



INPUT SPACE PARTITIONING

DR. ISAAC GRIFFITH

IDAHO STATE UNIVERSITY



"Just because you've counted all the trees doesn't mean you've seen the forest." – Anonymous

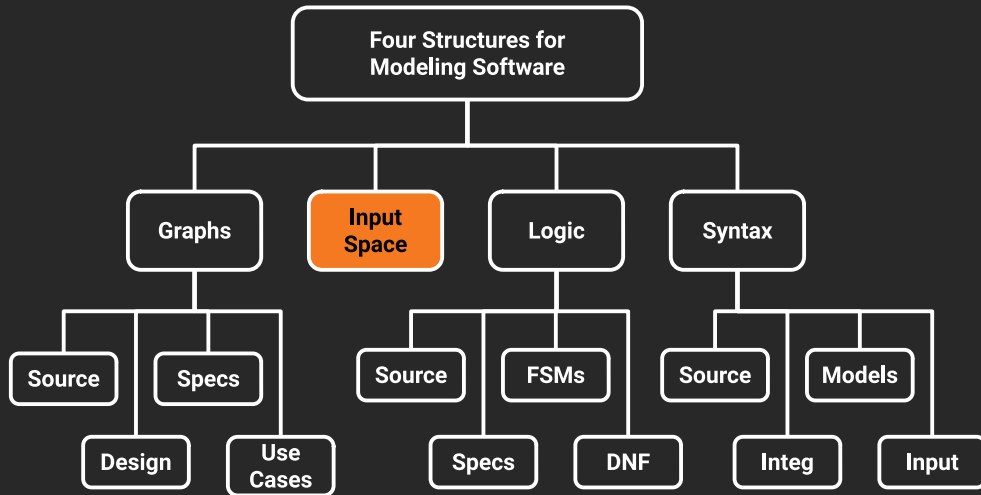
Outcomes



After today's lecture you will be able to:

- Understand the basics of ISP
- To partition an input domain
- Model an input domain
- Understand parameters and their characteristics
- Understand multiple ISP based criteria





Benefits of ISP



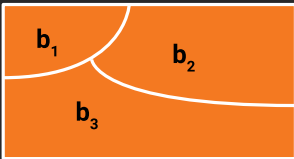
- Can be **equally applied** at several levels of testing
 - Unit
 - Integration
 - System
- Relatively easy to apply with **no automation**
- Easy to **adjust** the procedure to get more or fewer tests
- No **implementation knowledge** is needed
 - Just the input space

- The **input domain** for 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
- Input domains are **partitioned into regions** (blocks)
- At least **one value** is chosen from each block

Partitioning Domains



- Domain D
- Partition Scheme q of D
- The partition q defines a sets of blocks, $B_q = b_1, b_2, \dots, b_q$
- The partition must satisfy two **properties**:
 1. **Disjoint** (no overlap): $b_i \cap b_j = \emptyset, \forall i \neq j, b_i, b_j \in B_q$
 2. **Coverage** (all blocks cover the domain D): $\bigcup_{b \in B_q} b = D$



In Class Exercise



Design a partitioning for all integers. That is, partition integers into blocks such that each block seems to be equivalent in terms of testing.

Make sure your partition is valid:

1. Pairwise disjoint
2. Complete

Using Partitions - Assumptions



- Choose a **value** from each block
- 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**
 - Whether input X is null
 - Order of the input list F (sorted, inverse sorted, arbitrary, ...)
 - Min separation of two aircraft
 - Input device (DVD, CD, VCR, computer, ...)
 - Hair color, height, major, age

Choosing Partitions



- Choosing (or defining) **partitions** seems easy, but is easy to get wrong
- Consider the characteristic “**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 fishy

Length 1 : [14]

The list will be in all three blocks

That is, disjointness is not satisfied

Solution:

Each characteristic should address just one property

- C_1 : List F sorted ascending
 - $c_1.b_1 = true$
 - $c_1.b_2 = false$
- C_2 : List F sorted descending
 - $c_2.b_1 = false$
 - $c_2.b_2 = true$

Modeling the Input Domain



- **Step 1:** Identify testable **functions**
- **Step 2:** Find all **inputs & parameters**
 - Move from implementation level to design abstraction level
- **Step 3:** Model the **input domain** -> Produce an *Input Domain Model (IDM)*
 - Entirely at the design abstraction level
- **Step 4:** Apply a test **criterion** to choose **combinations** of values
 - Entirely at the design abstraction level
- **Step 5:** Refine combinations of blocks into **test inputs**
 - Back to the implementation abstraction level

In Class Exercise



- Pick one of the programs from Chapter 1 (findLast, numZero, etc.)
- Create an IDM for your program
- Take 10 minutes

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** (list is not empty)
 - **Relationships** to constants and among variables ($x > 0, x \neq y$)
 - Based on software **behavior** (element is in the set)
- 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

Example IDM Based on Syntax



- Consider method `triang()` from class `TriangleType` in `Triangle.java` on Moodle

```
public enum Triangle { Scalene, Isosceles, Equilateral, Invalid }  
public static Triangle triang(int Side1, int Side2, int Side3)  
// Side1, Side2, and Side3 represent the lengths of the sides of a triangle  
//Returns the appropriate enum value
```

The IDM for each parameter is identical

Reasonable characteristic: Relation of side with zero

Example IDM Based on Behavior



- Again, consider method `triang()` from class `TriangleType`:
 - The three parameters represent a `triangle`
 - The IDM can combine all parameters
 - Reasonable characteristic: **type of triangle**

In Class Exercise



- **Identify functionalities, parameters, and characteristics for** `findElement()`

```
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
```



```
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)

Step 3



Modeling the Input Domain

- Partitioning characteristics into blocks and values is a very **creative engineering** step
- **More blocks** means more tests
- Partitioning often flows directly from the definition of **characteristics** and both steps are done together
 - Should **evaluate** them separately – sometimes fewer characteristics can be used with more blocks and vice versa

Step 3



Modeling the Input Domain, cont'd

- **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**

- `triang()` has one testable function and three integer inputs

First Characterization of TriType's Inputs

Characteristic	b_1	b_2	b_3
q_1 = "Relation of Side 1 to 0"	> 0	$= 0$	< 0
q_2 = "Relation of Side 2 to 0"	> 0	$= 0$	< 0
q_3 = "Relation of Side 3 to 0"	> 0	$= 0$	< 0

For Next Time



Idaho State
University

Computer
Science

- Review the Reading
- Review this Lecture
- Come to Class





Are there any questions?