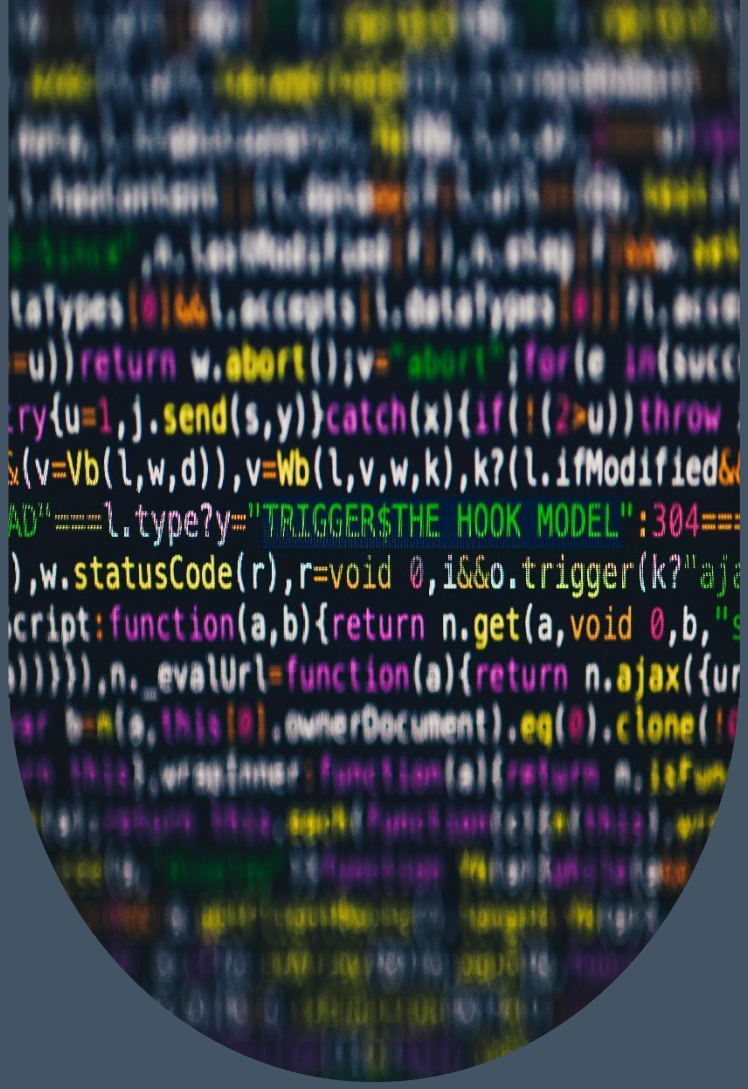
Input Space Partitioning (ISP)

Partially adopted from Introduction to Software Testing, Edition 2 , *Ammann & Offutt*



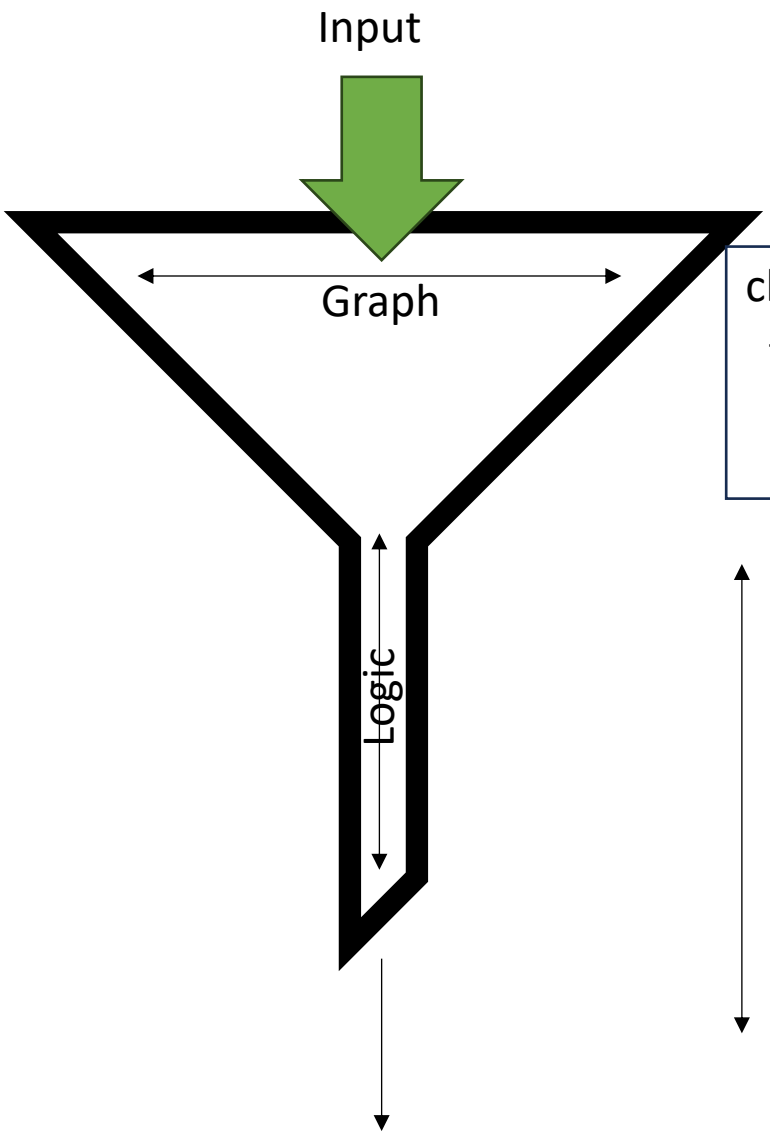
SETU
Ollscoil Teicneolaíochta an Oirdheiscirt
South East Technological University

setu.ie
INSPIRING FUTURES

From Paths and Predicates to Inputs

- So far:
- Graph-based testing: nodes, edges, paths
- Logic-based testing: predicates, clauses, ACC/CACC
- Today:
- Model the input space itself
- Design tests from the input domain

From Paths and Predicates to Inputs




```
class compute
 {
   int main ( )
     {
```

```
If ( c1 > c2 && c3 != 0 )
  {
    if (c3||C4 > 5)
      {
        do something

      }
  }
```

Why Study Input Space Partitioning ?

- Inputs often form large or infinite sets
- Exhaustive testing over inputs is infeasible
- We need:
- Structure over the input domain
- A way to choose a small set of representative inputs

Benefits of ISP

- Equally applicable at several levels of testing
  - Unit
  - Integration
  - System
- Easy to apply with no automation
- Can adjust the procedure to get more or fewer tests
- No implementation knowledge is needed
- Just the input space

Input Domains

- Input domain: all possible inputs to a program
- Most input domains are so large that they are effectively infinite
- *Input parameters* define the scope of the input domain
- Parameter values to a method
- Data from a file
- Global variables
- User inputs
- We partition input domains into regions (called *blocks*)
- Choose at least one value from each block

Input domain: Alphabetic letters

Partitioning characteristic: Case of letter

- Block 1: upper case
- Block 2: lower case

Key Definitions

- Input domain: set of all possible inputs
- Partition: blocks that are disjoint and cover the entire domain
- Block: values equivalent for testing for a given property
- Characteristic: property used to form blocks

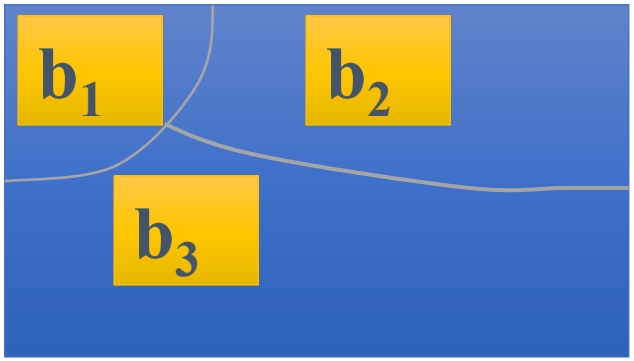
Partitioning Domains

- *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***)

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

  2. Together the blocks *cover* the domain *D* (***complete***)

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



What is a characteristic?

"A feature or quality belonging typically to a person, place, or thing and serving to identify it."

Input: people

concrete level

Characteristics: hair color, major

Blocks:
A=(red, black, brown, blonde, other)
B=(CS, SWE, CE, Math, IST, other)

abstract level

Abstraction:
A = [a1, a2, a3, a4, a5]
B = [b1, b2, b3, b4, b5, b6]

Examples

- Example characteristics
- Whether X is null
- Order of the list F (sorted, inverse sorted, arbitrary, …)
- Min separation of two aircraft
- Input device (DVD, CD, HMI, …)
- Hair color, height, major, age
- Partition characteristic into blocks
- Blocks may be single-value or a set of values
- Each value in a block should be equally useful for testing
- Each abstract test has one block from each characteristic

Choosing Partitions

- Defining partitions is not hard, but is easy to get wrong
- Consider the "*order of elements in list **F***"

b_1 = sorted in ascending order

b_2 = sorted in descending order

b_3 = arbitrary order

but … something's wrong …

Length 1 : [14]

The list will be in all three blocks …

That is, disjointness is not satisfied

Solution:

Two characteristics that address just one property

C1: List F sorted ascending

- c1.b1 = true
- c1.b2 = false

C2: List F sorted descending

- c2.b1 = true
- c2.b2 = false

Input Domain Modelling (IDM)

- IDM: abstract description of input space
- Expressed using:
- • Characteristics
- • Blocks for each characteristic
- Aim: make domain assumptions explicit

Modeling the input domain

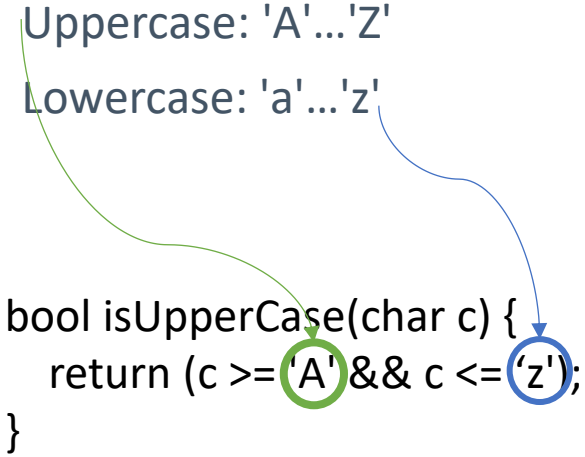
- Step 1 : Identify testable functions
- Step 2 : Find all inputs, parameters, & characteristics

} Concrete level

- Step 3 : Model the input domain } Move from imp level to design abstraction level
- 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

Example 1: Letters and Case

- Input domain **D**: alphabetic characters
- Characteristic: letter case
- Blocks:
- Uppercase: 'A'…'Z'
- Lowercase: 'a'…'z'



Example 2: Even Integer Function

- Function:
- bool isEven(int x) {
- return x % 2 == 0;
- }
- Characteristic ***C1***: parity of x
- Block: even integers
- Block: odd integers

Representative tests: x=0 (true), x=1 (false).

Combining Characteristics

- Same function isEven(int x)
- Add characteristic ***C2***: sign of x
- • *Negative*
- • *Zero*
- • *Positive*
- Now have parity (even/odd) and sign (− / 0 / +)

C1 : Odd, Even
C2: (-/0/+)

Test Requirements must contain (C1) * (C2) , so that:

- They do not overlap
- Together, they must cover the entire domain

That would mean all input combinations
(negative Odd, negative Even, 0 Odd, 0 Even, positive Odd, positive Even)

IDM in Action: Steps 1 & 2

Identify testable functions

Find inputs, parameters, characteristics

Steps 1 & 2—IDM

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

Parameters and Characteristics

Two parameters : ***list***, ***element***

Characteristics based on syntax :
- ***list*** is null (block1 = true, block2 = false)
- ***list*** is empty (block1 = true, block2 = false)

Characteristics based on behavior :
- number of occurrences of ***element*** in list
  - (0, 1, >1)
- ***element*** occurs ***first*** in list
  - (true, false)
- ***element*** occurs ***last*** in list
  - (true, false)

IDM in Action: Step 3

Model input domain

Partition characteristics into blocks

Choose values for blocks

Parameters and Characteristics

Two parameters : ***list***, ***element***

Characteristics based on syntax :
- ***list*** is null (block1 = true, block2 = false)
- ***list*** is empty (block1 = true, block2 = false)

Characteristics based on behavior :
- number of occurrences of ***element*** in list (0, 1, >1)
- ***element*** occurs ***first*** in list (true, false)
- ***element*** occurs ***last*** in list (true, false)

Characteristic q	Block b1	Block b2
Q1 = "list is null"	true	false
Q2 = "list is empty"	true	false
Q3 = "list contains elements"	true	true

This partition will yield 3 * 3 = 9 tests.
However, elements occurrence is not covered and may require further tests.

Refining IDM

Parameters and Characteristics

Two parameters : ***list***, ***element***

Characteristics based on syntax :
- ***list*** is null (block1 = true, block2 = false)
- ***list*** is empty (block1 = true, block2 = false)

Characteristics based on behavior :
- number of occurrences of ***element*** in list (0, 1, >1)
- ***element*** occurs ***first*** in list (true, false)
- ***element*** occurs ***last*** in list (true, false)

Characteristic q	Block b1	Block b2	Block b3
Q1 = "list is null"	true	false	0
Q2 = "list contains element at first position"	true	true	1
Q3 = "list contains elements with occurrence"	true	true	(0,1,>1)

This partition will yield 3 * 3 * 3 = 27 tests.
However, elements occurrence is not covered and may require further tests.

Refining IDM

Parameters and Characteristics

Two parameters : ***list***, ***element***

Characteristics based on syntax :
- ***list*** is null (block1 = true, block2 = false)
- ***list*** is empty (block1 = true, block2 = false)

Characteristics based on behavior :
- number of occurrences of ***element*** in list (0, 1, >1)
- ***element*** occurs ***first*** in list (true, false)
- ***element*** occurs ***last*** in list (true, false)

Characteristic q	Block b1	Block b2
Q1 = "list is null"	true	True
Q2 = "list contains element at first position"	true	1
Q3 = "list contains elements at last position"	true	>1

This partition will yield 3 * 3 = tests, but all scenarios covered

Refining IDM

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

Characteristic	b_1	b_2	b_3	b_4	b_5	b_6
A: size and contents	list=null	size=0	size=1	size>1 varied unsorted	size>1 varied sorted	size>1 all same
B: match	Element not found	Element found once	Element found more than once	--	--	--
Infeasible combinations: (Ab_1, Bb_2), (Ab_1, Bb_3), (Ab_2, Bb_2), (Ab_2, Bb_3), (Ab_3, Bb_3), (Ab_6,Bb_2)						

Element cannot be in a null list once (or more than once)

Element cannot be in a 0-element list once (or more than once)

Element cannot be in a 1-element list more than once

If a list has many of the same element, we can't find it just once

IDM hints

- More characteristics → more tests
- More blocks → more tests
- Do not use program source
- ***Design more characteristics with fewer blocks***
  - ❑ Fewer mistakes
  - ❑ Fewer tests
- Choose values strategically
- Valid, invalid, special values
- Explore boundaries
- Balance the number of blocks in the characteristics

Step 4 – Choosing combinations of values (6.2)

- After partitioning characteristics into blocks, testers design tests by combining blocks from different characteristics
- Consider an arbitrary test space that has 3 Characteristics (abstract): A, B, C
- Abstract blocks: ***A*** = [a1, a2, a3,a4]; ***B*** = [b1, b2]; ***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

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

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

All Combinations (ACoC) : Test with all combinations of blocks from all characteristics.

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 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

ISP criteria – each choice

- We should try at least one value from each block

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

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

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

ISP criteria – base choice (BCC)

- ECC is simple, but very few tests
- The base choice criterion recognizes :
- Some blocks are more important than others
- Using diverse combinations can strengthen testing
- Lets testers bring in domain knowledge of the program

Base Choice Coverage (BCC) : Choose a base choice block for each characteristic. Form a base test by using the base choice for each characteristic. Choose subsequent tests 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

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

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

Example

Input: students

Characteristics: Level, Mode, Major, Classification

Blocks:

A Level: (grad, undergrad)

B Mode: (full-time, part-time)

C Major: (cs, swe, other)

D Classification: (in-state, out-of-state)

Abstract IDM:

A = [a1, a2] C = [c1, c2, c3]

B = [b1, b2] D = [d1, d2]

In-class exercise

All combinations criterion (ACoC)

Consider this abstract IDM

4 Characteristics: A, B, C, D

Abstract blocks: A = [a1, a2]; B = [b1, b2];
C = [c1, c2, c3]; D = [d1, d2]

How many tests are needed to satisfy ACoC?

In-class exercise (*answer*)

All combinations criterion (ACoC)

4 Characteristics: A, B, C, D

Abstract blocks: A = [a1, a2]; B = [b1, b2];
C = [c1, c2, c3]; D = [d1, d2]

Number of tests: 2*2*3*2 = 24

a1 b1 c1 d1	a1 b2 c1 d1	a2 b1 c1 d1	a2 b2 c1 d1
a1 b1 c1 d2	a1 b2 c1 d2	a2 b1 c1 d2	a2 b2 c1 d2
a1 b1 c2 d1	a1 b2 c2 d1	a2 b1 c2 d1	a2 b2 c2 d1
a1 b1 c2 d2	a1 b2 c2 d2	a2 b1 c2 d2	a2 b2 c2 d2
a1 b1 c3 d1	a1 b2 c3 d1	a2 b1 c3 d1	a2 b2 c3 d1
a1 b1 c3 d2	a1 b2 c3 d2	a2 b1 c3 d2	a2 b2 c3 d2

In-class exercise

Each choice criterion (ECC)

Apply ECC to our previous example

4 Characteristics: A, B, C, D
Abstract blocks: A = [a1, a2]; B = [b1, b2];
C = [c1, c2, c3]; D = [d1, d2]

1. How many tests are needed for ECC?
2. Design the (abstract) tests

In-class exercise (*answer*)

Each choice criterion (ECC)

4 Characteristics: A, B, C, D

Abstract blocks: A = [a1, a2]; B = [b1, b2];
C = [c1, c2, c3]; D = [d1, d2]

Number of tests: max(2,2,3,2) = 3

a1 b1 c1 d1
a2 b2 c2 d2
a1 b1 c3 d1

In-class exercise

Base choice criterion (BCC)

Apply BCC to our previous example

4 Characteristics: A, B, C, D
Abstract blocks: A = [a1, a2]; B = [b1, b2];
C = [c1, c2, c3]; D = [d1, d2]

1. How many tests are needed for BCC?
2. Pick base values and write one base test
3. Design the remaining (abstract) tests

In-class exercise (*answer*)

Base choice criterion (BCC)

4 Characteristics: A, B, C, D

Abstract blocks: A = [a1, a2]; B = [b1, b2];
C = [c1, c2, c3]; D = [d1, d2]

Number of tests: 1(base)+1+1+2+1 = 6

Base	a1 b1 c1 d1
A	**a2** b1 c1 d1
B	a1 **b2** c1 d1
C	a1 b1 **c2** d1
C	a1 b1 **c3** d1
D	a1 b1 c1 **d2**

ISP criteria – multiple base choice

– We sometimes have more than one logical base choice

Multiple Base Choice Coverage (MBCC) : Choose at least one, and possibly more, base choice blocks for each characteristic. Form base tests 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:

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

For our example: Two base tests: a1, b1, c1, d1 a2, b2, c2, d2

Tests from a1, b1, c1, d1: a1, b1, c3, d1

Tests from a2, b2, c2, d2: a2, b2, c3, d2

MCBC – Example

- If M base tests and m_i base choices for each characteristic: $M + \sum_{i=1}^{Q} (M * (B_i - m_i))$

Characteristics	blocks		
A	a1	a2	a3
B	b1	b2	--
C	c1	c2	--

Multiple base choices

TR = { **(a1, b1, c1)**, Base choice #1
(a2, b1, c1), Variations on A
~~(**a3**, b1, c1)~~,
(a1, **b2**, c1), Variation on B
~~(a1, b1, **c2**)~~, Variation on C
(a3, b1, c2), Base choice #2
~~(**a1**, b1, c2)~~, Variations on A
(a2, b1, c2),
(a3, **b2**, c2), Variation on B
~~(a3, b1, **c1**)~~ } Variation on C

Substituting a3 in place of a1 is not necessary because a3 is also a base choice and will show up in a later TR

Comparing ECC, ACoC, and BCC

- ECC:
- • Each block appears at least once
- • Minimal, weak
- ACoC:
- • All block combinations
- • Strong, expensive
- BCC:
- • Structured around base test
- • Between ECC and ACoC

ISP in the Testing Framework

- ISP is a black-box technique
- Complementary to:
- • Graph-based coverage
- • Logic-based coverage
- Used to:
- • Design test inputs systematically
- • Expose domain-level faults

ISP Coverage Criteria Subsumption