CS3300 Introduction to Software Engineering

Lecture 16: Black-Box Testing

Nimisha Roy ▸ nroy9@gatech.edu

Slides adapted from Alessandro Orso

Black- Box Testing

Advantages

- Focus on the domain
- No need for the code
  - Early test design
  - Prevents the highly occurring scenario of no-time-for-testing
- Catches logic defects
- Applicable at all granularity levels

Slide adapted from Alessandro Orso

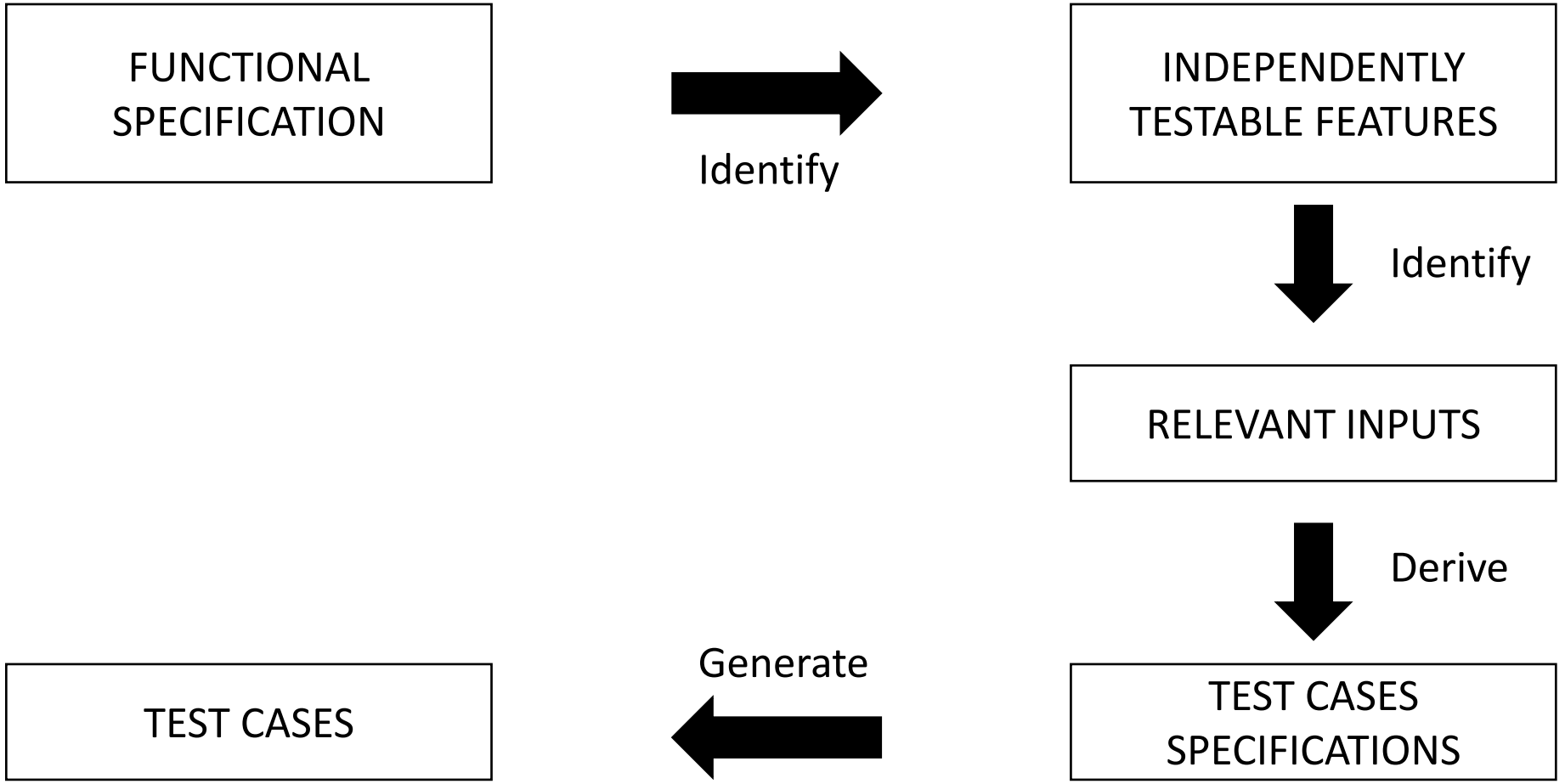
From Specifications to Test Cases

FUNCTIONAL SPECIFICATION

TEST CASES

Slide adapted from Alessandro Orso

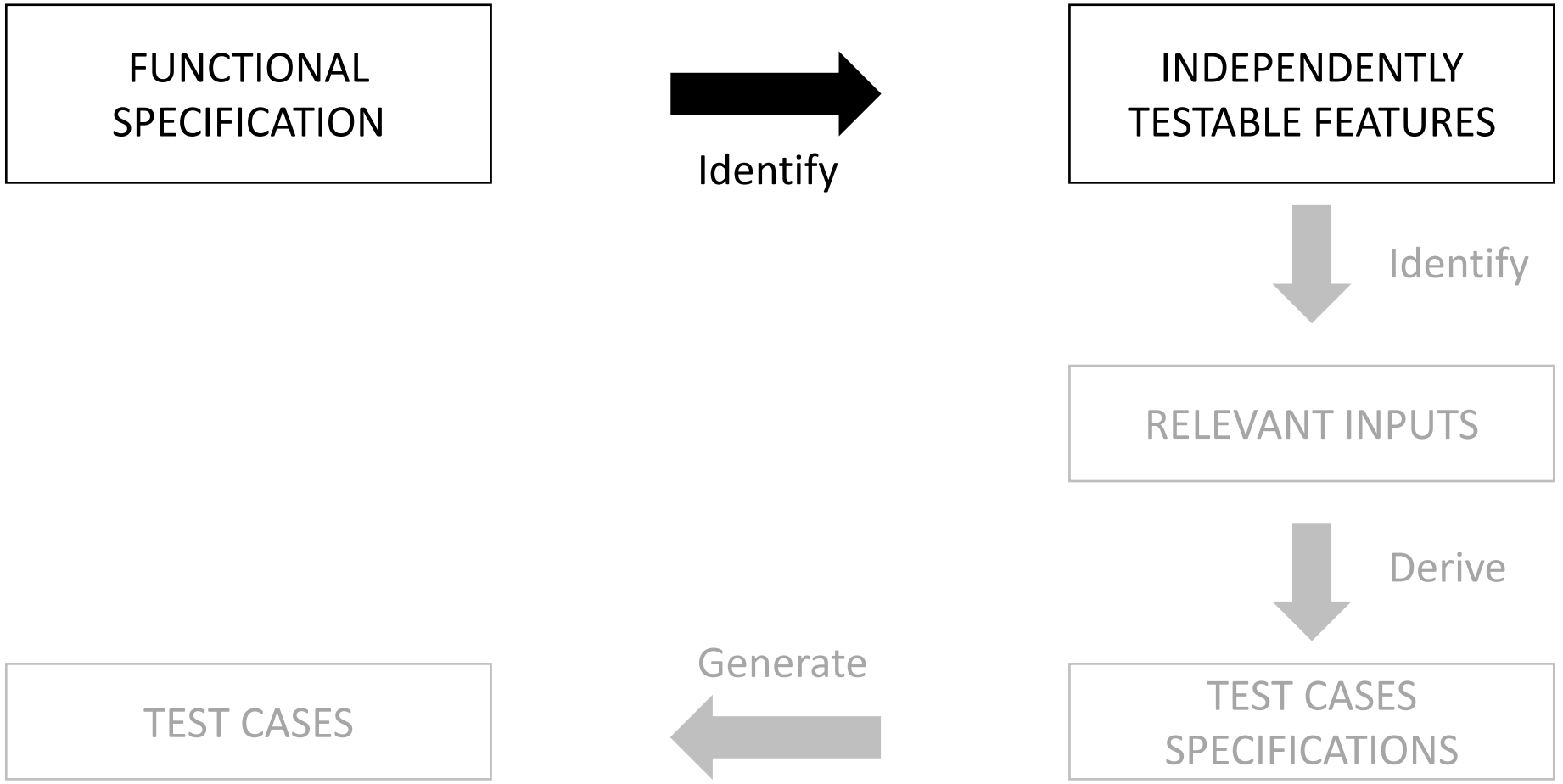
A systematic Functional-Testing Approach



Decoupling; Automated Sub-tasks; Monitor testing process

Slide adapted from Alessandro Orso

A systematic Functional-Testing Approach



Slide adapted from Alessandro Orso

Identifying Testable Features

printSum (int a, int b)

How many independently testable features do we have here?

[✓] 1

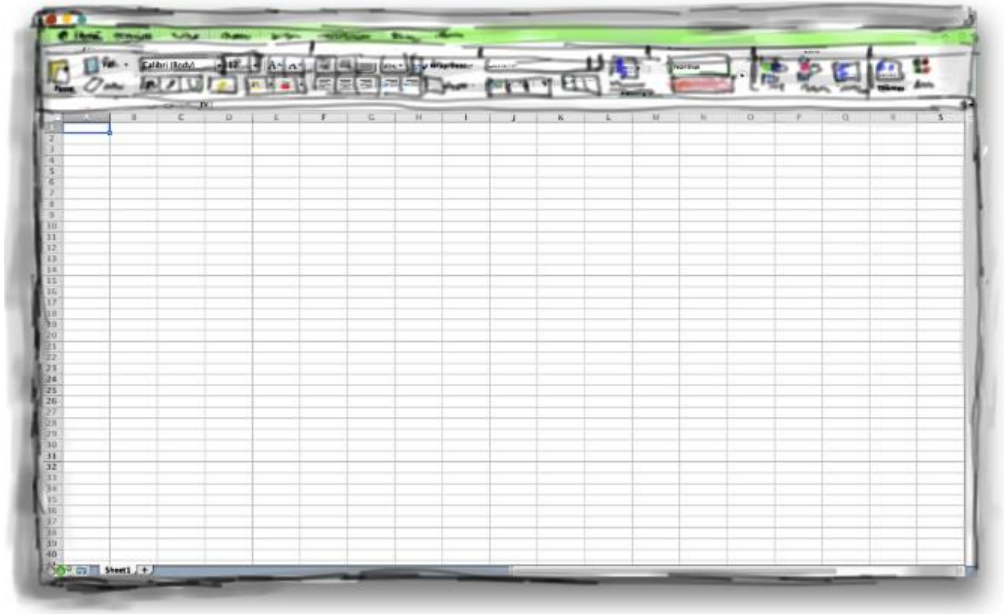
[ ] 2

[ ] 3

[ ] 4

Identifying Testable Features

Identify 3 possible independently testable features for a spreadsheet

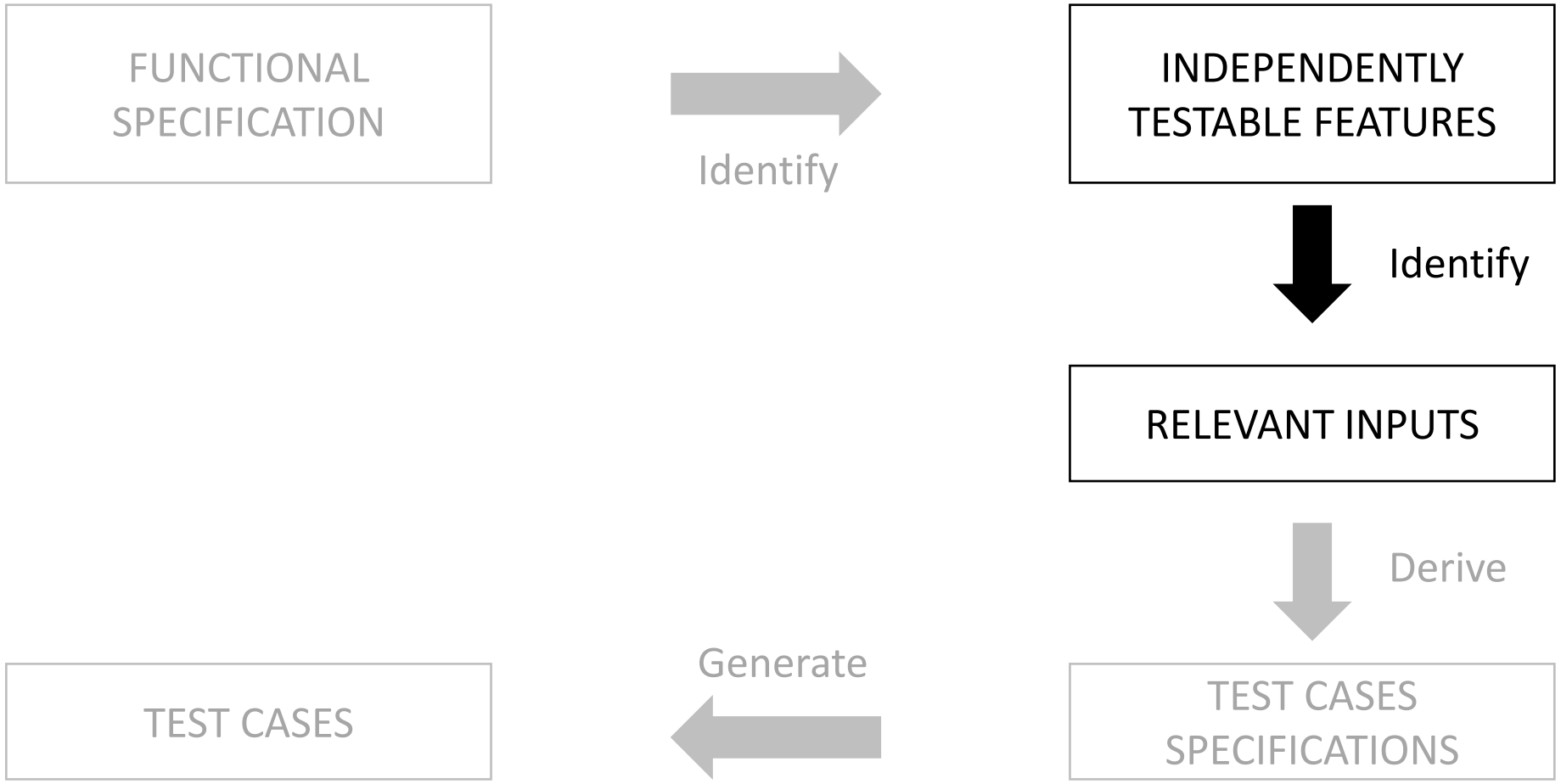


[Statistical Functions]

[Cell Merging]

[Chart creation]

A systematic Functional-Testing Approach



Slide adapted from Alessandro Orso

Test Data Selection

Input Domain D

Software

Output Domain O

How to select meaningful set of inputs and corresponding outputs?

Powerful machines, why not exhaustive search?

Straw-Man Idea: Exhaustive Testing!

How long would it take to exhaustively test the function printSum(int a, int b)?

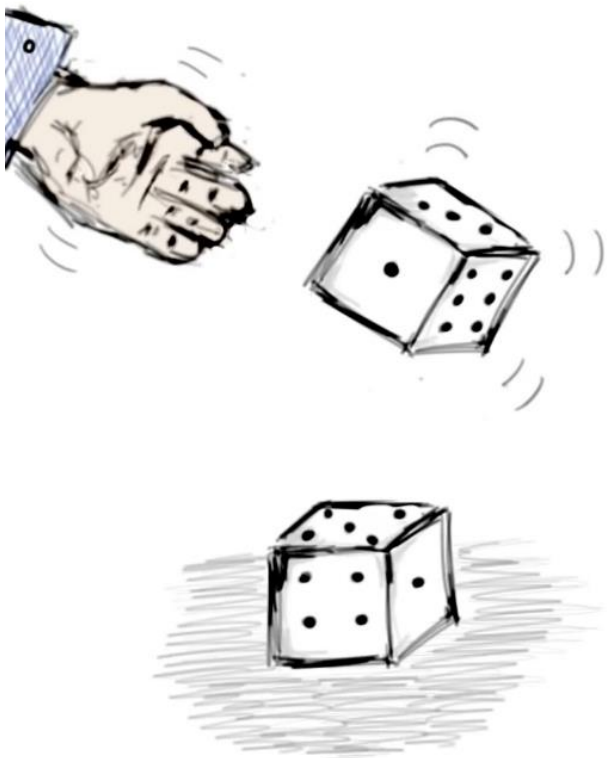
$2^{32} * 2^{32} = 2^{64} \sim= 10^{19}$ tests

1 test per nanosecond

10^9 tests per second

10^{10} seconds overall ~ 600 years

Random Testing



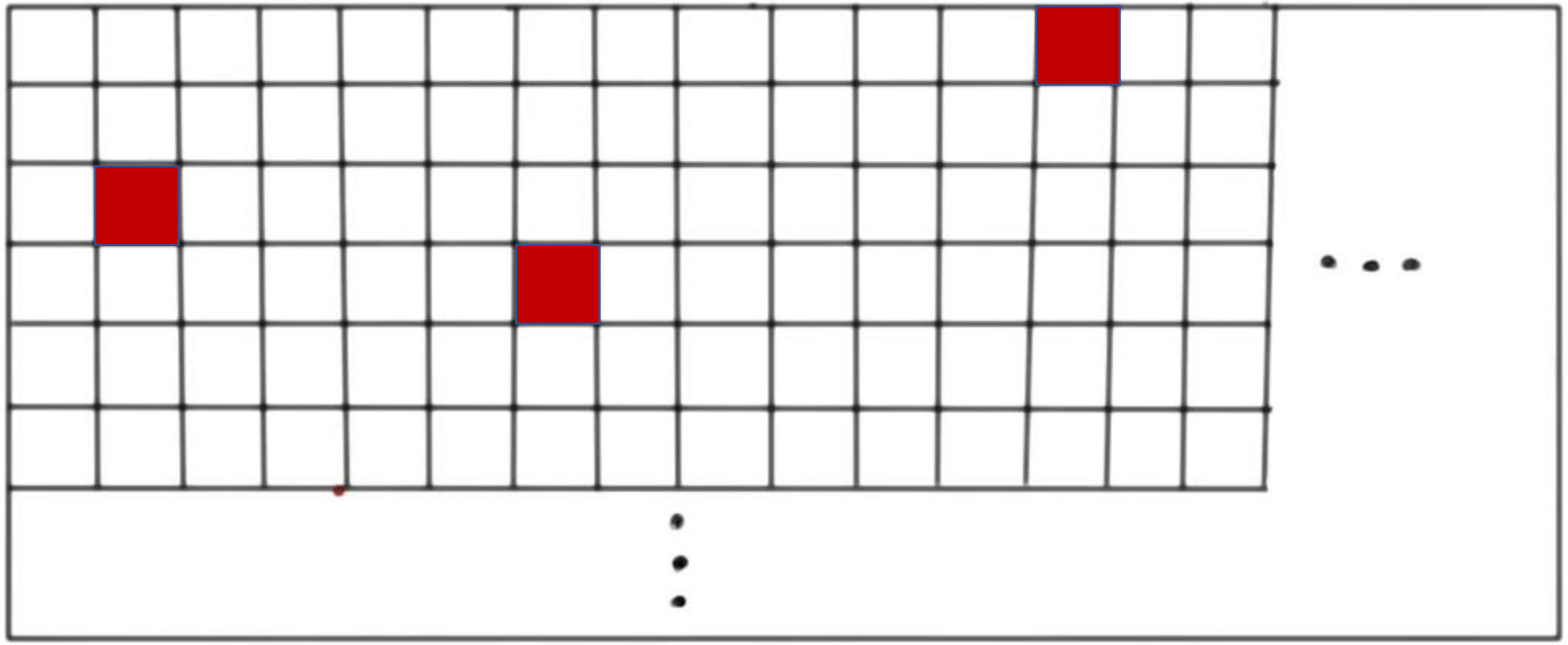
Advantages

- Pick inputs uniformly
- All inputs considered equal
- No designer bias (developer may develop code based on an assumption, test cases may also be biased)

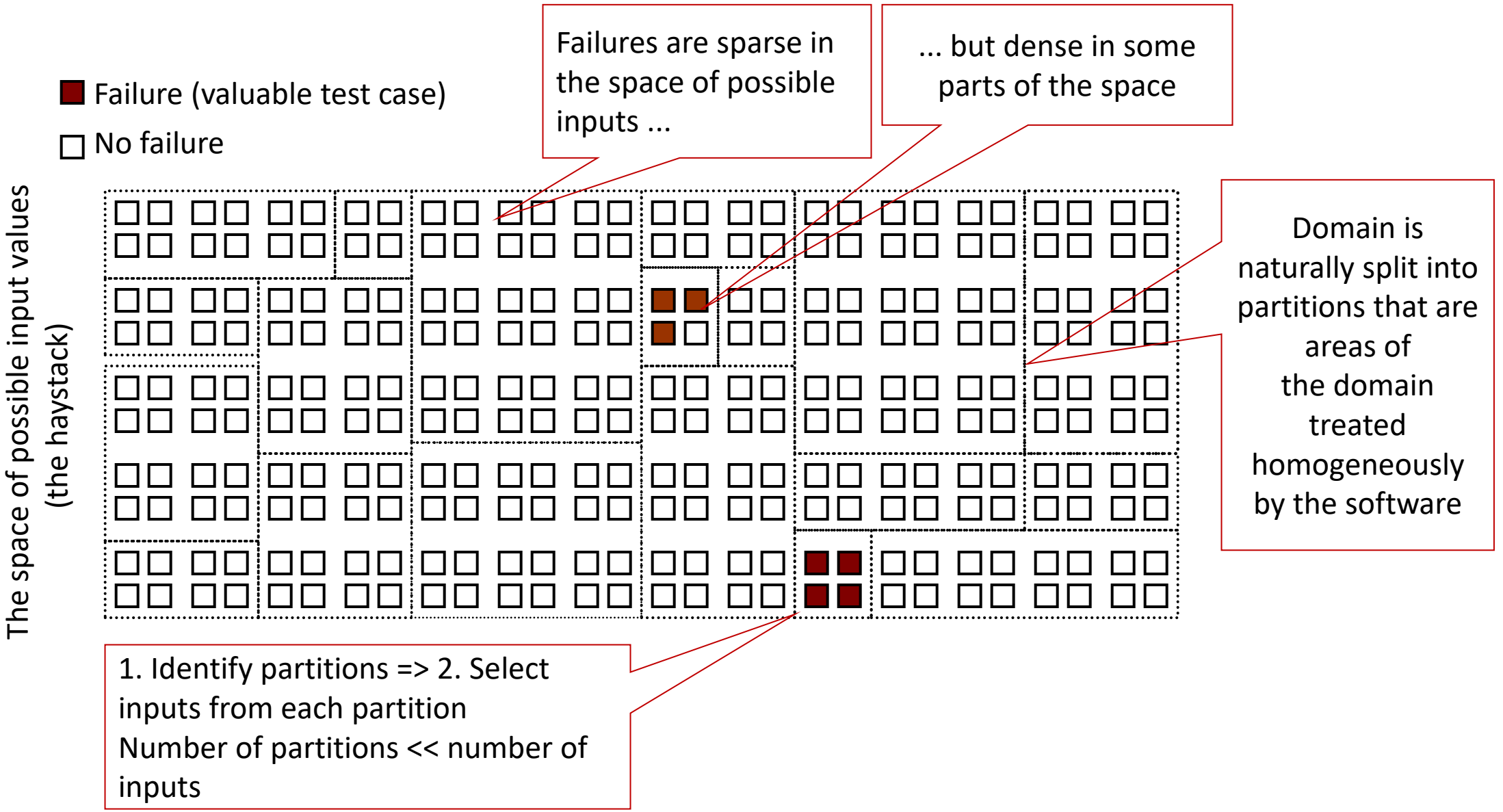
So why not random?

Same as finding many needles in a haystack

So why not random?



Systematic Partition Testing



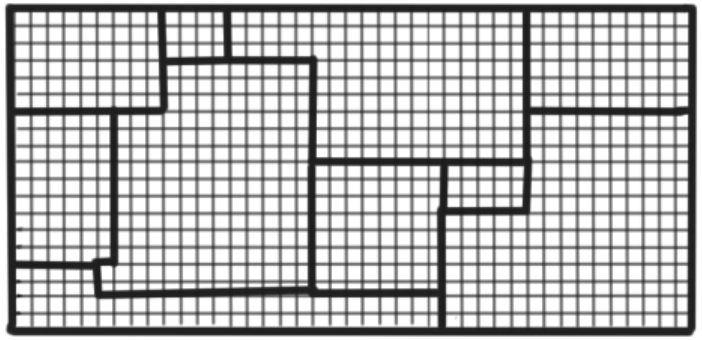
Example

split (string Str, int Size)

1. Identify partitions:

- Size < 0 (Designer bias might let you not pick this partition)
- Size = 0
- Size > 0
- Str with length < Size
- Str with length in [Size,Size*2]
- Str with length > Size*2
- …

Boundary Values



2. Select **interesting** Inputs from each partition

Basic Idea: Errors tend to occur at the boundary of a sub-domain

=> Select inputs at these boundaries

Example

split (string Str, int Size)

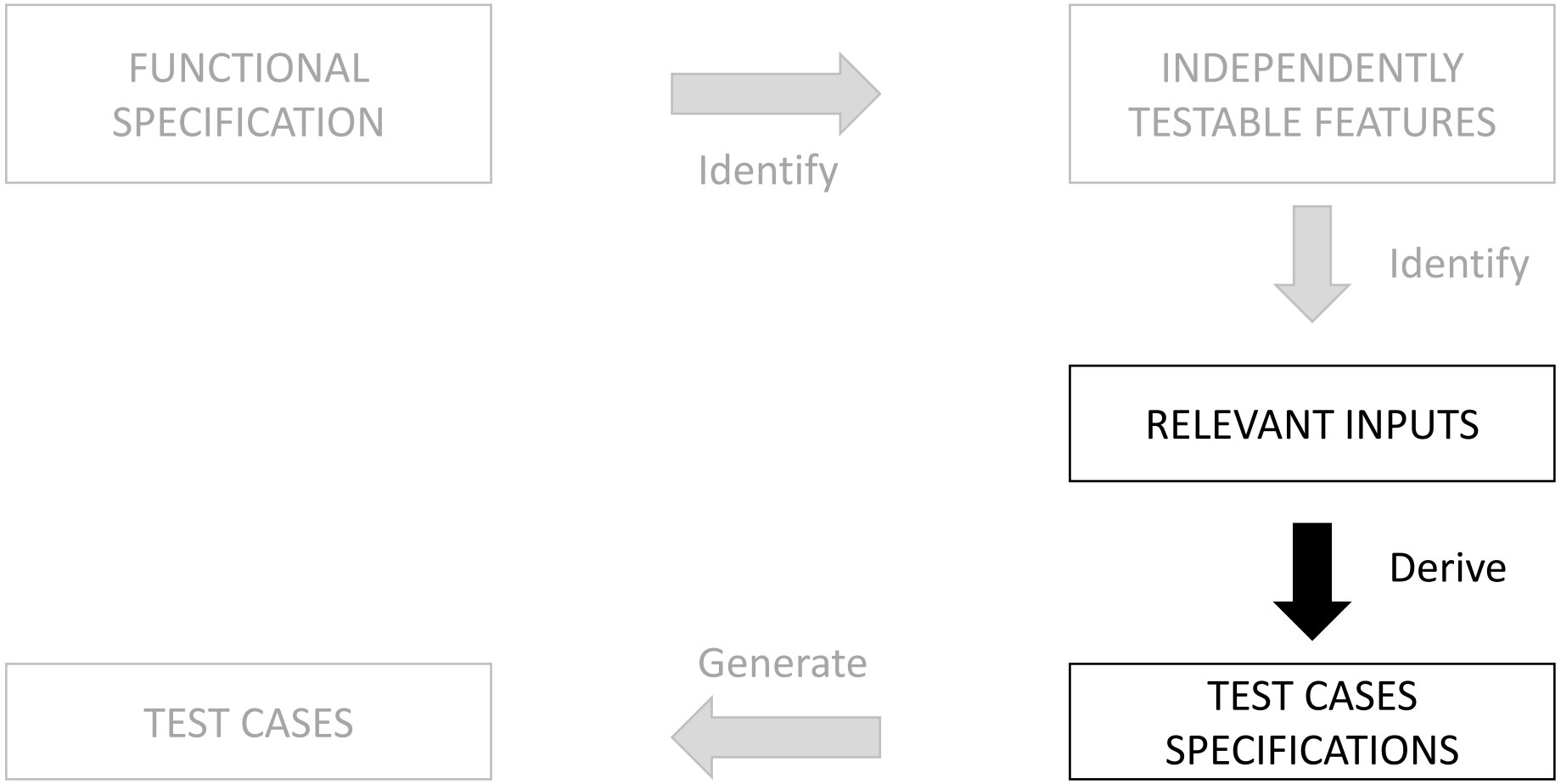
Some possible partitions:

- Size < 0
- Size = 0
- Size > 0
- Str with length < Size
- Str with length in [Size, Size*2]
- Str with length > Size*2

Some possible inputs:

- Size = -1
- Size = 1
- Size = MAXINT
- Str with length = Size- 1
- Str with length = Size
- …

A systematic Functional-Testing Approach



Example

split (string Str, int Size)

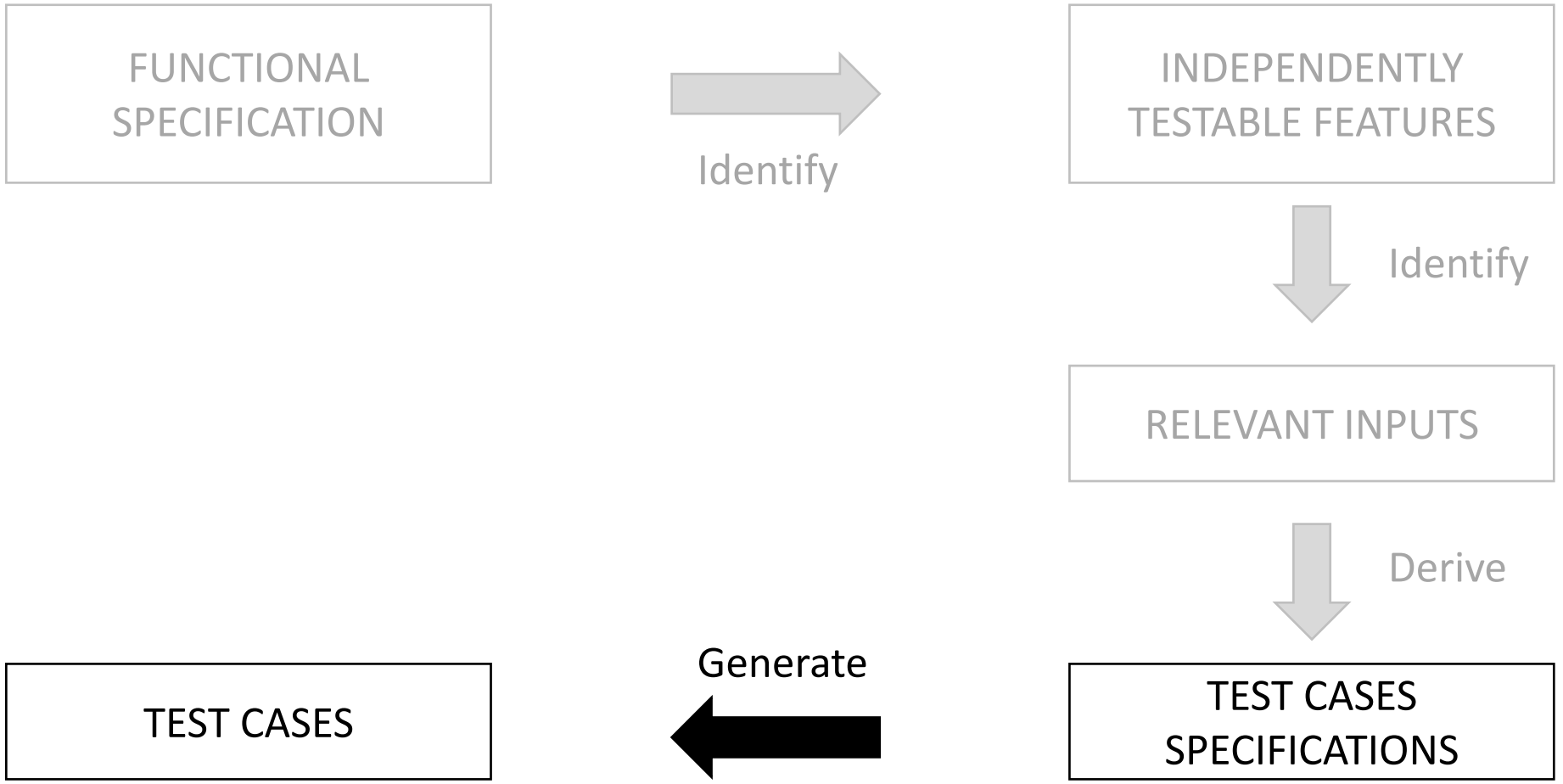
Some possible inputs:

- Size = -1 — Str with length = Size- 1
- Size = 1 — Str with length = Size
- Size = MAXINT — …

Test Case Specifications: (combine input values)

- ~~Size = -1, Str with length = -2~~
- ~~Size = -1, Str with length = -1~~
- Size = 1, Str with length = 0
- Size = 1, Str with length = 1
- …

A systematic Functional-Testing Approach



Slide adapted from Alessandro Orso

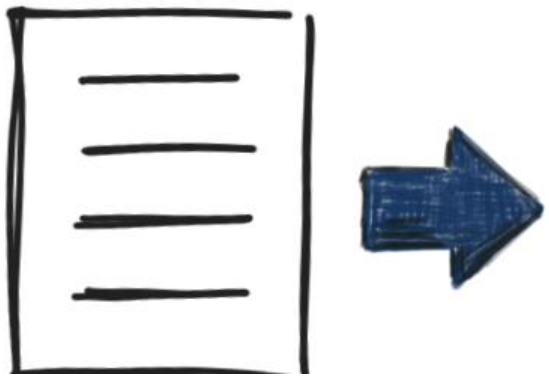
A Specific Functional Testing Black-Box Approach
The Category-Partition Method

[Ostrand & Balcer, CACM, June 1988]

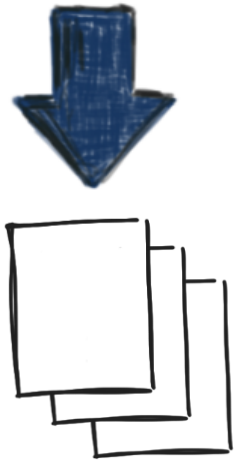
Specification

Test Cases

The Category-Partition Method



1. Identify independently testable features
2. Identify Categories
3. Partition Categories into choices
4. Identify constraints among choices
5. Produce/Evaluate test case specifications
6. Generate test cases from test case specifications



Test Cases

Identify Categories

Characteristics of each input element

split (string Str, int Size)

Input Str

- Length
- Content

Input Size

- value

Partition Categories into choices

Interesting cases (subdomains) – boundary values

split (string Str, int Size)

Input Str

- Length
  - 0
  - Size-1
- Content
  - Only Spaces
  - Special characters

Input Size

- Value
  - 0
  - >0
  - <0
  - MAXINT
  - …

Identify Constraints among choices

To Eliminate meaningless combinations & To reduce number of test cases

Three types: PROPERTY---- IF, ERROR, SINGLE

Input Str

- Length
  - 0 PROPERTY zerovalue
- Content
  - Special characters If !zerovalue

Input Size

- Value
  - <0 ERROR
  - MAXINT SINGLE

Produce And Evaluate Test Case Specifications

Can be automated

Produces test frames

Example (specify the characteristic of the inputs for that test)

Test frame #45
- Input Str
  - length: size -1
  - content: special characters
- Input Size
  - value: >0

Produce and evaluate test case specification
-how many test frames?
-add additional constraints to reduce the number if required

Generate Test Cases from Test Case Specification

Simple Instantiation of frames

Final result: Set of concrete tests

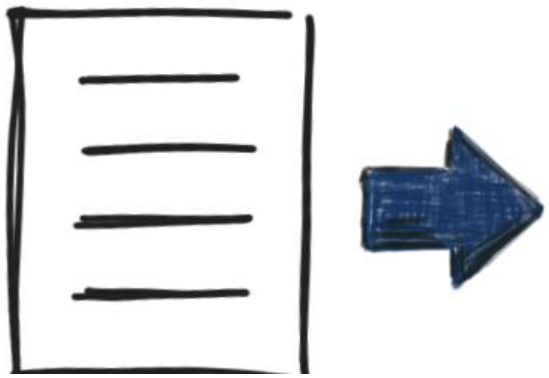
Example (specify the characteristic of the inputs for that test)

Test frame #45

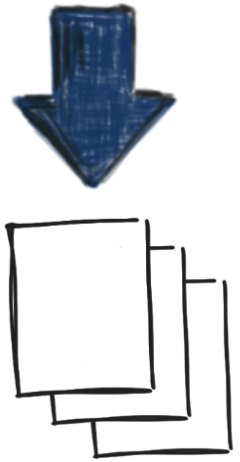
Str = “ABCC!\n\t”

Size = 10

The Category-Partition Method



1. Identify independently testable features
2. Identify Categories
3. Partition Categories into choices
4. Identify constraints among choices
5. Produce/Evaluate test case specifications
6. Generate test cases from test case specifications



Test Cases

DEMO TIME

- Use category partition to generate test frames from a specification file (with categories, partitions, and constraints)
- Tool called TSLgenerator is used: Developed by team at UC Irvine, Oregon State, and Georgia Tech
- Download from: https://github.com/alexorso/tslgenerator/tree/master/Binaries
- run the code from command prompt: **./TSLgenerator-win8.exe**
- For help: **./TSLgenerator-win8.exe –manpage**
- To get number of test cases and write the test frames against your specification file: **./TSLgenerator-win8.exe -c *filename***

Next Class:

A systematic Functional-Testing Approach => E.g. : Category Partition method

A Model Based Black-Box Testing Approach => E.g. Finite State Machine