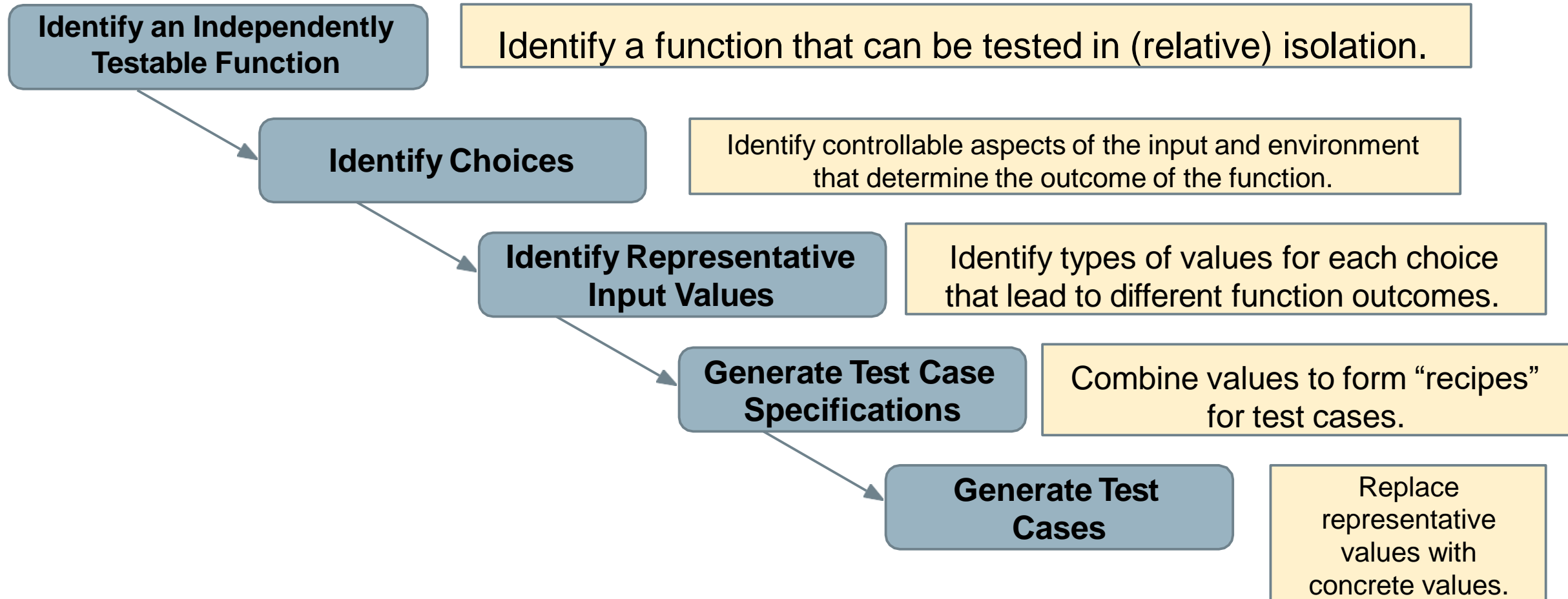


# System Testing - Test Selection Techniques

CSE 4495 - Lecture 5 - 25/072022

Instructor : Md. Mohaiminul Islam

# Creating System-Level Tests



# Test Specifications

- **May end up with thousands of test specifications.**
- Which do you turn into concrete test cases?
- **Identify the important interactions.**



# Today's Goals

- Understand how interactions can create faults.
- Examine **how to select system tests** to increase likelihood of detecting interaction faults.
  - Category-Partition Method
  - Combinatorial Interaction Testing

# Internal Interaction

- **Low-level functions are expected to interact.**
  - Usually this is planned!
  - Sometimes unplanned interactions break the system.
  - **We want to select tests that thoroughly test interactions.**

# Triggering Interactions

- **Interactions** result from combining **values** of individual **choices**.
  - Inadvertent interactions cause unexpected behavior
  - (ex. incorrect output, timing)
- Want to detect, manage, resolve inadvertent interactions.



# Fire and Flood Control



- FireControl activates sprinklers when fire detected.
- FloodControl cuts water supply when water detected on floor.
- **Interaction means building burns down.**

# WordPress Plug-Ins

[ :weather: ]



WORDPRESS

:) 8-) ;-) ...



Today's weather: [ :weather: 🤖 ]

- Weather and emoji plug-ins tested independently.
- Their interaction results in unexpected behavior.



# Selecting Test Specifications

- We want to select *interesting* specifications.
- **Category-Partition Method**
  - Apply constraints to reduce the number of specifications.
- **Combinatorial Interaction Testing**
  - Identify a subset that covers all interactions between pairs of choices.

# Category-Partition Method

# Category-Partition Method

Creates a set of test specifications.

- **Choices, representative values, and constraints.**
  - **Choices:** What you can control when testing.
  - **Representative Values:** Logical options for each choice.
  - **Constraints:** Limit certain combinations of values.
- Apply more constraints to further limit set.

# Example: Computer Configurations

- Web shop that sells custom computers.
- *A configuration is a set of options for a model.*
  - Some combinations are invalid (i.e., display port monitor with HDMI video output).
- Function: **checkConfiguration(model, configuration)**
  - What are the parameters?
  - What are the choices to be made for each parameter?

# Example: Computer Configuration

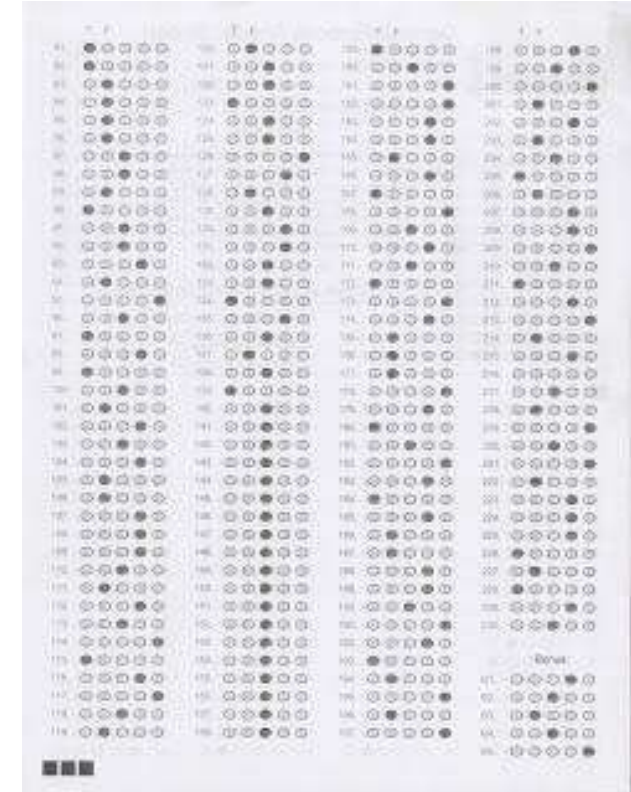
- **Model:** Identifies a product and determines constraints on available components. Identified by a model number. Characterized by a set of slots. Slots may be required (must be filled) or optional (may be left empty).
- **Configuration:** Set of <slot, component> pairs. Must correspond to the required and optional slots of the model. Available components and a default for each slot are determined by the model. Slots may be empty (may be default for optional slots). Components can be compatible or incompatible with a model or with each other.

# Example: Configuration Choices

- **Parameter: Model**
  - Model number
  - Number of required slots (must have a component)
  - Number of optional slots (component or empty)
- **Parameter: Configuration**
  - Selected configuration valid for model?
  - Number [required/optional] slots with non-empty selections.
  - Selected components for [required/optional] slots OK?
- **Parameter: Product Database**
  - Number of models in database
  - Number of components in database

# Identify Representative Values

- Many values can be selected for each choice.
- Partition each choice into *types of values*.
  - Consider all outcomes of function.
  - Consider logical ranges or groupings.
- A test specification is a selection of values for all choices.
  - Concrete test case fills values for each abstract selection.



# Values for Each Choice

## Parameter: Model

- Choice: Model number
  - malformed
  - not in database
  - valid
- Choice: Number of required slots
  - 0
  - 1
  - many
- Choice: Number of optional slots
  - 0
  - 1
  - many

## Parameter: Product Database

- Choice: Number of models in database
  - 0
  - 1
  - many
- Number of components in database
  - 0
  - 1
  - many

## Parameter: Configuration

- Choice: Configuration Matches Model
  - complete correspondence
  - omitted slots in configuration
  - extra slots in configuration
  - mismatched number of required and optional slots
- Choice: Number of required slots that are empty
  - all required slots filled
  - some required slots empty
  - all required slots empty
- Choice: Number of optional slots that are empty
  - all optional slots filled
  - some optional slots empty
  - all optional slots empty
- Choice: Selected components for required slots
  - all valid
  - some kept at default
  - $\geq 1$  incompatible with slot
  - $\geq 1$  incompatible with another component
  - $\geq 1$  not in database
- Choice: Selected components for optional slots
  - all valid
  - some kept at default
  - $\geq 1$  incompatible with slot
  - $\geq 1$  incompatible with another component
  - $\geq 1$  not in database



# Generate Test Case Specifications

- Test specification = selection of values for choices.
- **Constraints** limit number of specifications.
  - Eliminate impossible pairings.
  - Remove unnecessary options.
  - Choose a subset to turn into concrete tests.

**1944 tests (all combinations)**



**678 Tests**



**40 Tests!**

- Seven choices with three values, one with four values, two with five values.
  - $3^7 \times 5^2 \times 4 = 218700$  test specifications
- Not all combinations correspond to reasonable specifications.

#### Parameter: Model

- Choice: Model number
  - malformed
  - not in database
  - valid
- Choice: Number of required slots
  - 0
  - 1
  - many
- Choice: Number of optional slots
  - 0
  - 1
  - many

#### Parameter: Product Database

- Choice: Number of models in database
  - 0
  - 1
  - many
- Number of components in database
  - 0
  - 1
  - many

#### Parameter: Configuration

- Choice: Configuration Matches Model
  - complete correspondence
  - omitted slots in configuration
  - extra slots in configuration
  - mismatched number of required and optional slots
- Choice: Number of required slots that are empty
  - all required slots filled
  - some required slots empty
  - all required slots empty
- Choice: Number of optional slots that are empty
  - all optional slots filled
  - some optional slots empty
  - all optional slots empty
- Choice: Selected components for required slots
  - all valid
  - some kept at default
  - $\geq 1$  incompatible with slot
  - $\geq 1$  incompatible with another component
  - $\geq 1$  not in database
- Choice: Selected components for optional slots
  - all valid
  - some kept at default
  - $\geq 1$  incompatible with slot
  - $\geq 1$  incompatible with another component
  - $\geq 1$  not in database

# Constraints Between Values

- IF-CONSTRAINT
  - This value only needs to be used under certain conditions  
(**if X is true, use value Y**)
- ERROR
  - Value causes error regardless of values of other choices.
- SINGLE
  - Only a single test with this value is needed.
  - Corner cases that should give “good” outcome.

# Example - Substring

```
substr(string str, int index)
```

## Choice: Str length

length = 0 property zeroLen, TRUE if length = 0

length = 1

length >= 2

## Choice: Str contents

contains letters and numbers if !zeroLen

contains special characters if !zeroLen SINGLE

empty if zeroLen

## Choice: index

value < 0 ERROR

value = 0

value = 1

value > 1

8 (error cases) + 6 (single cases) +  $(1^7 \cdot 2^1 \cdot 3^2)$  (RSMANY = true/OSMANY = true) +  $(1^5 \cdot 2^3 \cdot 3^2)$  (false/true) +  $(1^5 \cdot 2^3 \cdot 3^2)$  (true/false) +  $(1^3 \cdot 2^5 \cdot 3^2)$  (false/false) = 464 test specifications

(after error + single): 8 (error cases) + 6 (single cases) +  $(1^3 \cdot 2^3 \cdot 3^4)$  = 662 218700 test specifications  
(after error): 8 (error cases) +  $(1^1 \cdot 2^3 \cdot 3^4 \cdot 4^2)$  = 10376

# Example - Configuration Constraints

## Parameter: Model

- Choice: Model number
  - malformed [error]
  - not in database
  - valid
- Choice: Number of required slots
  - 0 [single]
  - 1
  - many [property RSMANY]
- Choice: Number of optional slots
  - 0 [single]
  - 1
  - many [property OSMANY]

## Parameter: Product Database

- Choice: Number of models in database
  - 0 [error]
  - 1 [single]
  - many
- Number of components in database
  - 0 [error]
  - 1 [single]
  - many

## Parameter: Configuration

- Choice: Configuration Matches Model
  - complete correspondence
  - omitted slots in configuration [error]
  - extra slots in configuration [error]
  - mismatched number of required and optional slots [error]
- Choice: Number of empty required slots that are empty
  - all required slots filled
  - some required slots empty [if RSMANY]
  - all required slots empty
- Choice: Number of optional slots that are empty
  - all optional slots filled
  - some optional slots empty [if OSMANY]
  - all optional slots empty
- Choice: Selected components for required slots
  - all valid
  - some kept at default [single]
  - $\geq 1$  incompatible with slot
  - $\geq 1$  incompatible with another component
  - $\geq 1$  not in database [error]
- Choice: Selected components for optional slots
  - all valid
  - some kept at default [single]
  - $\geq 1$  incompatible with slot
  - $\geq 1$  incompatible with another component
  - $\geq 1$  not in database [error]

# Activity - find service

## **find(pattern,file)**

- Finds instances of a pattern in a file
  - **find("john",myFile)**
    - Finds all instances of john in the file
  - **find("john smith",myFile)**
    - Finds all instances of john smith in the file
  - **find("“john” smith",myFile)**
    - Finds all instances of “john” smith in the file

# Activity - find Service

- Parameters: pattern, file
- What can we vary for each?
  - What can we control about the pattern? Or the file?
- What values can we choose for each choice?
  - **File name:**
    - File exists with that name
    - File does not exist with that name
- What constraints can we apply between choice values? (if, single, error)

# Example - find Service

- Pattern size:
  - Empty
  - single character
  - many characters
  - longer than any line in the file
- Quoting:
  - pattern has no quotes
  - pattern has proper quotes
  - pattern has improper quotes (only one “)
- Embedded spaces:
  - No spaces
  - One space
  - Several spaces

**$(2^2 * 3^3 * 4^1) = 108$  test specifications**

- File name:
  - Existing file name
  - no file with this name
- Number of occurrence of pattern in file:
  - None
  - exactly one
  - more than one
- Pattern occurrences on any single line line:
  - One
  - more than one



# ERROR and SINGLE Constraints

$$4 \text{ (error)} + 2 \text{ (single)} + (1^2 * 2^3 * 3^1) = 30$$

- Pattern size:
  - [error]** • Empty
  - single character
  - many character
  - [error]** • longer than any line in the file
- Quoting:
  - pattern has no quotes
  - pattern has proper quotes
  - [error]** • pattern has improper quotes (only one “)
- Embedded spaces:
  - No spaces
  - One space
  - Several spaces
- File name:
  - Existing file name
  - no file with this name **[error]**
- Number of occurrence of pattern in file:
  - None
  - exactly one **[single]**
  - more than one
- Pattern occurrences on target line:
  - One
  - more than one **[single]**

# IF Constraints

$$4 \text{ (error)} + 2 \text{ (single)} + (1^3 * 2^3) \text{ (quoted = true)} + (1^4 * 2^2) \text{ (quoted = false)} = 18$$

- Pattern size:
  - [error]** • Empty
  - single character
  - many character
  - [error]** • longer than any line in the file
- Quoting:
  - pattern has no quotes
  - [property quoted\*]** pattern has proper quotes
  - [error]** • pattern has improper quotes (only one “)
- Embedded spaces:
  - No spaces
  - [if quoted]** • One space
  - [if quoted]** • Several spaces

- File name:
  - Existing file name
  - no file with this name **[error]**
- Number of occurrence of pattern in file:
  - None
  - exactly one **[single]**
  - more than one
- Pattern occurrences on target line:
  - One
  - more than one **[single]**

**Let's take a break.**

# Combinatorial Interaction Testing

# Limiting Num. of Test Specifications

Bandwidth Mode	Language	Fonts
Desktop Site	English	Standard
Mobile Site	French	Open-Source
Text Only	German	Minimal
	Swedish	
Advertising	Screen Size	
No Advertising	Phone	
Targeted Advertising	Tablet	
General Advertising	Full Size	
Minimal Advertising		

- Full set = 432 specifications
- No natural IF, SINGLE, ERROR constraints for these features.
- What is important to cover?

# Combinatorial Interaction Testing

- Cover all  $k$ -way interactions ( $k < N$ ).
  - Typically **2-way (pairwise)** or 3-way.
- Set of all combinations grows exponentially.
- Set of pairwise combinations grows logarithmically.
  - (last slide) 432 combinations.
  - Possible to cover all pairs in 16 tests.

# Example - Paragraph Effects

Paragraph spaces has two values: **selected** and **unselected**.  
Mirror indents has two values: **selected** and **unselected**.  
And finally, line spacing has three values: **single**, **multiple** and **double**.

## Paragraph Space

☐ Don't add space between paragraphs of the same style

## Indentation

☐ Mirror Indents

## Line Spacing

Single

Paragraph Space	Indentation	Line Spacing
Selected	Selected	Single
Unselected	Unselected	Double
		Multiple

**$2 * 2 * 3 = 12$**   
**combinations**

# Example - Paragraph Effects

Single	Indent Selected	Paragraph Selected
Single	Indent Unselected	Paragraph Selected
Single	Indent Selected	Paragraph Unselected
Single	Indent Unselected	Paragraph Unselected
Multiple	Indent Selected	Paragraph Selected
Multiple	Indent Unselected	Paragraph Selected
Multiple	Indent Selected	Paragraph Unselected
Multiple	Indent Unselected	Paragraph Unselected
Double	Indent Selected	Paragraph Selected
Double	Indent Unselected	Paragraph Selected
Double	Indent Selected	Paragraph Unselected
Double	Indent Unselected	Paragraph Unselected

Single	Indent Selected	Paragraph Selected
Single	Indent Unselected	Paragraph Selected
Single	Indent Selected	Paragraph Unselected
Single	Indent Unselected	Paragraph Unselected
Multiple	Indent Selected	Paragraph Selected
Multiple	Indent Unselected	Paragraph Selected
Multiple	Indent Selected	Paragraph Unselected
Multiple	Indent Unselected	Paragraph Unselected
Double	Indent Selected	Paragraph Selected
Double	Indent Unselected	Paragraph Selected
Double	Indent Selected	Paragraph Unselected
Double	Indent Unselected	Paragraph Unselected

Single	Indent Selected	Paragraph Selected
Single	Indent Unselected	Paragraph Selected
Single	Indent Selected	Paragraph Unselected
Single	Indent Unselected	Paragraph Unselected
Multiple	Indent Selected	Paragraph Selected
Multiple	Indent Unselected	Paragraph Selected
Multiple	Indent Selected	Paragraph Unselected
Multiple	Indent Unselected	Paragraph Unselected
Double	Indent Selected	Paragraph Selected
Double	Indent Unselected	Paragraph Selected
Double	Indent Selected	Paragraph Unselected
Double	Indent Unselected	Paragraph Unselected



# Example - Paragraph Effects

- Goal of CIT is to produce **covering array**.
  - Set of configurations that covers all K-way combinations.
    - (2-way here).
  - Cover in 6 test specifications.

Single	Indent Selected	Paragraph Selected
Single	Indent Unselected	Paragraph Unselected
Multiple	Indent Selected	Paragraph Selected
Multiple	Indent Unselected	Paragraph Unselected
Double	Indent Selected	Paragraph Unselected
Double	Indent Unselected	Paragraph Selected

# Example - Website Display

Bandwidth Mode
Desktop Site
Mobile Site
Text Only
Fonts
Standard
Open-Source
Minimal
Screen Size
Phone
Tablet
Full Size

Bandwidth Mode	Fonts	Screen Size
Desktop Site	Standard	Phone
Desktop Site	Open-Source	Tablet
Desktop Site	Minimal	Full Size
Mobile Site	Standard	Tablet
Mobile Site	Open-Source	Full Size
Mobile Site	Minimal	Phone
Text Only	Standard	Full Size
Text Only	Open-Source	Phone
Text Only	Minimal	Tablet

- Cover all combinations for two variables.
- Add a third, account for all combinations of pairs of values.
  - Each test specification can cover up to three pairs.

# Example

Bandwidth Mode	Language	Fonts
Desktop Site	English	Standard
Mobile Site	French	Open-Source
Text Only	German	Minimal
	Swedish	
Advertising	Screen Size	
No Advertising	Phone	
Targeted Advertising	Tablet	
General Advertising	Full Size	
Minimal Advertising		

Language	Advertising	Bandwidth Mode	Fonts	Screen Size
English	No Advertising	Desktop Site	Standard	Phone
English	Targeted Advertising	Mobile Site	Open-Source	Tablet
English	General Advertising	Text Only	Minimal	Full Size
English	Minimal Advertising	Mobile Site	Minimal	Phone
French	No Advertising	-	-	-
French	Targeted Advertising	Desktop Site	Minimal	Full Size
French	General Advertising	Mobile Site	Standard	Tablet
French	Minimal Advertising	Text Only	Open-Source	Phone
German	No Advertising	Text Only	Minimal	Tablet
German	Targeted Advertising	-	-	-
German	General Advertising	Desktop Site	Open-Source	Phone
German	Minimal Advertising	Mobile Site	Standard	Full Size
Swedish	No Advertising	Mobile Site	Open-Source	Full Size
Swedish	Targeted Advertising	Text Only	Standard	Phone
Swedish	General Advertising	-	-	-
Swedish	Minimal Advertising	Desktop Site	Minimal	Tablet

# Constraints

- Remove all ERROR/SINGLE cases before CIT.
  - Error output, one-time corner cases
- Constraints on value combinations specified:
  - OMIT(Text-Only, \*, \*, Full Size, \*)
  - OMIT(\*, \*, \*, Full Size, Minimal)
- Further reduces number of test specifications.

# CIT Tools

- Pairwise Independent Combinatorial Testing (Microsoft): <https://github.com/microsoft/pict>
- Automated Combinatorial Testing for Software (NIST):  
<https://csrc.nist.gov/projects/automated-combinatorial-testing-for-software>
- .. Many more: <https://www.pairwise.org/tools.html>

# Activity - Browser Configuration

Allow Content to Load	Notify About Pop-Ups	Allow Cookies	Warn About Add-Ons	Warn About Attack Sites	Warn About Forgeries
Allow	Yes	Allow	Yes	Yes	Yes
Restrict	No	Restrict	No	No	No
Block		Block			

- Full set of test specifications = 144
- Create set covering all pairwise value combinations.
  - Hint: Start with two variables with most values. Add one variable at a time.

# Activity Solution

<b>Allow Content</b>	<b>Allow Cookies</b>	<b>Pop-Ups</b>	<b>Add-Ons</b>	<b>Attacks</b>	<b>Forgeries</b>
Allow	Allow	Yes	Yes	Yes	Yes
Allow	Restrict	No	No	Yes	No
Allow	Block	No	No	No	Yes
Restrict	Allow	Yes	-	No	No
Restrict	Restrict	Yes	-	-	Yes
Restrict	Block	No	Yes	Yes	No
Block	Allow	No	-	-	Yes
Block	Restrict	-	Yes	No	-
Block	Block	Yes	No	Yes	No

# We Have Learned

- Process for deriving system-level tests often results in **too many test specifications**.
- Two methods that **identify important interactions**:
  - **Category-Partition Method**: Use constraints to eliminate unnecessary tests.
  - **Combinatorial Interaction Testing**: Identify important pairs of input values.



# Next Time

- Exercise Session Tomorrow:
  - More practice in system-level testing.
- Unit Testing

**Thank You**