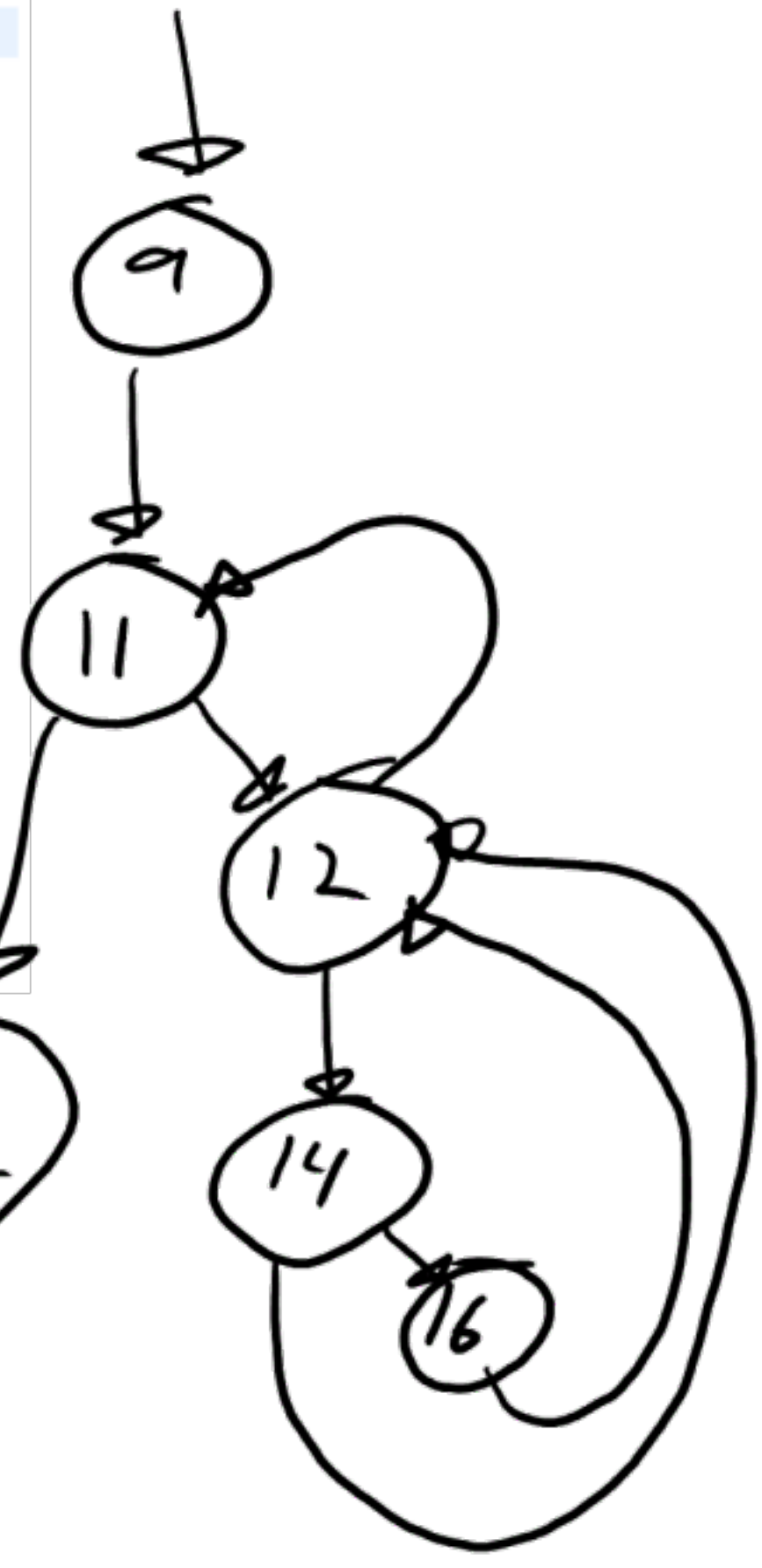
Input Domain Testing

Lecture Objectives:

1) Explain the concept of an input domain
2) List the steps necessary to perform input domain modeling.
3) Compare and contrast interface based input domain modeling with functionality based input domain modeling.

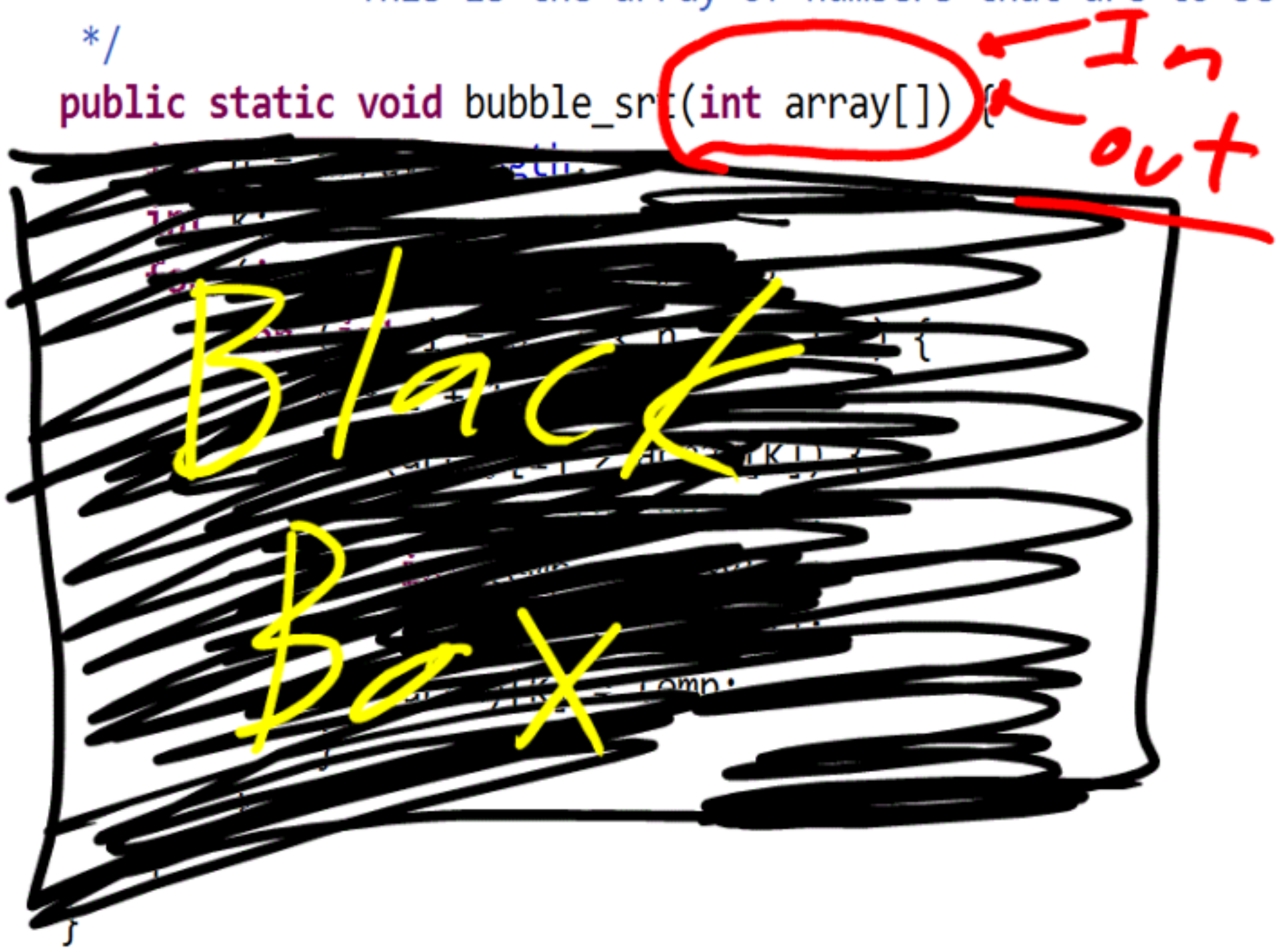
Problem: I would like to test this...

```
3       /**
4        * This method will sort the array of numbers using a bubble sort.
5        * @param array
6        *            This is the array of numbers that are to be sorted.
7        */
8       public static void bubble_srt(int array[]) {
9           int n = array.length;
10          int k;
11          for (int m = n; m >= 0; m--) {
12              for (int i = 0; i < n - 1; i++) {
13                  k = i + 1;
14                  if (array[i] > array[k]) {
15                      // Swap the numbers.
16                      int temp = array[i];
17                      array[i] = array[k];
18                      array[k] = temp;
19                  }
20              }
21          }
22      }
```

Problem: I would like to test this...

```
3    /**
4     * This method will sort the array of numbers using a bubble sort.
5     * @param array
6     *            This is the array of numbers that are to be sorted.
7     */
8    public static void bubble_sr(int array[]) {
9
10
11
12                                                    {
13
14
15
16
17
18                                    temp;
19
20
21
22   }
```



Test Cases:

{}

{1,2,3}

{2,1,3}

{1 2 3 4 5}

{5 4 3 2 1}

{-1 -2 -3 -4 -5}

{7} NULL

{1 3 5 2 4}

{-1 2 4 -7 -11}

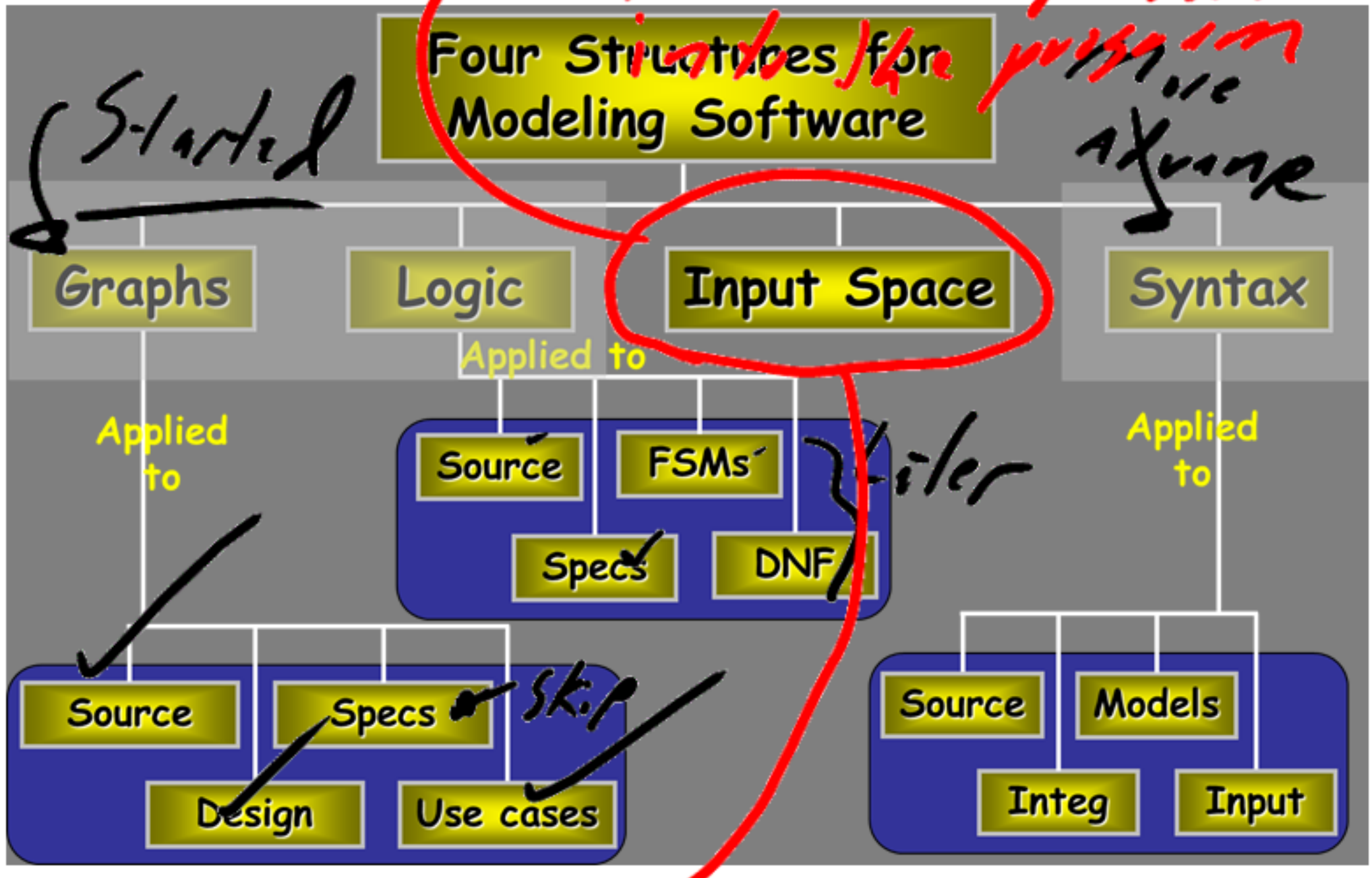
{1 2 3 11 5 6

55 55 77 ....

1001 ....

}

Input Space Coverage



Input Domains

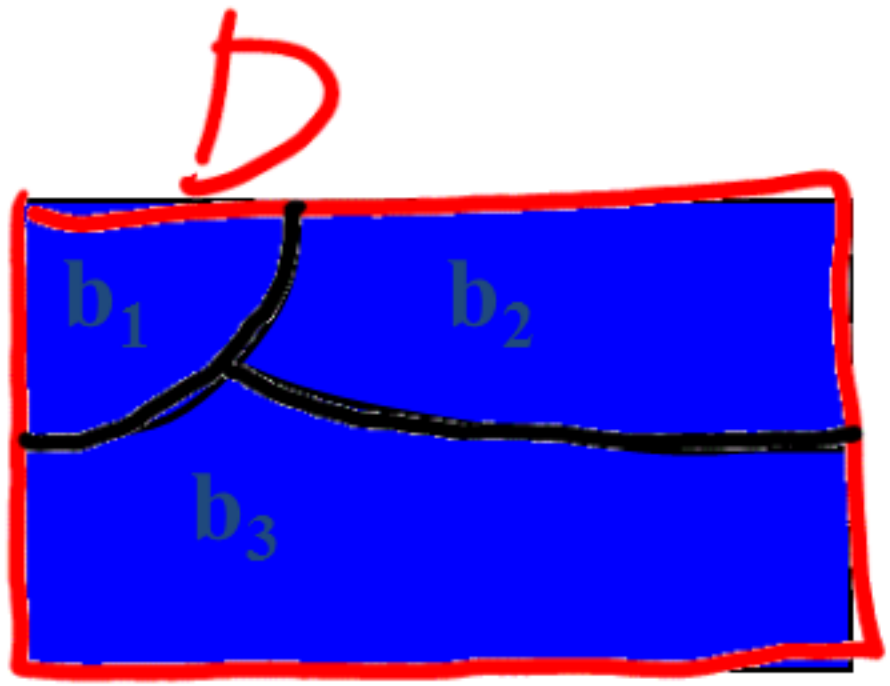
- The input domain to a program contains all the possible inputs to that program
- For even small programs, the input domain is so large that it might as well be infinite
- Testing is fundamentally about choosing finite sets of values from the input domain
- *Input parameters* define the scope of the input domain
  - Parameters to a method
  - Data read from a file
  - Global variables
  - User level inputs
- Domain for each input parameter is partitioned into regions
- At least one value is chosen from each region

Benefits of ISP

- Can be equally applied at several levels of testing
  - Unit
  - Integration
  - System
- Relatively easy to apply with no automation
- Easy to adjust the procedure to get more or fewer tests
- No implementation knowledge is needed
  - just the input space

Partitioning Domains

- *Domain* D
- *Partition scheme* q of D
- The partition q defines a *set of blocks*, $Bq = b_1, b_2, \ldots b_Q$
- The partition must satisfy two properties :
  1. blocks must be *pairwise disjoint* (no overlap)
  2. together the blocks *cover* the domain D (complete)



$$b_i \cap b_j = \Phi, \forall\, i \neq j, b_i, b_j \in B_q$$

$$\bigcup_{b \in Bq} b = D$$

Using Partitions – Assumptions

- Choose a value from each partition
- Each value is assumed to be equally useful for testing
- Application to testing
  - Find characteristics in the inputs : parameters, semantic descriptions, …
  - Partition each characteristic
  - Choose tests by combining values from characteristics
- Example Characteristics
  - Input X is null
  - Order of the input file F (sorted, inverse sorted, arbitrary, …)
  - Min separation of two aircraft
  - Input device (DVD, CD, VCR, computer, …)

Empty Set / Nonempty Set

Sorted

Positive Negative both

ODD / Even Length

Non-Sorted

Random

Reverse Sorted

{0, 2, 4}

{-7, -5, -3, -1}

Choosing Partitions

- Choosing (or defining) partitions seems easy, but is easy to get wrong
- Consider the "*order of file F*"

b_1 = sorted in ascending order

b_2 = sorted in descending order

b_3 = arbitrary order

but … something's fishy …

What if the file is of length 1?

The file will be in all three blocks …

That is, disjointness is not satisfied

Solution:

Each characteristic should address just one property

File F sorted ascending

- b1 = true
- b2 = false

File F sorted descending

- b1 = true
- b2 = false

Properties of Partitions

- If the partitions are not complete or disjoint, that means the partitions have not been considered carefully enough

- They should be reviewed carefully, like any design attempt

- Different alternatives should be considered

- We model the input domain in five steps …

Modeling the Input Domain

- Step 1 : Identify testable functions
  - Individual methods have one testable function
  - In a class, each method often has the same characteristics
  - Programs have more complicated characteristics—modeling documents such as UML use cases can be used to design characteristics
  - Systems of integrated hardware and software components can use devices, operating systems, hardware platforms, browsers, etc

- Step 2 : Find all the parameters
  - Often fairly straightforward, even mechanical
  - Important to be complete
  - Methods : Parameters and state (non-local) variables used
  - Components : Parameters to methods and state variables
  - System : All inputs, including files and databases

- 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
- 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

Two Approaches to Input Domain Modeling

1. Interface-based approach
   - Develops characteristics directly from individual input parameters
   - Simplest application
   - Can be partially automated in some situations
2. Functionality-based approach
   - Develops characteristics from a behavioral view of the program under test
   - Harder to develop—requires more design effort
   - May result in better tests, or fewer tests that are as effective

Input Domain Model (IDM)

1. Interface-Based Approach

- Mechanically consider each parameter in isolation
- This is an easy modeling technique and relies mostly on syntax
- Some domain and semantic information won't be used
  - Could lead to an incomplete IDM
- Ignores relationships among parameters

2. Functionality-Based Approach

- Identify characteristics that correspond to the intended functionality
- Requires more design effort from tester
- Can incorporate domain and semantic knowledge
- Can use relationships among parameters
- Modeling can be based on requirements, not implementation
- The same parameter may appear in multiple characteristics, so it's harder to translate values to test cases

Steps 1 & 2 – Identifying Functionalities, Parameters and Characteristics

- A creative engineering step
- More characteristics means more tests
- Interface-based : Translate parameters to characteristics
- Candidates for characteristics :
  - Preconditions and postconditions
  - Relationships among variables
  - Relationship of variables with special values (zero, null, blank, …)
- Should not use program source – characteristics should be based on the input domain
  - Program source should be used with graph or *logic* criteria
- Better to have more characteristics with few blocks
  - Fewer mistakes and fewer tests

Step 3 : Modeling the Input Domain

- Partitioning characteristics into blocks and values is a very creative engineering step
- More blocks means more tests
- The partitioning often flows directly from the definition of characteristics and both steps are sometimes done together
  - Should evaluate them separately – sometimes fewer characteristics can be used with more blocks and vice versa
- Strategies for identifying values :
  - Include valid, invalid and special values
  - Sub-partition some blocks
  - Explore boundaries of domains
  - Include values that represent "normal use"
  - Try to balance the number of blocks in each characteristic
  - Check for completeness and disjointness

Example

- I would like to write a program which will determine if a triangle is valid and what type of triangle it is.
  - How would I determine if a side is valid?
  - How would I determine if a triangle is valid?
  - How would I determine what type of a triangle it is?

Interface-Based IDM – TriTyp

- TriTyp, from Chapter 3, had one testable function and three integer inputs

First Characterization of TriTyp's Inputs

Characteristic	b_1	b_2	b_3
q_1 = "Relation of Side 1 to 0"	greater than 0	equal to 0	less than 0
q_2 = "Relation of Side 2 to 0"	greater than 0	equal to 0	less than 0
q_3 = "Relation of Side 3 to 0"	greater than 0	equal to 0	less than 0

- A maximum of 3*3*3 = 27 tests
- Some triangles are valid, some are invalid
- Refining the characterization can lead to more tests ...