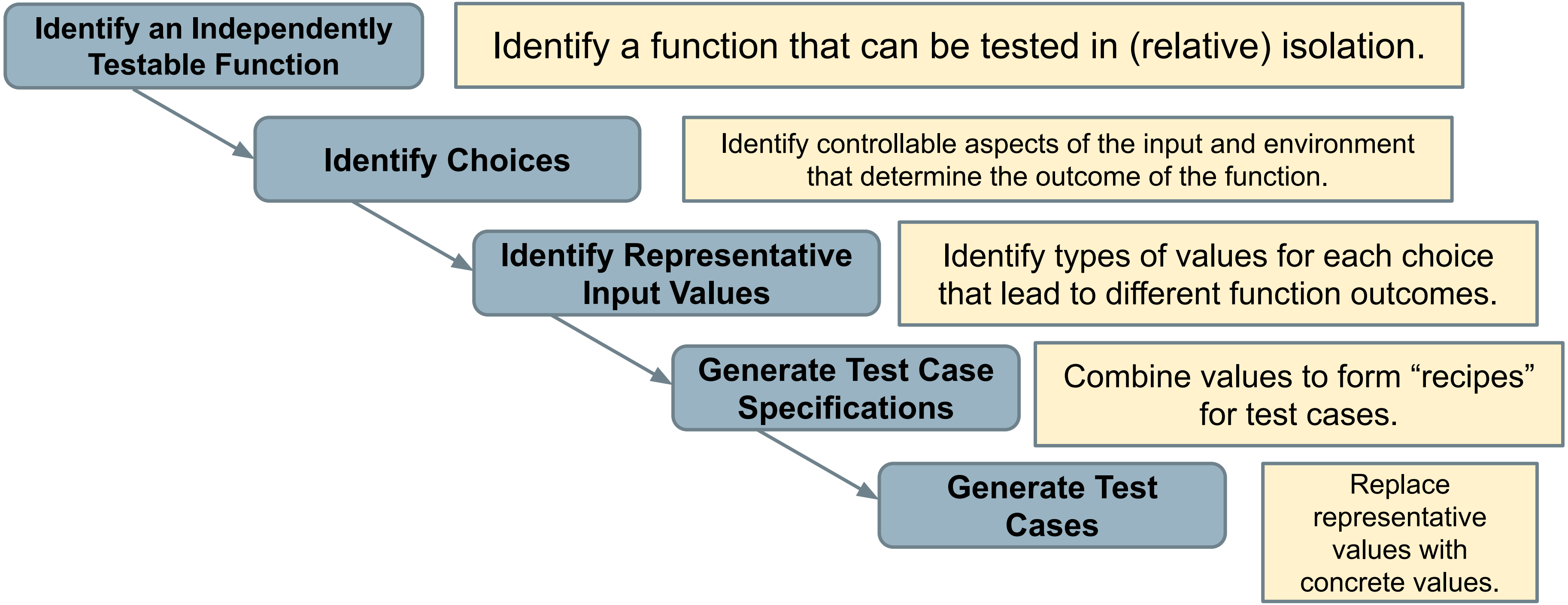
CHALMERS UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG

Lecture 6: System Testing - Test Selection Techniques

Gregory Gay
DIT636/DAT560 - February 1, 2023

Creating Test Cases



Identify an Independently Testable Function

Identify a function that can be tested in (relative) isolation.

Identify Choices

Identify controllable aspects of the input and environment that determine the outcome of the function.

Identify Representative Input Values

Identify types of values for each choice that lead to different function outcomes.

Generate Test Case Specifications

Combine values to form “recipes” for test cases.

Generate Test Cases

Replace representative values with concrete values.

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

- Examine how component interactions can create faults and failures.
- Examine **how to select system tests** to increase likelihood of detecting integration faults.
  - Category-Partition Method
  - Combinatorial Interaction Testing

Component Interactions

- Components are expected to interact.
  - Usually this is planned!
  - Sometimes unplanned interactions break the system.
  - **We should select tests that thoroughly test component integrations.**

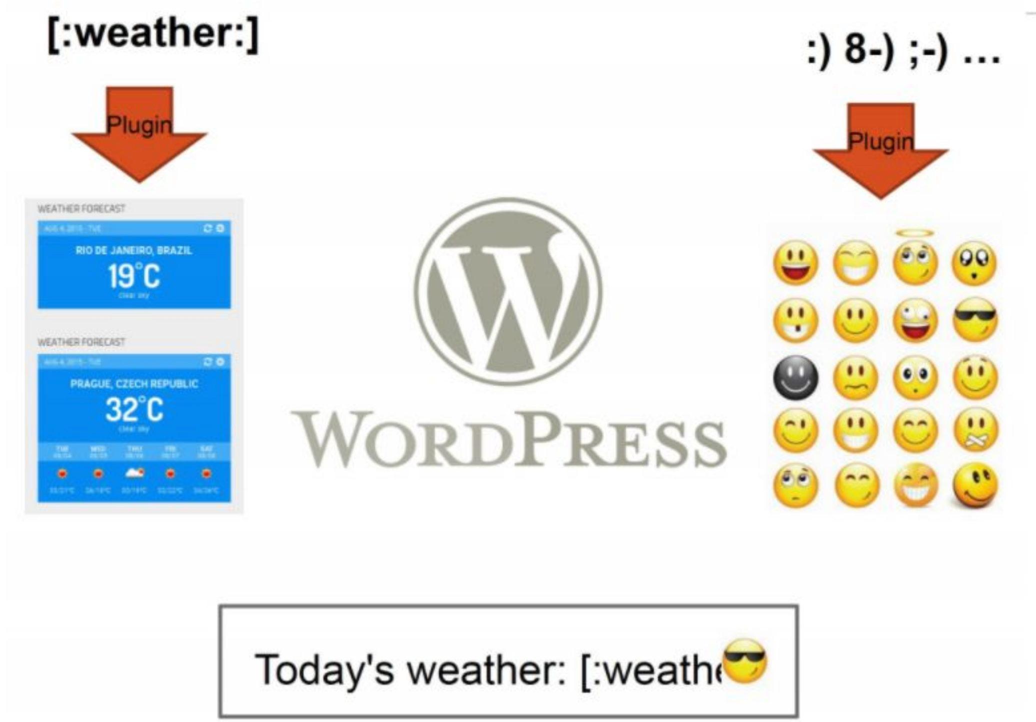
Component Interactions

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

Component Interactions

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.

Identify Choices

- Examine parameters of function.
  - *Direct input*, *environmental parameters (i.e., databases)*, and *configuration options*.
- Identify characteristics of each parameter.
  - What aspects influence outcome? (**the choices**)
- **Choices** are also called ***categories*** if you look up category-partition method.

Example - Set Functions

- Small function library related to Sets:
  - `POST /insert/SETID {“object”: VALUE}`
    - `Returns { “result”: VALUE (“OK” if success or error)}`
  - `GET /find/SETID {“object”: VALUE}`
    - `Returns { “result”: VALUE (TRUE or FALSE)}`
  - `GET /delete/SETID {“object”: VALUE}`
    - `Returns { “result”: VALUE (“OK” if success or error)}`
- We want to write tests for these three functions.

Example - Set Functions

Identify an Independently Testable Function

POST /insert/SETID {“object”: VALUE}

- What are our choices?

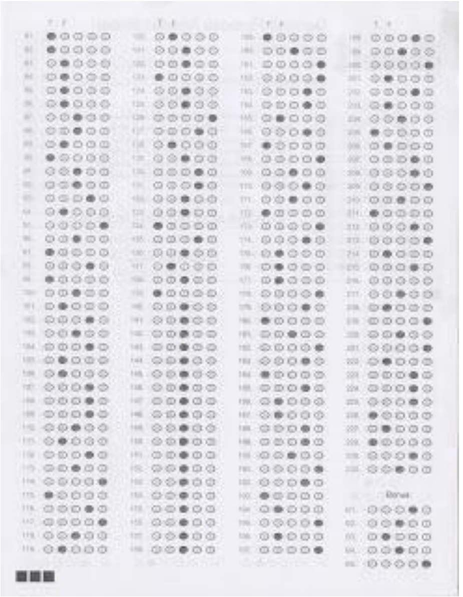
```
// Set up the existing set, either empty or
with items.
POST /insert/ {“set”: [ …]}

// Insert an object
POST /insert/SETID {“object”: VALUE}

// Check the result
pm.test(“Insertion”, function() {
  var jsonData = pm.response.json();
  pm.expect(jsonData.result).to.eql(VALUE);});
```

Identify Choices

- **Parameter:** Set ID
  - **Choice 1:** How many items are in the set? (performance may degrade with larger sets)
- **Parameter:** Object
  - **Choice 2:** Is obj already in the set?
  - **Choice 3:** Is the object valid? (e.g., not null)?

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.



Example - Set Functions

Identify Representative Input Values

```
POST /insert/SETID {“object”: VALUE}
```

Parameter: Set ID

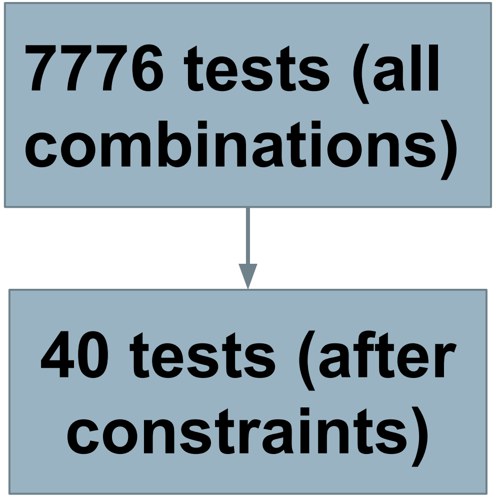
- **Choice:** How many items are in the set?
  - **Representative Values:**
    - Empty Set
    - Set with 1 item
    - Set with 10 items
    - Set with 10000 items

Parameter: object

- **Choice:** Is the object already in the set?
  - **Representative Values:**
    - obj already in set
    - obj not in set
- **Choice:** Is the object valid?
  - **Representative Values:**
    - Valid obj
    - Null obj

Generate Test Case Specifications

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



7776 tests (all combinations)

40 tests (after constraints)

Example - Set Functions

Generate Test Case Specifications

Set Size	Obj in Set	Obj Status	Outcome
Empty	Yes	Valid	No change
Empty	Yes	Null	Error
Empty	No	Valid	Obj added to Set
Empty	No	Null	Error
1 item	Yes	Valid	No change
1 item	Yes	Null	Error
1 item	No	Valid	Obj added to Set
1 item	No	Null	Error
10 items	Yes	Valid	No change
10 items	Yes	Null	Error
10 items	No	Valid	Obj added to Set
10 items	No	Null	Error

```
POST /insert/SETID
{“object”: VALUE}
```

- (4 * 2 * 2) = 16 specifications
- Each can become 1+ tests.
- Use constraints to remove impossible combinations.

Set Size	Obj in Set	Obj Status	Outcome
10000	Yes	Valid	No change (may be slowdown)
10000	Yes	Null	Error
10000	No	Valid	Obj added to Set(may be slowdown)
10000	No	Null	Error (may be slowdown)

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

Example - Set Functions

Identify Constraints

```
POST /insert/SETID {“object”: VALUE}
```

Parameter: set

- **Choice:** How many items are in the set?
  - **Representative Values:**
    - Empty Set property empty
    - Set with 1 item
    - Set with 10 items single
    - Set with 10000 items single

Parameter: obj

- **Choice:** Is the object already in the set?
  - **Representative Values:**
    - obj already in set if !empty
    - obj not in set
- **Choice:** Is the object valid?
  - **Representative Values:**
    - Valid obj
    - Null obj error

Example - Set Functions

Apply Constraints

Set Size	Obj in Set	Obj Status	Outcome
~~Empty~~	~~Yes~~	~~Valid~~	~~No change~~
~~Empty~~	~~Yes~~	~~Null~~	~~Error~~
Empty	No	Valid	Obj added to Set
Empty	No	Null	Error
1 item	Yes	Valid	No change
~~1 item~~	~~Yes~~	~~Null~~	~~Error~~
1 item	No	Valid	Obj added to Set
~~1 item~~	~~No~~	~~Null~~	~~Error~~
~~10 items~~	~~Yes~~	~~Valid~~	~~No change~~
~~10 items~~	~~Yes~~	~~Null~~	~~Error~~
10 items	No	Valid	Obj added to Set
~~10 items~~	~~No~~	~~Null~~	~~Error~~

```
POST /insert/SETID
{“object”: VALUE}
```

(4 * 2 * 2) = 16 specifications

Can’t already be in empty set, - 2

error (null), - 6

single (10, 10000), - 2

Set Size	Obj in Set	Obj Status	Outcome
~~10000~~	~~Yes~~	~~Valid~~	~~No change (may be slowdown)~~
~~10000~~	~~Yes~~	~~Null~~	~~Error (may be slowdown)~~
10000	No	Valid	Obj added to Set(may be slowdown)
~~10000~~	~~No~~	~~Null~~	~~Error (may be slowdown)~~

Example - Set Functions

Apply Constraints

Set Size	Obj in Set	Obj Status	Outcome
Empty	No	Valid	Obj added to Set
Empty	No	Null	Error
1 item	Yes	Valid	No change
1 item	No	Valid	Obj added to Set
10 items	No	Valid	Obj added to Set
10000	No	Valid	Obj added to Set(may be slowdown)

```
POST /insert/SETID
{“object”: VALUE}
```

- From 16 -> 6 specifications
- Each can become 1+ tests.
- Can further constrain if needed.

Example - Set Functions

Create Test Cases

POST /insert/SETID {“object”: VALUE}

Set Size	Obj in Set	Obj Status	Outcome
Empty	No	Valid	Obj added to Set

Set Size	Obj in Set	Obj Status	Outcome
Empty	No	Null	Error

```
// Set up empty set.
POST /insert/ {“set”: []}
// Insert a valid object
POST /insert/SETID {“object”: “Test”}
// Check the result
pm.test(“Valid Insert”, function() {
  var jsonData = pm.response.json();
pm.expect(jsonData.result).to.eql(“OK”);
});
```

```
// Set up empty set.
POST /insert/ {“set”: []}
// Insert a null object
POST /insert/SETID {“object”: null}
// Check the result
pm.test(“Null Insert”, function() {
  var jsonData = pm.response.json();
pm.expect(jsonData.result).to.eql(“Null object
cannot be inserted into set”);});
```

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)

Let’s take a break.

Example - `find` Service

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

Pattern:

- 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

File:

- 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 (error) + 2 (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 (error) + 2 (single) + (1^3*2^3) (quoted = true) + (1^4*2^2) (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]**

Combinatorial Interaction Testing

Limiting Num. of Test Specifications

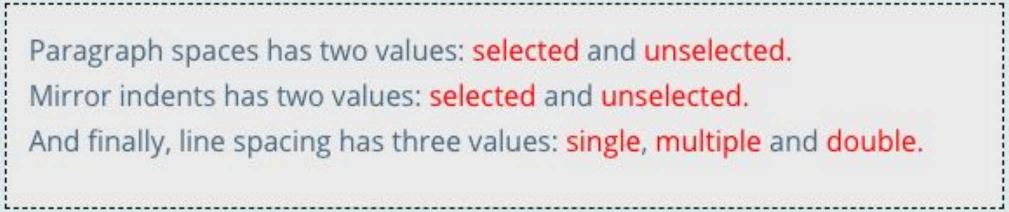
Bandwidth Mode	**Language**	**Fonts**
Desktop Site	English	Standard
Mobile Site	French	Open-Source
Text Only	German	Minimal
	Swedish	
Advertising	**Screen Size**	
No Advertising	Phone	
Targeted Advertising	Tablet	
General Advertising	Full Size	
Minimal Advertising		

- Full set = 432 specifications
- Few natural IF, SINGLE, ERROR constraints for these features.
- What is important to cover?

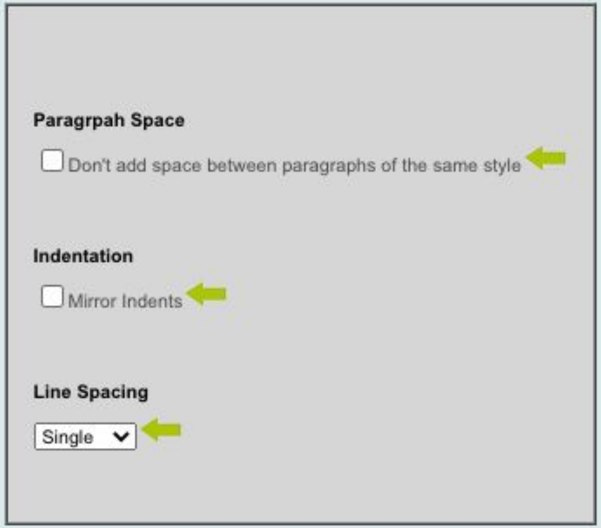
Combinatorial Interaction Testing

- Cover all 2-way (pairwise) interactions.
  - **Can cover multiple pairs with one test case.**
- Set of all combinations grows exponentially.
- Set of pairwise combinations grows logarithmically.
  - (last slide) 432 combinations.
  - Possible to cover all **pairs of choices** 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.



Paragrpah Space

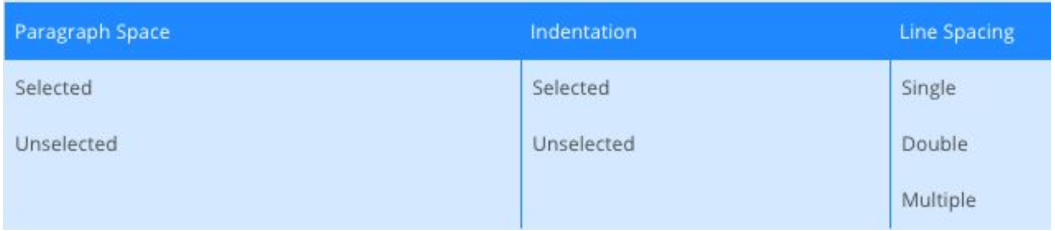
☐ Don't add space between paragraphs of the same style

Indentation

☐ Mirror Indents

Line Spacing

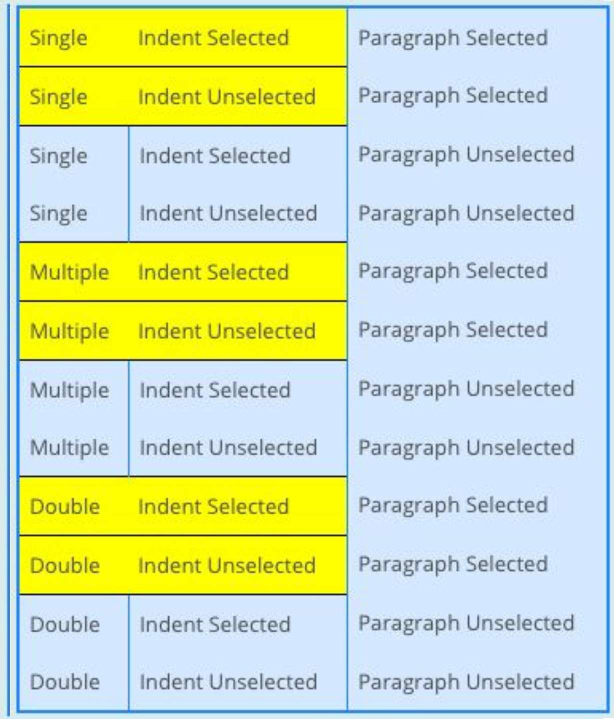
Single



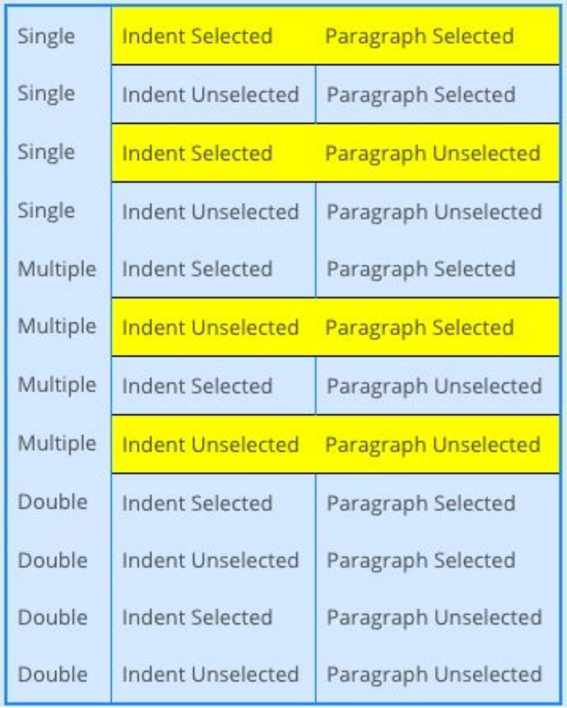
Paragraph Space	Indentation	Line Spacing
Selected	Selected	Single
Unselected	Unselected	Double
		Multiple

2 * 2 * 3 = 12 combinations

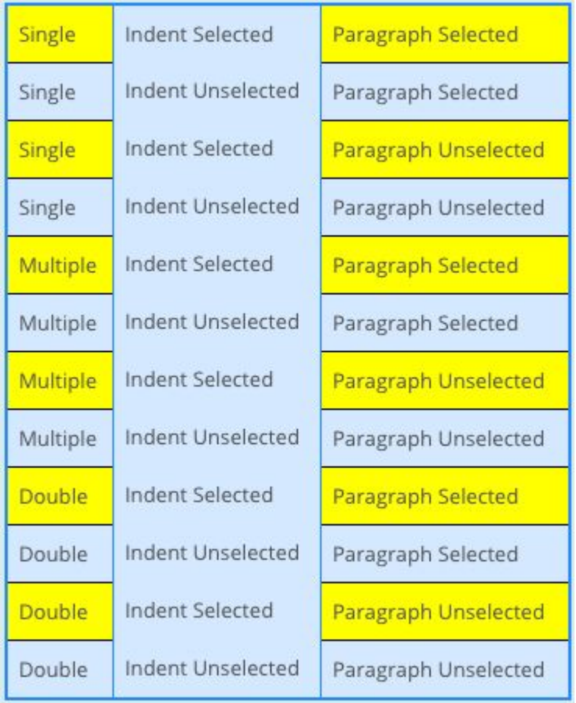
Example - Paragraph Effects



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



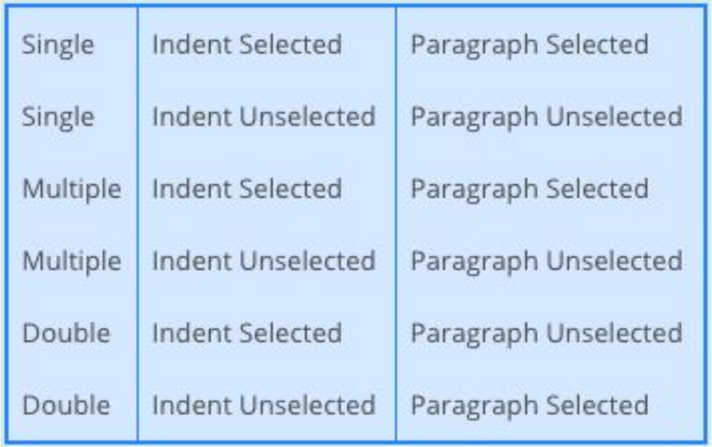
Single	Indent Selected	Paragraph Selected
Single	Indent Unselected	Paragraph Selected
Single	Indent Selected	Paragraph Unselected
Single	Indent Unselected	Paragraph Unselected
Multiple	Indent Selected	Paragraph Selected
Multiple	Indent Unselected	Paragraph Selected
Multiple	Indent Selected	Paragraph 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 2-way combinations.
  - Cover in 6 test cases.



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 - W

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

Activity - Browser Configuration

Choices and Representative Values

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

Allow Content	Allow Cookies	Pop-Ups	Add-Ons	Attacks	Forgeries
Allow	Allow	Yes	Yes	Yes	Yes
Allow	Restrict	No	No	Yes	No
Allow	Block	No	No	No	Yes
Restrict	Allow	Yes	No	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

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: http://www.pairwise.org/tools.asp

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:
  - Practice in system-level test design.
- Next Tuesday:
  - Exploratory Testing
- Assignment 1 - Feb 12
  - All topics now covered.
  - Any questions?

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY