requirements

  Blocks for characteristic A = {A1, A2, A3, A4}
  Blocks for characteristic B = {B1, B2, B3, B4}
  Blocks for characteristic C = {C1, C2, C3, C4}

  Suppose base choice blocks are A1, B1, C1 and A2, B2, C2
  Then the base choice tests are (A1, B1, C1) and (A2, B2, C2)

  Hold all but one base choice constant for each base test, use each non-base choice in each other characteristic



MBCC – Example *(cont.)*

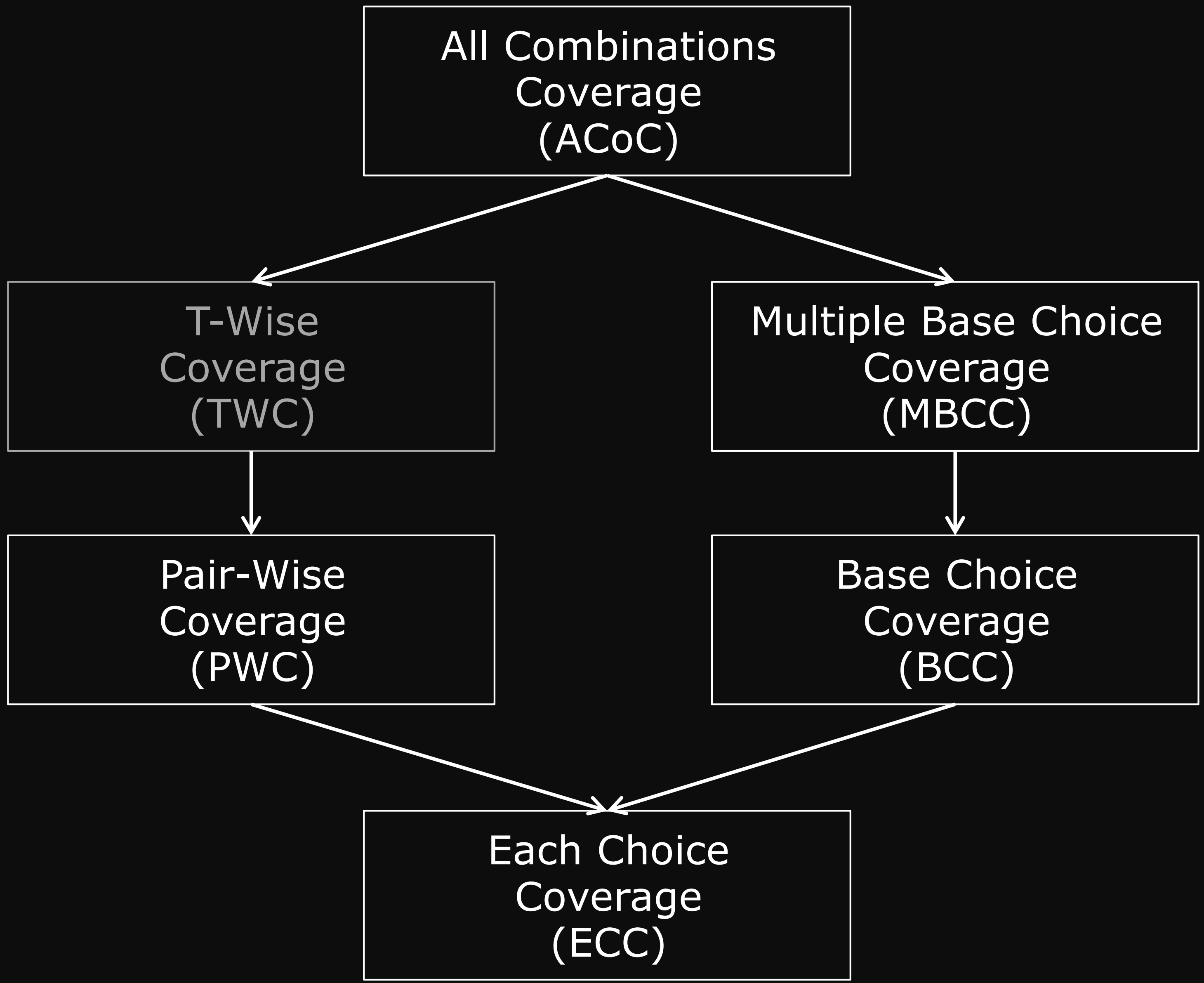
- Test requirements: 2+(2*(4-2))+(2*(4-2))+(2*(4-2)) = 14

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

- Substituting test input values

(2, 2, 2) Base	(2, 2, 0)	(2, 0, 2)	(0, 2, 2)
	(2, 2, -1)	(2, -1, 2)	(-1, 2, 2)
(1, 1, 1) Base	(1, 1, 0)	(1, 0, 1)	(0, 1, 1)
	(1, 1, -1)	(1, -1, 1)	(-1, 1, 1)

ISP Coverage Criteria Subsumption



Constraints Among Characteristics

- Some combinations of blocks are infeasible
  - A triangle cannot be "less than 0" and "scalene" at the same time
- These are represented as constraints among blocks
- Two kinds of constraints
  - A block from one characteristic cannot be combined with a block from another characteristic
  - A block from one characteristic must be combined with a specific block from another characteristic
- Handling constraints depends on the criterion used
  - ACoC – drop the infeasible pairs
  - ECC – change a value to find a feasible combination
  - BCC, MBCC – change a value to another non-base choice to find a feasible combination

Handling Constraints - Example

```
# Return index of the first occurrence of a letter in string,
# Otherwise, return -1

def get_index_of(string, letter):
```

Characteristic	b1	b2	b3
C1 = number of occurrence of **letter** in **string**	0	1	> 1
C2 = **letter** occurs first in **string**	True	False	
Invalid combinations: (C1b1)			

If a `letter` cannot be found in `string`, it cannot appear first in `string`

Summary

- Sometimes testers decide to use more than one IDM
- Once characteristics and partitions are defined, criteria are used to choose the combinations of test values
- Different criteria provide different coverage and result in different number of test requirements (and hence testing effort)
- ACoC may not be practical
- ECC may be too simplistic and ineffective
- BCC and MBCC pick meaningful blocks → *"do smarter"*

ISP testing is simple, straightforward, effective, and widely used