Lecture 03:
BBT: Cause-Effect Graphing & Combinatorial Testing

Spring 2022

Dietmar Pfahl
email: dietmar.pfahl@ut.ee
Lectures

- Lecture 1 (10.02) – Introduction to Software Testing
- Lecture 2 (17.02) – Basic Black-Box Testing Techniques: Boundary Value Analysis & Equivalence Class Partitioning
- Lecture 3 (03.03) – BBT advanced: Combinatorial Testing
- Lecture 4 (17.02) – Basic White-Box Testing Techniques: Control-Flow Coverage
- Lecture 5 (17.03) – BBT adv.: State-Transition, Metamorphic, Random Testing
- Lecture 6 (24.03) – Test Levels, Test Tools, Test Automation
- Lecture 7 (31.03) – BBT adv.: Exploratory Testing, Behaviour Testing
- Lecture 9 (14.04) – Security Testing of Mobile Applications
- Lecture 10 (21.04) – WBT adv.: Data-Flow Testing / Mutation Testing
- Lecture 12 (05.05) – Defect Estimation / Test Documentation, Organisation and Process Improvement (Test Maturity Model)
- Lecture 13 (12.05) – Exam Preparation
- Lecture 14 (19.05) – Advanced Topics (optional)
Black-Box vs. White-Box

Specification-based Testing: Test against specification

Structural Testing: Test against implementation

System

Specification

Implementation

Missing functionality: Cannot be (directly) revealed by white-box techniques

Unexpected functionality: Cannot be (directly) revealed by black-box techniques
Black-Box Testing Techniques

- Equivalence class partitioning (ECP)
- Boundary value analysis (BVA)
- Cause-effect graphing
- Combinatorial testing
- State transition testing (State-based testing)
- Metamorphic testing
- Random testing
- Exploratory testing
- Usability testing, A/B testing (UX)
Black-Box Testing Techniques

- Equivalence class partitioning (ECP)
- Boundary value analysis (BVA)
- **Cause-effect graphing**
- Combinatorial testing
- State transition testing (State-based testing)
- Metamorphic testing
- Random testing
- Exploratory testing
- Usability testing, A/B testing (UX)
Example – Insurance System

Specification Statement:

• System shall reject over-age insurance applicants

Specification Item:

• Reject male insurance applicants, if over the age of 80 years on day of application

• Reject female insurance applicants, if over the age of 85 years on day of application
Example (cont.)
Input: Gender & Age | Output: accept/reject

**UI – Case A**

<table>
<thead>
<tr>
<th>Age:</th>
<th>18, 80</th>
<th>O</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(80, 85)</td>
<td>O</td>
</tr>
<tr>
<td></td>
<td>(85, 99)</td>
<td>O</td>
</tr>
<tr>
<td>Other?</td>
<td>O</td>
<td></td>
</tr>
<tr>
<td><strong>Gender:</strong></td>
<td>male</td>
<td>O</td>
</tr>
<tr>
<td></td>
<td>female</td>
<td>O</td>
</tr>
</tbody>
</table>

Enter

**Result:** <text>

**Message:** <text>

Result text in {<empty>, accept, reject}
Message text in {<empty>, missing input}

**UI – Case B**

Please enter gender (m, f):
<Message text>

Please enter age (integer>0):
<Message text>

**Result:** <text>

Message text in {
<empty>,
invalid input – retry or quit with Ctrl^D}

Result text in {
accept,
reject}
Example – UI Case A
Input: Gender & Age | Output: accept/reject

**Equ. Classes**

EC1: InputAge: [18, 80]
EC2: InputAge: (80, 85]
EC3: InputAge: (85, 99]
EC4: InputAge: other
EC5: InputAge: <empty>
EC6: InputGender: Male
EC7: InputGender: Female
EC8: InputGender: <empty>

**Test Cases**

Data: age, gender, result, message

EC9: OutputResult: <empty>
EC10: OutputResult: ‘accept’
EC11: OutputResult: ‘reject’
EC12: OutputMsg: <empty>
EC13: OutputMsg: ’missing input’

minimal,
TCs cover all classes
Example – UI Case A

Input: Gender & Age | Output: accept/reject

Equ. Classes

EC1: InputAge: [18, 80]
EC2: InputAge: (80, 85]
EC3: InputAge: (85, 99]
EC4: InputAge: other
EC5: InputAge: <empty>
EC6: InputGender: Male
EC7: InputGender: Female
EC8: InputGender: <empty>

EC9: OutputResult: <empty>
EC10: OutputResult: ‘accept’
EC11: OutputResult: ‘reject’
EC12: OutputMsg: <empty>
EC13: OutputMsg: ’missing input’

Test Cases

Data: age, gender, result, message

TC1: <empty>, <empty>, <empty>, ’missing input’
TC2: 56, male, ’accept’, <empty>
TC3: 83, male, ’reject’, <empty>
TC4: 88, female, ’reject’, <empty>
TC5: other, female, ’reject’, <empty>

minimal,
TCs cover all classes
Example – UI Case A

Input: Gender & Age | Output: accept/reject

Equ. Classes

EC1: InputAge: [18, 80]
EC2: InputAge: (80, 85]
EC3: InputAge: (85, 99]
EC4: InputAge: other
EC5: InputAge: <empty>
EC6: InputGender: Male
EC7: InputGender: Female
EC8: InputGender: <empty>
EC9: OutputResult: <empty>
EC10: OutputResult: ‘accept’
EC11: OutputResult: ‘reject’
EC12: OutputMsg: <empty>
EC13: OutputMsg: ’missing input’

Test Cases

TC1: <empty>, male, <empty>, ’missing input’
TC2: other, <empty>, <empty>, ’missing input’
TC3: 56, male, ’accept’, <empty>
TC4: 83, male, ’reject’, <empty>
TC5: 88, female, ’reject’, <empty>
TC6: other, female, ’reject’, <empty>

If we consider ’missing input’ to be an error message caused by invalid input (<empty>), then it’s good practice to check for the effect of each invalid input class independently.
Example – UI Case A

Input: Gender & Age | Output: accept/reject

<table>
<thead>
<tr>
<th>In/Out</th>
<th>Valid EC (C)</th>
<th>Invalid EC (C)</th>
<th>TC</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Age</td>
<td>C1: [18, 80]</td>
<td>C5: &lt;empty&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>C2: (80, 85]</td>
<td></td>
<td></td>
<td>em</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>C3: (85, 99]</td>
<td></td>
<td></td>
<td></td>
<td>em</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>C4: other</td>
<td></td>
<td></td>
<td></td>
<td>em</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Gender</td>
<td>C6: Male</td>
<td>C8: &lt;empty&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>C7: Female</td>
<td></td>
<td></td>
<td></td>
<td>em</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Result</td>
<td>C10: ‘acc’</td>
<td>C9: &lt;empty&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>C11: ‘reject’</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>em</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Message</td>
<td>C12: &lt;emp&gt;</td>
<td>C13: &lt;miss. In&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Minimum set of tests covering all ECs. All invalid input ECs covered individually.
Example – UI Case A

Input: Gender & Age | Output: accept/reject

Equ. Classes

EC1: InputAge: [18, 80]
EC2: InputAge: (80, 85]
EC3: InputAge: (85, 99]
EC4: InputAge: other
EC5: InputAge: <empty>
EC6: InputGender: Male
EC7: InputGender: Female
EC8: InputGender: <empty>
EC9: OutputResult: <empty>
EC10: OutputResult: ‘accept’
EC11: OutputResult: ‘reject’
EC12: OutputMsg: <empty>
EC13: OutputMsg: ’missing input’

Test Cases

TC1: <empty>, male, <empty>, ’missing input’
TC2: other, <empty>, <empty>, ’missing input’
TC3: 56, male, ’accept’, <empty>
TC4: 83, male, ’reject’, <empty>
TC5: 88, female, ’reject’, <empty>
TC6: other, female, ’reject’, <empty>

What if this valid input changes to ‘female’?
Example – UI Case A

Input: Gender & Age | Output: accept/reject

**Equ. Classes**

EC1: InputAge: [18, 80]
EC2: InputAge: (80, 85]
EC3: InputAge: (85, 99]
EC4: InputAge: other
EC5: InputAge: <empty>
EC6: InputGender: Male
EC7: InputGender: Female
EC8: InputGender: <empty>
EC9: OutputResult: <empty>
EC10: OutputResult: ‘accept’
EC11: OutputResult: ‘reject’
EC12: OutputMsg: <empty>
EC13: OutputMsg: ‘missing input’

**Test Cases**

TC1: <empty>, male, <empty>, ’missing input’
TC2: other, <empty>, <empty>, ’missing input’
TC3: 56, male, ’accept’, <empty>
TC4: 83, male, ’reject’, <empty>
TC4*: 83, female, ’accept’, <empty>
TC5: 88, female, ’reject’, <empty>
TC6: other, female, ’reject’, <empty>

In TC4: change from ’male’ to ’female’ causes output to change from ’reject’ to ’accept’. Now we have covered all cause-effect relationships.
Cause-Effect Graphing

1. The tester must decompose the specification into lower level units.
2. For each specification unit, the tester needs to identify causes and effects.
   - A cause is a distinct input condition (or equivalence class of input conditions).
   - An effect is an output condition (or equivalence class of output conditions) or a system transformation.
   - A table of causes and effects helps the tester to record the necessary details.
   - Then relationships between the causes and effects should be determined, ideally in the form of a set of rules (business rules).
3. From the set of rules, the boolean cause-effect graph is constructed, which in turn is converted into a decision table.
4. Test cases are then generated from the decision table.
Cause-Effect Graphing

1. The tester must decompose the specification into lower level units.

2. For each specification unit, the tester needs to identify causes and effects.
   - A cause is a distinct input condition (or equivalence class of input conditions)
   - An effect is an output condition (or equivalence class of output conditions) or a system transformation
   - A table of causes and effects helps the tester to record the necessary details.
   - Then relationships between the causes and effects should be determined, ideally in the form of a set of rules (business rules).

3. From the set of rules, the boolean cause-effect graph is constructed, which in turn is converted into a decision table.

4. Test cases are then generated from the decision table.

Rules (simplified – only valid ECs):

- If below or equal 80 years \( \rightarrow \) accept
- If above 85 years \( \rightarrow \) reject
- If in \((80, 85]\) and ‘female’ \( \rightarrow \) accept
- If in \((80, 85]\) and ‘male’ \( \rightarrow \) reject
Cause-Effect Graphing

1 and/or 3

2

1

2

2

1

2

‘3’ occurs if both/one of ‘1’ and ‘2’ are present

‘2’ occurs if ‘1’ occurs

‘2’ occurs if ‘1’ does not occur
Cause-Effect Graphing

Gender

<table>
<thead>
<tr>
<th>Age</th>
<th>&lt;=80</th>
<th>&lt;80</th>
<th>&lt;=85</th>
<th>&gt;85</th>
</tr>
</thead>
<tbody>
<tr>
<td>male</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>female</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Accept/Reject

<table>
<thead>
<tr>
<th>Age</th>
<th>&lt;=80</th>
<th>&lt;80</th>
<th>&lt;=85</th>
<th>&gt;85</th>
</tr>
</thead>
<tbody>
<tr>
<td>accept</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>reject</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Male
- Female
- Age <= 80
- 80 < Age <= 85
- Age > 85

- Accept
- Reject
One advantage of this method is that development of the rules and the graph from the specification allows for a thorough inspection (logically, semantically) of the specification.
### Decision Table Format

<table>
<thead>
<tr>
<th>Case</th>
<th>Female</th>
<th>Male</th>
<th>(\leq 80)</th>
<th>(&gt;80 &amp; \leq 85)</th>
<th>&gt; 85</th>
<th>Accept</th>
<th>Reject</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule 1</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Rule 2</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Rule 3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Rule 4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Test cases would then be generated from this table, E.g.:

- TC1: female, 88 → reject (could also be male)
- TC2: male, 75 → accept (could also be female)
- TC3: male, 83 → reject
- TC4: female, 83 → accept
## Decision Table Format

<table>
<thead>
<tr>
<th>Case</th>
<th>Female</th>
<th>Male</th>
<th>&lt;= 80</th>
<th>&gt;80 &amp; &lt;= 85</th>
<th>&gt; 85</th>
<th>Accept</th>
<th>Reject</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule 1</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Rule 2</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Rule 3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Rule 4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Test cases would then be generated from this table, e.g.:
- TC1: female, 88 → reject (could also be male)
- TC2: male, 75 → accept (could also be female)
- TC3: male, 83 → reject
- TC4: female, 83 → accept

Is that all?
## Decision Table Format with invalid classes

<table>
<thead>
<tr>
<th>Case</th>
<th>Female</th>
<th>Male</th>
<th>&lt;= 80</th>
<th>&gt;80 &amp; &lt;= 85</th>
<th>&gt; 85</th>
<th>Accept</th>
<th>Reject</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rule 1</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Rule 2</td>
<td>X</td>
<td>X</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Rule 3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Rule 4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Rule 5</td>
<td>0</td>
<td>0</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Rule 6</td>
<td>X</td>
<td>X</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Rule 5: If not(F) and not(M) then not(A or R)
Rule 6: If not(<=80) and not (>80 & <=85) and not(>85) then not(A or R)
Cause-Effect Graphing vs ECP

- Although I introduced Cause-Effect-Graphing starting out from an ECP example, it is actually a very different approach.
- For doing Cause-Effect-Graphing it is not necessary to think in terms of ECs.
- It suffices to identify input conditions (causes) and their relation to output conditions (effects).
## Cause-Effect Graphing for the Triangle Problem

### Triangle Problem

Input conditions (causes) and Output conditions (effect).

### Causes:

<table>
<thead>
<tr>
<th>Cause</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>Side “x” is less than sum of “y” and “z”</td>
</tr>
<tr>
<td>C2</td>
<td>Side “y” is less than sum of “x” and “z”</td>
</tr>
<tr>
<td>C3</td>
<td>Side “z” is less than sum of “x” and “y”</td>
</tr>
<tr>
<td>C4</td>
<td>Side “x” is equal to side “y”</td>
</tr>
<tr>
<td>C5</td>
<td>Side “x” is equal to side “z”</td>
</tr>
<tr>
<td>C6</td>
<td>Side “y” is equal to side “z”</td>
</tr>
<tr>
<td>C7</td>
<td>…</td>
</tr>
</tbody>
</table>

### Effects:

<table>
<thead>
<tr>
<th>Effect</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>e1</td>
<td>Not a triangle</td>
</tr>
<tr>
<td>e2</td>
<td>Scalene triangle</td>
</tr>
<tr>
<td>e3</td>
<td>Isosceles triangle</td>
</tr>
<tr>
<td>e4</td>
<td>Equilateral triangle</td>
</tr>
<tr>
<td>e5</td>
<td>…</td>
</tr>
</tbody>
</table>

Note: ‘C’ stands for ‘Condition’ (and not for ‘Class’ as in ECP)
Cause-Effect Graphing for the Triangle Problem

Triangle Problem

Cause-effect rules:
R1: If (C1 or C2 or C3) then e1
R2: If not(C1 or C2 or C3) and not(C4 or C5 or C6) then e2
R3: If (C4 and C5) and not(C6) then e3

... R6: If (C4 and C5) or (C4 and C6) or (C5 and C6) then e4

Causes:
C1: Side “x” is less than sum of “y” and “z”
C2: Side “y” is less than sum of “x” and “z”
C3: Side “z” is less than sum of “x” and “y”
C4: Side “x” is equal to side “y”
C5: Side “x” is equal to side “z”
C6: Side “y” is equal to side “z”
C7: ...

Effects:
e1: Not a triangle
e2: Scalene triangle
e3: Isosceles triangle
e4: Equilateral triangle
e5: ...
Black-Box Testing Techniques

- Equivalence class partitioning (ECP)
- Boundary value analysis (BVA)
- Cause-effect graphing
- **Combinatorial testing**
- State transition testing (State-based testing)
- Metamorphic testing
- Random testing
- Exploratory testing
- Usability testing, A/B testing (UX)
Combinatorial Designs

- ECP and BVA define test cases per class
- Combinations of equivalence classes need to be handled
- Combinatorial explosion needs to be handled
Combinatorial Testing – Example

2 input variables

Age: 4 radio buttons + empty = 5 ECs
Gender: 2 radio buttons + empty = 3 ECs

Minimum coverage of all ECs:
6 test cases
(picked one value from each EC and combined each invalid input only with valid inputs)

All input combinations?

Result text in {<empty>, accept, reject}
Message text in {<empty>, missing input}
Combinatorial Testing – Example

2 input variables

Age: in [18, 80] ? O
in (80, 85] ? O
in (85, 99] ? O
other ? O

Gender: male O
female O

Minimum coverage of all ECs:

6 test cases
(picked one value from each EC and combined each invalid input only with valid inputs)

All input combinations?

5 x 3 = 15 test cases
(one value from each EC combined in all possible ways)
How Do We Test This?

34 switches (boolean) \( \Rightarrow 2^{34} = 1.7 \times 10^{10} \) possible inputs \( \Rightarrow 1.7 \times 10^{10} \) tests
How Do We Test This?

34 switches (boolean) => $2^{34} = 1.7 \times 10^{10}$ possible inputs => 1.7 x $10^{10}$ tests

How many tests needed to cover all 1-way interactions?
How Do We Test This?

34 switches = \(2^{34}\)
= \(1.7 \times 10^{10}\) possible inputs
= \(1.7 \times 10^{10}\) tests
(34-way interaction)

1-way interactions:
= each switch at least once ‘on’ and once ‘off’
= 2 tests:
0000000000000000000000000000000000
1111111111111111111111111111111111

... 
3-way interactions: need only 33 tests
4-way interactions: need only 85 tests

This covers all 68 ECs, assuming two ECs ({true} and {false}) per variable.

Problem: How do we know this?
How Do We Test This?

34 switches = $2^{34}$

= $1.7 \times 10^{10}$ possible inputs

= $1.7 \times 10^{10}$ tests

(34-way interaction)

1-way interactions:

= each switch at least once ‘on’ and once ‘off’

= 2 tests: 0000000000000000000000000000000000

1111111111111111111111111111111111

...  

3-way interactions: need only 33 tests

4-way interactions: need only 85 tests

Problem: How do we know this?

2-way interactions: How many tests needed?

This covers all 68 ECs, assuming two ECs ({true} and {false}) per variable.
How does it work?

2 switches = 2 boolean variables (‘0’ or ‘1’):

- all combinations: 
  2 x 2 = 4 tests

3 switches:

- all combinations: 
  $2^3 = 8$ tests

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
How does it work?

2 switches = 2 boolean variables (‘0’ or ‘1’):

- all combinations:
  - 2 x 2 = 4 tests

3 switches:

- all combinations:
  - $2^3 = 8$ tests

- all 2-way interactions:
  - $C(3, 2) * 2^2 = 3 * 4 = 12$

4 tests
10 Booleans: 2-Way Interactions?

Note: In this example we make a simplification. We assume that there are no constraints on combinations. In reality, however, there are contraints. For example, 'Strikethrough' and 'Double strikethrough' cannot go together. We ignore this in the following. In order to treat this correctly, one could re-formulate the problem by merging two binary variables into one tertiary variable, e.g., 'Strikethrough' could have three mutually exclusive values: no, yes, yes-double.

0 = effect off
1 = effect on

\[ 2^{10} = 1,024 \text{ tests for all 10-way interactions} \]
10 Booleans: 2-Way Interactions?

How many tests needed to cover all 2-way interactions?

0 = effect off
1 = effect on

$2^{10} = 1,024$ tests for all 10-way interactions
### 10 Booleans: 2-Way Interactions?

<table>
<thead>
<tr>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>x</td>
</tr>
</tbody>
</table>

| 0 = effect off  
1 = effect on |

#### 7 (6?) tests for all 2-way interactions

$2^{10} = 1,024$ tests for all 10-way interactions
10 Booleans: 3-Way Interactions?

How many tests needed to cover all 3-way interactions?

0 = effect off
1 = effect on

$2^{10} = 1,024$ tests for all 10-way interactions
10 Booleans: 3-Way Interactions?

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

0 = effect off
1 = effect on

13 tests for all 3-way interactions

$2^{10} = 1,024$ tests for all 10-way interactions
### 10 Booleans: 3-Way Interactions?

All 3-way interactions between the first 3 variables:

<p>| | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

0 = effect off  
1 = effect on  

13 tests for **all 3-way interactions**

\[ 2^{10} = 1,024 \] tests for **all 10-way interactions**
Combinatorial Testing

What is it?

• Methods for systematically testing t-way interaction effects of input (or configuration parameter) values.

Why do it?

• The interaction of specific combinations of input values may trigger failures that won’t be triggered if testing input values (or configurations) only in isolation.
Is Testing 2-Way Interactions Enough?

Analyses of failure-triggering conditions showed this:

- Medical device (dark blue)
- NASA distrib. DB (light blue)
- Browser (green)
- Web server (magenta)
- Network security (orange)
- TCAS* module (purple)

* Traffic Collision Avoiding System

Several studies have shown that Pair-wise Testing triggers between 50% and 90% of all failures.

More strengths than 6-way interaction has hardly ever shown to find more defects.
Two Scopes of Combinatorial Testing

Test Configurations

App 1: Pizza Service

<table>
<thead>
<tr>
<th>Test Inputs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Size</strong></td>
</tr>
<tr>
<td>Sm</td>
</tr>
<tr>
<td>Sm</td>
</tr>
<tr>
<td>Med</td>
</tr>
<tr>
<td>Med</td>
</tr>
<tr>
<td>Lg</td>
</tr>
<tr>
<td>Lg</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test case</th>
<th>OS</th>
<th>CPU</th>
<th>Protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Windows</td>
<td>Intel</td>
<td>IPv4</td>
</tr>
<tr>
<td>2</td>
<td>Windows</td>
<td>AMD</td>
<td>IPv6</td>
</tr>
<tr>
<td>3</td>
<td>Linux</td>
<td>Intel</td>
<td>IPv6</td>
</tr>
<tr>
<td>4</td>
<td>Linux</td>
<td>AMD</td>
<td>IPv4</td>
</tr>
</tbody>
</table>

Pizza Ordering
System under test (SUT)
Combinatorial Testing – Test Inputs

Many variables
Many values per variable
Need to abstract values (equivalence classes, boundary values)

Plan: flt, flt+hotel, flt+hotel+car
From: CONUS, HI, AK, Europe, Asia …
To: CONUS, HI, AK, Europe, Asia …
Compare: yes, no
Date-type: exact, 1to3, flex
Depart: today, tomorrow, 1yr, Sun, Mon …
Return: today, tomorrow, 1yr, Sun, Mon …
Adults: 1, 2, 3, 4, 5, 6
Minors: 0, 1, 2, 3, 4, 5
Seniors: 0, 1, 2, 3, 4, 5

CONUS = Continental US
HI = Hawaii
AK = Alaska
Combinatorial Testing – Test Configuration

Platform configuration parameters:

- OS: Windows XP, Apple OS X, Red Hat Linux
- Browser: Internet Explorer, Firefox
- Protocol: IPv4, IPv6
- CPU: Intel, AMD
- DBMS: MySQL, Sybase, Oracle

Total number of combinations: $3 \times 2 \times 2 \times 2 \times 3 = 72$

Do we really need to test all 72 combinations?
Pair-Wise Testing (2-Way Interaction)

# of pairs:
\[
\binom{5}{2} = \frac{5!}{2!(5-2)!} = 10
\]

# of 2-way interactions is
\[
40 = \left(\binom{5}{2}\right)^2 \leq N \leq \left(\binom{5}{2}\right)^3 = 90
\]

a) OS [3]: Windows XP, Apple OS X, Red Hat Linux
b) Browser [2]: Internet Explorer, Firefox
c) Protocol [2]: IPv4, IPv6
d) CPU [2]: Intel, AMD
e) DBMS [3]: MySQL, Sybase, Oracle
Pair-Wise Testing (2-Way Interaction)

# of pairs:

\[
\binom{5}{2} = \frac{5!}{2!(5-2)!} = 10
\]

# of 2-way interactions is

N=57:

\[
40 = \binom{5}{2}2^2 \leq N \leq \binom{5}{2}3^2 = 90
\]

<table>
<thead>
<tr>
<th>Pair</th>
<th>Multiplication</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>a:b</td>
<td>3 x 2</td>
<td>6</td>
</tr>
<tr>
<td>a:c</td>
<td>3 x 2</td>
<td>6</td>
</tr>
<tr>
<td>a:d</td>
<td>3 x 2</td>
<td>6</td>
</tr>
<tr>
<td>a:e</td>
<td>3 x 3</td>
<td>9</td>
</tr>
<tr>
<td>b:c</td>
<td>2 x 2</td>
<td>4</td>
</tr>
<tr>
<td>b:d</td>
<td>2 x 2</td>
<td>4</td>
</tr>
<tr>
<td>b:e</td>
<td>2 x 3</td>
<td>6</td>
</tr>
<tr>
<td>c:d</td>
<td>2 x 2</td>
<td>4</td>
</tr>
<tr>
<td>c:e</td>
<td>2 x 3</td>
<td>6</td>
</tr>
<tr>
<td>d:e</td>
<td>2 x 3</td>
<td>6</td>
</tr>
</tbody>
</table>

Total pairs: 57

- OS [3]: Windows XP, Apple OS X, Red Hat Linux
- Browser [2]: Internet Explorer, Firefox
- Protocol [2]: IPv4, IPv6
- CPU [2]: Intel, AMD
- DBMS [3]: MySQL, Sybase, Oracle
Pair-Wise Testing (2-Way Interaction)

# of pairs:

\[
\binom{5}{2} = \frac{5!}{2!(5-2)!} = 10
\]

# of 2-way interactions is N=57:

\[
40 = \binom{5}{2}2^2 \leq N \leq \binom{5}{2}3^2 = 90
\]

\[
\begin{array}{c|c}
\text{a:b} & 3 \times 2 = 6 \\
\text{a:c} & 3 \times 2 = 6 \\
\text{a:d} & 3 \times 2 = 6 \\
\text{a:e} & 3 \times 3 = 9 \\
\text{b:c} & 2 \times 2 = 4 \\
\text{b:d} & 2 \times 2 = 4 \\
\text{b:e} & 2 \times 3 = 6 \\
\text{c:d} & 2 \times 2 = 4 \\
\text{c:e} & 2 \times 3 = 6 \\
\text{d:e} & 2 \times 3 = 6 \\
\end{array}
\]

\[
\sum = 57
\]

What is the lower bound for the number of test cases?

a) OS [3]: Windows XP, Apple OS X, Red Hat Linux
b) Browser [2]: Internet Explorer, Firefox
c) Protocol [2]: IPv4, IPv6
d) CPU [2]: Intel, AMD
e) DBMS [3]: MySQL, Sybase, Oracle
Pair-Wise Testing (2-Way Interaction)

Only 9 tests needed, if we want to test all interactions of one parameter with one other parameter (pair-wise interaction).

<table>
<thead>
<tr>
<th>Test</th>
<th>OS (a)</th>
<th>Browser (b)</th>
<th>Protocol (c)</th>
<th>CPU (d)</th>
<th>DBMS (e)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>XP</td>
<td>Firefox</td>
<td>IPv6</td>
<td>AMD</td>
<td>MySQL</td>
</tr>
<tr>
<td>2</td>
<td>XP</td>
<td>IE</td>
<td>IPv4</td>
<td>Intel</td>
<td>Sybase</td>
</tr>
<tr>
<td>3</td>
<td>XP</td>
<td>Firefox</td>
<td>IPv4</td>
<td>AMD</td>
<td>Oracle</td>
</tr>
<tr>
<td>4</td>
<td>OS X</td>
<td>IE</td>
<td>IPv6</td>
<td>Intel</td>
<td>MySQL</td>
</tr>
<tr>
<td>5</td>
<td>OS X</td>
<td>Firefox</td>
<td>IPv4</td>
<td>AMD</td>
<td>Sybase</td>
</tr>
<tr>
<td>6</td>
<td>OS X</td>
<td>IE</td>
<td>IPv6</td>
<td>Intel</td>
<td>Oracle</td>
</tr>
<tr>
<td>7</td>
<td>RHL</td>
<td>Firefox</td>
<td>IPv4</td>
<td>Intel</td>
<td>MySQL</td>
</tr>
<tr>
<td>8</td>
<td>RHL</td>
<td>IE</td>
<td>IPv6</td>
<td>AMD</td>
<td>Sybase</td>
</tr>
<tr>
<td>9</td>
<td>RHL</td>
<td>Firefox</td>
<td>IPv6</td>
<td>Intel</td>
<td>Oracle</td>
</tr>
</tbody>
</table>

NOTE: In this case, the lower bound is the same as what is actually needed for covering all 2-way interactions. But that is just a coincidence!
Pair-Wise Testing (2-Way Interaction)

Only 9 tests needed, if we want to test all interactions of one parameter with one other parameter (pair-wise interaction).

<table>
<thead>
<tr>
<th>Test</th>
<th>OS (a)</th>
<th>Browser (b)</th>
<th>Protocol (c)</th>
<th>CPU (d)</th>
<th>DBMS (e)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>XP</td>
<td>Firefox</td>
<td>IPv6</td>
<td>AMD</td>
<td>MySQL</td>
</tr>
<tr>
<td>2</td>
<td>XP</td>
<td>IE</td>
<td>IPv4</td>
<td>Intel</td>
<td>Sybase</td>
</tr>
<tr>
<td>3</td>
<td>XP</td>
<td>Firefox</td>
<td>IPv4</td>
<td>AMD</td>
<td>Oracle</td>
</tr>
<tr>
<td>4</td>
<td>OS X</td>
<td>IE</td>
<td>IPv6</td>
<td>Intel</td>
<td>MySQL</td>
</tr>
<tr>
<td>5</td>
<td>OS X</td>
<td>Firefox</td>
<td>IPv4</td>
<td>AMD</td>
<td>Sybase</td>
</tr>
<tr>
<td>6</td>
<td>OS X</td>
<td>IE</td>
<td>IPv6</td>
<td>Intel</td>
<td>Oracle</td>
</tr>
<tr>
<td>7</td>
<td>RHL</td>
<td>Firefox</td>
<td>IPv4</td>
<td>Intel</td>
<td>MySQL</td>
</tr>
<tr>
<td>8</td>
<td>RHL</td>
<td>IE</td>
<td>IPv6</td>
<td>AMD</td>
<td>Sybase</td>
</tr>
<tr>
<td>9</td>
<td>RHL</td>
<td>Firefox</td>
<td>IPv6</td>
<td>Intel</td>
<td>Oracle</td>
</tr>
</tbody>
</table>

Pair (a, b):
3 x 2 = 6 Combinations
Pair-Wise Testing (2-Way Interaction)

Only 9 tests needed, if we want to test all interactions of one parameter with one other parameter (pair-wise interaction)

Pair (b, c):
2 x 2 = 4 Combinations

<table>
<thead>
<tr>
<th>Test</th>
<th>OS (a)</th>
<th>Browser (b)</th>
<th>Protocol (c)</th>
<th>CPU (d)</th>
<th>DBMS (e)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>XP</td>
<td>Firefox</td>
<td>IPv6</td>
<td>AMD</td>
<td>MySQL</td>
</tr>
<tr>
<td>2</td>
<td>XP</td>
<td>IE</td>
<td>IPv4</td>
<td>Intel</td>
<td>Sybase</td>
</tr>
<tr>
<td>3</td>
<td>XP</td>
<td>Firefox</td>
<td>IPv4</td>
<td>AMD</td>
<td>Oracle</td>
</tr>
<tr>
<td>4</td>
<td>OS X</td>
<td>IE</td>
<td>IPv6</td>
<td>Intel</td>
<td>MySQL</td>
</tr>
<tr>
<td>5</td>
<td>OS X</td>
<td>Firefox</td>
<td>IPv4</td>
<td>AMD</td>
<td>Sybase</td>
</tr>
<tr>
<td>6</td>
<td>OS X</td>
<td>IE</td>
<td>IPv6</td>
<td>Intel</td>
<td>Oracle</td>
</tr>
<tr>
<td>7</td>
<td>RHL</td>
<td>Firefox</td>
<td>IPv4</td>
<td>Intel</td>
<td>MySQL</td>
</tr>
<tr>
<td>8</td>
<td>RHL</td>
<td>IE</td>
<td>IPv6</td>
<td>AMD</td>
<td>Sybase</td>
</tr>
<tr>
<td>9</td>
<td>RHL</td>
<td>Firefox</td>
<td>IPv6</td>
<td>Intel</td>
<td>Oracle</td>
</tr>
</tbody>
</table>
ACTS Tool (NIST & UT Arlington)
ACTS – Example 1

3 variables: VAR1, VAR2, VAR3
2 values per variable: 0, 1
⇒
4 TCs
ACTS – Example 2

5 variables: VAR1 … VAR 5
3 values per variable: 0, 1, 2
→ 15 TCs
### ACTS – Example 3

<table>
<thead>
<tr>
<th></th>
<th>OS</th>
<th>BROWSER</th>
<th>PROTOCOL</th>
<th>CPU</th>
<th>DEMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Windows XP</td>
<td>Firefox</td>
<td>IPv6</td>
<td>AMD</td>
<td>MySQL</td>
</tr>
<tr>
<td>2</td>
<td>Windows XP</td>
<td>Internet Explorer</td>
<td>IPv4</td>
<td>Intel</td>
<td>Sybase</td>
</tr>
<tr>
<td>3</td>
<td>Windows XP</td>
<td>Firefox</td>
<td>IPv4</td>
<td>AMD</td>
<td>Oracle</td>
</tr>
<tr>
<td>4</td>
<td>Apple OS X</td>
<td>Internet Explorer</td>
<td>IPv6</td>
<td>Intel</td>
<td>MySQL</td>
</tr>
<tr>
<td>5</td>
<td>Apple OS X</td>
<td>Firefox</td>
<td>IPv6</td>
<td>Intel</td>
<td>Sybase</td>
</tr>
<tr>
<td>6</td>
<td>Apple OS X</td>
<td>Internet Explorer</td>
<td>IPv4</td>
<td>AMD</td>
<td>Oracle</td>
</tr>
<tr>
<td>7</td>
<td>Red Hat Linux</td>
<td>Firefox</td>
<td>IPv4</td>
<td>Intel</td>
<td>MySQL</td>
</tr>
<tr>
<td>8</td>
<td>Red Hat Linux</td>
<td>Internet Explorer</td>
<td>IPv6</td>
<td>AMD</td>
<td>Sybase</td>
</tr>
<tr>
<td>9</td>
<td>Red Hat Linux</td>
<td>Firefox</td>
<td>IPv6</td>
<td>Intel</td>
<td>Oracle</td>
</tr>
</tbody>
</table>

- 5 variables
- 2 or 3 values per variable

⇒ 9 TCs
Defining Variables of SUT

12 variables
Two to many values per variable
Variable Interaction Strength

Define interaction strength for all variables or for subsets of variables.
Output: Covering Array

Variables and their values
Combinatorial Testing Link

http://csrc.nist.gov/groups/SNS/acts/index.html
How to (Automatically) Generate Test Oracles?

Creating test (input) data is the (relatively) easy part!

How do we check that the code worked correctly on the test input?

- **Using existing test** sets (with known oracles) – easy if test sets exist
- **Crash testing** code to ensure it does not crash for randomly generated test input (‘fuzz testing’) – easy but of limited value
- **Embedded assertions** – incorporate assertions in code to check critical states at different points in the code – will they cover all possible incorrect states?
- **Exploiting domain knowledge** – NLP on spec docs, metamorphic testing, …
- **Model-checking** using a mathematical model of the system under test and a model checker to generate expected results for each input – expensive but tractable
Combinatorial Testing – Summary

Combinatorial testing makes sense where

- More than ~8 variables and less than 300-400
- Logical or numeric interaction of variables

New algorithms make large-scale combinatorial testing possible

Tool-support exists (research prototypes)

Model-checking facilitates automatic test oracle generation
HW 3: Combinatorial Testing

Task 1

1. Create system of Booking app in ACTS tool
2. Generate covering arrays for all strengths
3. Run Booking app with test data and generate output files
4. Inspect output for crashes and other failures and create tables and graphs with data for each run (strengths 1 to n – using 2 algorithms)
HW 3: Combinatorial Testing

Task 2

1. Create system of Booking app in ACTS tool; setting parameters such that covering arrays have not more than 250 tests; use information from Task 1 as well as the given rules
2. Generate covering arrays for all strengths
3. Run Booking app with test data and generate output files
4. Inspect output and count unique failures

Repeat until maximum amount of unique failures found.
HW 3: Combinatorial Testing

Task 3

1. Analyze the output from Task 1 and try to find failures that are not reported through error messages from the application
   - look only at the output generated with maximum strength interaction (full combinatorial testing)
   - Use the rule set to spot incorrect outputs
To Do & Next Weeks

• Quiz 3 (in Moodle!):
  • Opens today – closes on Monday at 17:30am!

• Lab 2:
  – Finish and submit HW2 (Basic Black-Box-Testing)

• Lab 3:
  – Combinatorial Testing (HW3) Next Week!

• Lecture 4:
  – Basic White-Box Testing Techniques: Control-Flow Coverage