CS 5154: Software Testing

Introduction to Input Space Partitioning

Owolabi Legunsen

CS5154 is organized into six themes

1. How to automate the execution of tests?
2. How to design and write high-quality tests?
3. How to measure the quality of tests?
4. How to automate the generation of tests?
5. How to reduce the costs of running existing tests?
6. How to deal with bugs that tests reveal?

How many inputs can we test this method on?

```
/** Compute the arithmetic mean of A, B, and C.
 *  Assume a 32-bit computer
**/
private static double computeAverage (int A, int B, int C) {
   …
}
```

Total no. of possible inputs = |values of A| x |values of B| x |values of C|
= 2^32 x 2^32 x 2^32
= 79.23 x 10^{27}
≅ 80 Octillion

How many inputs should we test this method on?

```
/** Compute the arithmetic mean of A, B, and C.
 *  Assume a 32-bit computer
**/
private static double computeAverage (int A, int B, int C) {
   …
}
```

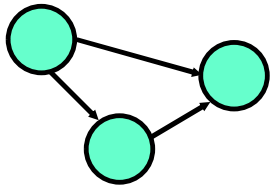
Let's see how model-driven test design can help

Recall that we will look at four models in this course

Input Domains

A: {0, 1, >1}
B: {600, 700, 800}
C: {cs, ece, is, sds}

Graphs



Logic Expressions

(not X or not Y) and A and B

Syntax

```
if (x > y)
    z = x - y;
else
   z = 2 * x;
```

ISP: choose inputs from the input domain

- You have been choosing inputs for programs all your career
- ISP helps you
  - choose those inputs more systematically
  - check that inputs come from different parts of the input domain
  - know when to stop choosing

3 key concepts in ISP

1. **Input Domain**: the set of all possible inputs to a program

2. **Input Parameters** that a program takes
   - e.g., method arguments, user inputs, file/database data, global variables
   - parameters define the scope of the input domain

3. **Input Domain Modeling:** process of reasoning about the input domain

The core activities in Input Domain Modeling

1. Partition the input domain into regions called **blocks**

2. Choose at least one value for each block

3. Combine values across various parameters

Wait… that's it?!

Partitioning the input domain into blocks

- Decide on characteristics of your input domain to partitioning on
- Assumption: values in each block are equally useful for testing
- Example:

Program: `void foo(String char)` // "char" is a letter

Input domain: Alphabetic letters

Partitioning characteristic: Case of letter

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

Your turn: examples on partitioning

- Program: `void foo(String char) // “char” is a letter`
- Input domain: Alphabetic letters
- Partitioning characteristic:
    - Block 1:
    - Block 2:
    - Block 3:

Your turn: examples on partitioning (2)

- Program: `void foo(String char) // “char” is a letter`
- Input domain: Alphabetic letters
- Partitioning characteristic:
    - Block 1:
    - Block 2:
    - Block 3:

Your turn: examples on partitioning (3)

- Program: `void bar(List<String> l)`
- Input domain:
- Partitioning characteristic:
    - Block 1:
    - Block 2:
    - Block 3:

Your turn: examples on partitioning (4)

- Program: `void baz(List<String> filenames)`
- Input domain:
- Partitioning characteristic:
    - Block 1:
    - Block 2:
    - Block 3:

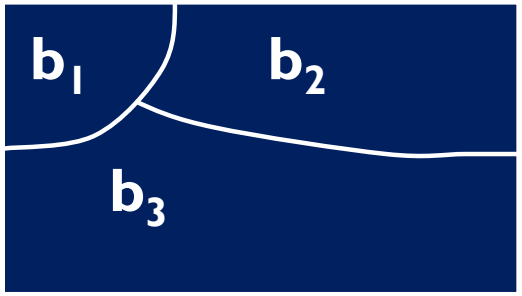
How to know that partitioning is “correct”?

- *Let the input domain be D*
- *Characteristic q* partitions *D* into *set of blocks*, $B_q = \{b_1, b_2, \ldots, b_Q\}$
- Each partition must satisfy two properties :
  1. Blocks must be *pairwise disjoint* (no overlap)

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

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

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



Partitioning is simple but easy to do wrong

- Consider the characteristic "*order of elements in list F*"

b_1 = sorted in ascending order

b_2 = sort

b_3 = arbitrary order

Design blocks for that characteristic

but … something's fishy …

What if the list is of length 0 or 1?

Can you find the problem?

The list … locks

That is, disjointness is not satisfied

One solution:

Two characteristics that each addres

Can you think of a solution?

C1: List …

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

C2: List F sorted descending

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

Your turn: work with your neighbor in pairs

Design a partitioning for the input domain: 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, and (2) Complete

Discuss with a nearby pair

- What characteristics did you all come up with?
- Are the blocks pairwise disjoint and complete?
- Work together to fix any problems that you find with the partitioning

Characteristics, blocks, problems that you discussed?

Characteristics, blocks, problems that you discussed?

How to know that partitioning is “complete”?

- Partitioning is a creative step
- Spend some time brainstorming
- We will give some hints that may help

Questions so far?

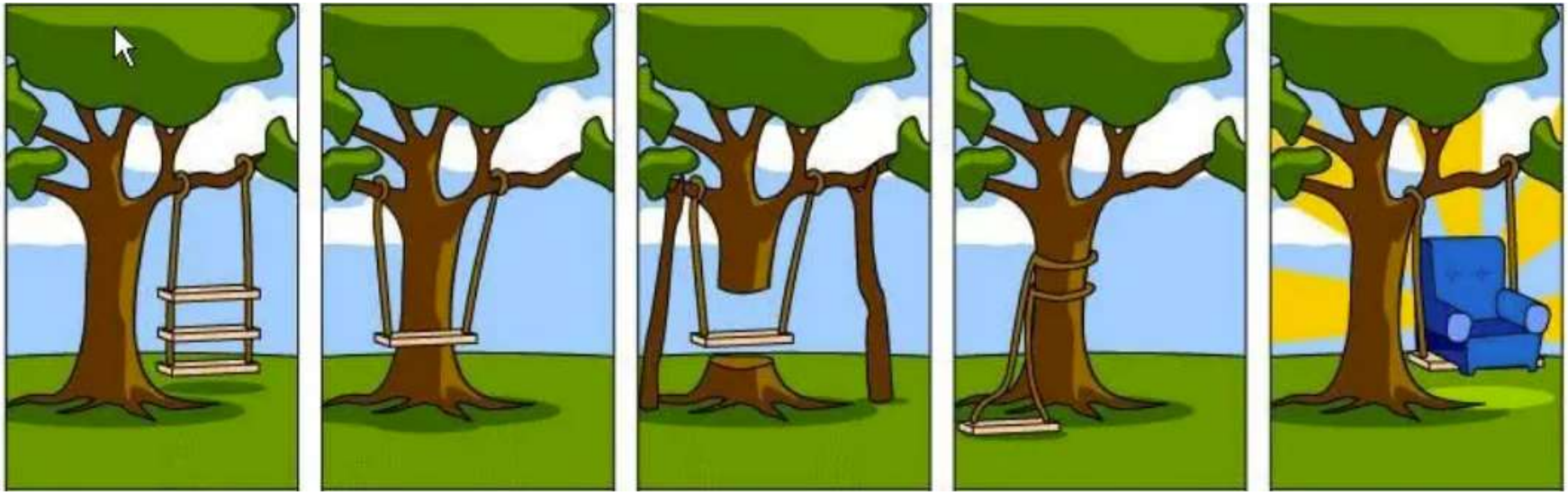
?

Let’s reflect on what we learned

- Partitioning is a creative design activity
  - many partitions can be created for the same input domain
- If done carelessly, partitions may not be complete or pairwise disjoint
- Consider different alternatives and the trade-offs that they induce

ISP is a design activity ☺

- Multiple partitioning schemes may exist for the same code



- Considering cost/benefit tradeoffs in designs is an essential part of SE

Image Credits: https://blogs.perficient.com/2011/07/22/how-to-build-a-tire-swing-a-case-for-agile-development/

Next..

(1) How is any of these systematic?

(2) Why did you say that ISP is a way to do model-driven test design?

Weekend reading?

Empir Software Eng (2014) 19:558–581
DOI 10.1007/s10664-012-9229-5

An industrial study of applying input space partitioning to test financial calculation engines

Jeff Offutt · Chandra Alluri

Published online: 23 September 2012

Editor: James Miller