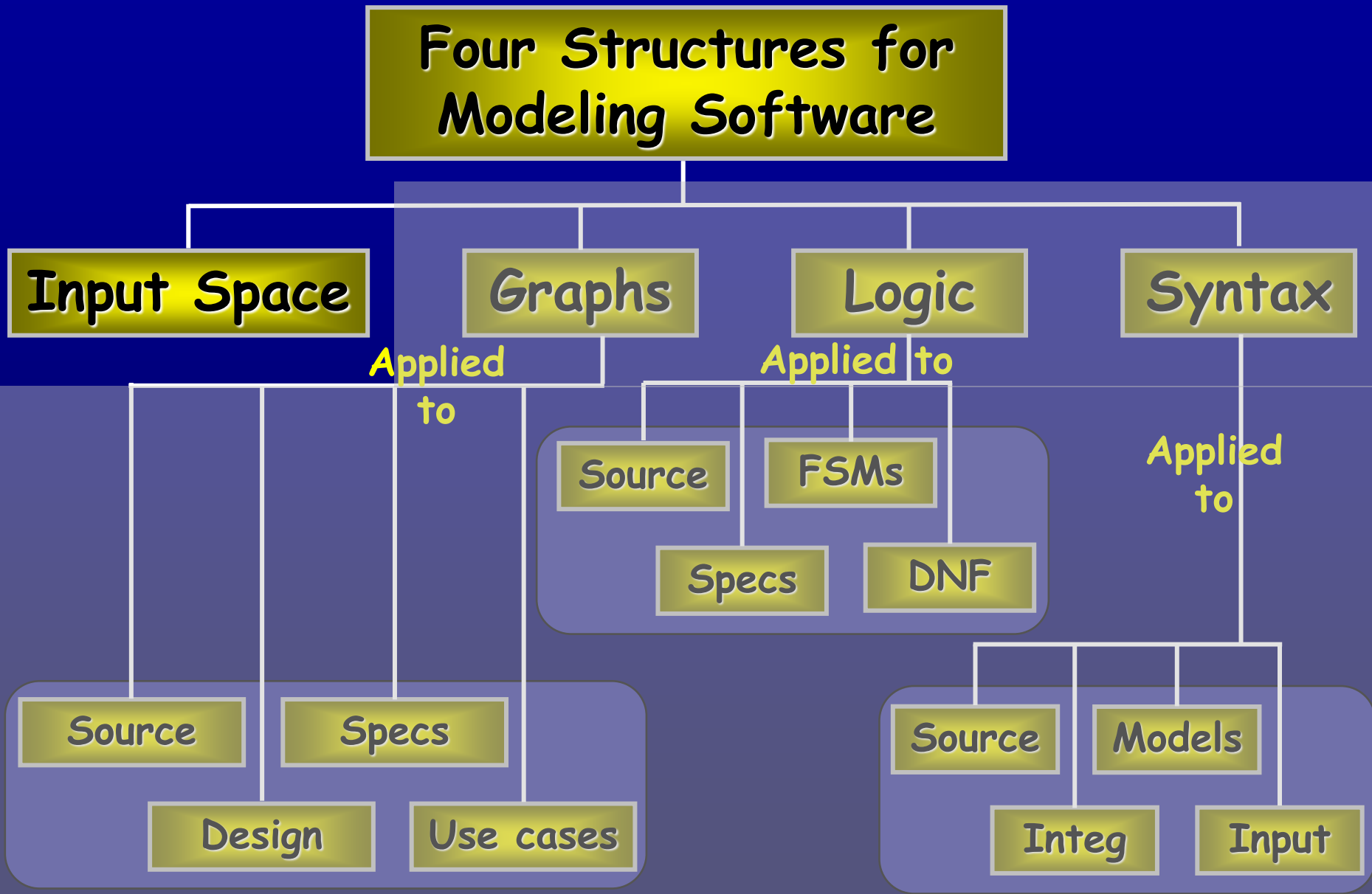
Introduction to Software Testing
Chapter 6
Input Space Partition Testing

Paul Ammann & Jeff Offutt

http://www.cs.gmu.edu/~offutt/softwaretest/

Ch. 6 : Input Space Coverage



Input Domains

- 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 Domains

- *Input parameters* define the scope of the input domain
  - Parameters to a method **(in unit testing)**
  - objects representing current state **(in class or integration testing)**
  - User level inputs **(in system testing)**
  - Depends on what kind of software artifact is being analyzed

- Input domains are partitioned into regions (blocks)
  - At least one value is chosen from each block

Partitioning Domains

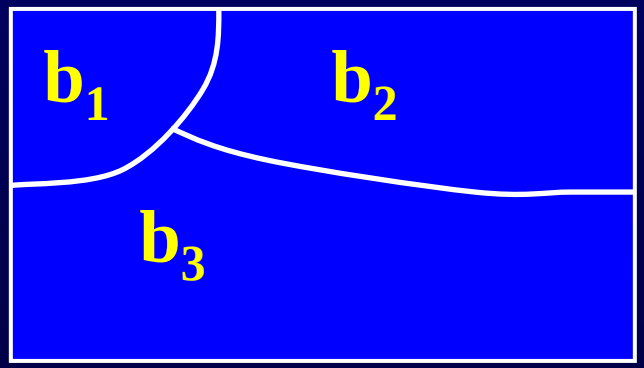
- *Domain D*
- *Partition scheme q* of *D*
- The partition *q* defines a *set of blocks*, $Bq = b_1, b_2, \ldots, b_Q$
- The partition must satisfy two properties :

1. Blocks must be *pairwise disjoint* (no overlap)

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

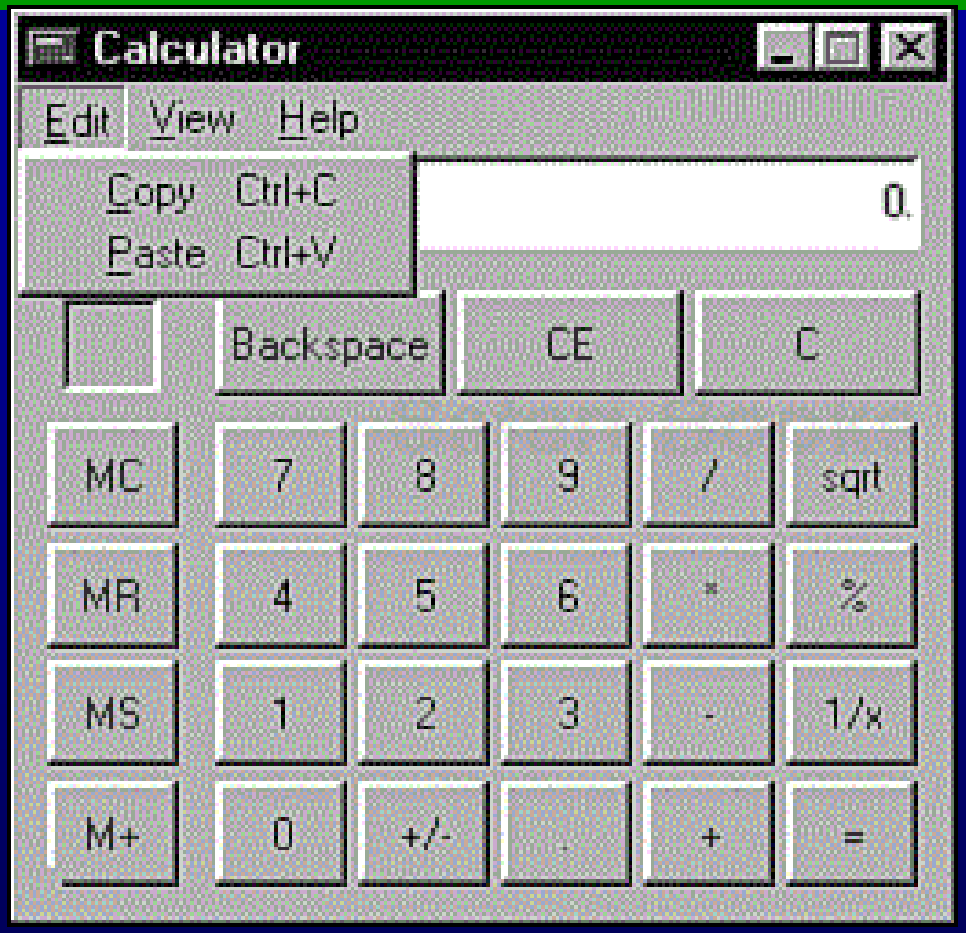
2. Together the blocks *cover* the domain *D* (complete)

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



Calculator Equivalence Partitioning

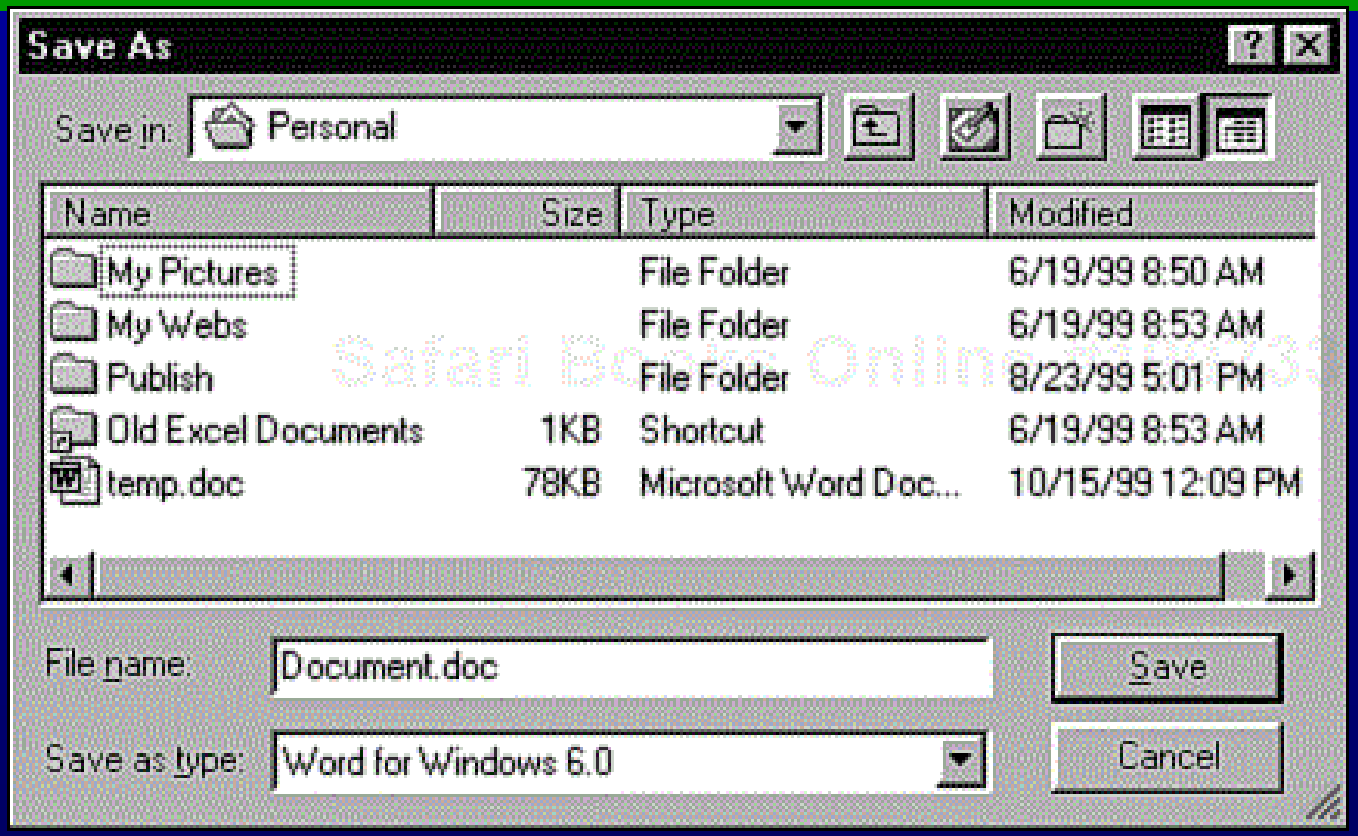
- The Edit menu shows Copy and Paste commands.
- There are 5 different ways to perform the copy command.
  - Click Copy menu item
  - Type C
  - Type c
  - Press Ctrl + c
  - Press Ctrl + Shift + c



- **What equivalence partitions do we have?**

(c) Copyright Soha Makady 2016

File-Save As Equivalence Partitioning



- A Windows filename can contain any character except \ / : * ? " < > and |
- A windows filename can have from 1 to 255 characters

(c) Copyright Soha Makady 2016

- A Windows filename can contain any character except \ / : * ? “ < > and |
- A windows filename can have from 1 to 255 characters

Benefits of ISP

- Can be equally applied at several levels of testing
  - Unit
  - Integration
  - System
- Relatively easy to apply
- No implementation knowledge is needed
  - Just the input space

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
  - Each partition is usually based on some *characteristic C* of the program, the program's inputs, or the program's environment
  - Choose tests by combining values from characteristics

Using Partitions – Assumptions

- Example Characteristics
  - Input X is null
  - Order of the input list F (sorted, inverse sorted, arbitrary, …)
  - Input device (DVD, CD, VCR, computer, …)

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 = sor… …der

b_3 = arbitrary order

Design blocks for that characteristic

but … something's fishy …

What if the list is of length 1?

Can you find the problem?

The list … …locks

That is, disjointness is not satisfied

Solution:

Each characteristic should address just one property

Can you think of a solution?

C1: … …ng
- c1.b1 = true
- c1.b2 = false

C2: List F sorted descending
- c2.b1 = true
- c2.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
- Different alternatives should be considered

Choosing Blocks and Values

- **Valid vs. invalid values:** Every partition must allow all values, whether valid or invalid. (This is simply a restatement of the completeness property.)
- **Sub-partition:** A range of valid values can often be partitioned into sub-partitions, such that each sub-partition exercises a somewhat different part of the functionality.
- **Boundaries:** Values at or close to boundaries often cause problems.
- **Normal use** (happy path)
- **Enumerated types:** A partition where blocks are a discrete, enumerated set often makes sense. The triangle example uses this approach.

Choosing Blocks and Values

- The input condition is a range of values (e.g., the item count can be from 1 to 999)
  - One valid equivalence class (1 <= value <= 999)
  - Two invalid equivalence classes: Value <1 and value > 999
- The input condition is a number of values (e.g., a car can have from one to six owners)
  - One valid equivalence class
  - Two invalid equivalence classes: no owners and more than 6 owners

Choosing Blocks and Values

- The input condition is a set of input values, where each value receives different handling (e.g., the vehicle can be TRUCK, BUS, MOTORCYLE).
  - One valid equivalence class for EACH value
  - One invalid equivalence class (e.g., TRAILER)
- The input specifies a must-be situation (e.g., first character of an identifier must be a letter)
  - One valid equivalence class: begins with a character
  - One invalid equivalence class: doesn't begin with a letter

Required Reading

- Chapter 6 from from the course's textbook: "Introduction to Software Testing", Cambridge University Press. P. Amman and J. Offutt, Second Edition, 2017.