Combinatorial Analysis – Beyond Pair-wise Testing

Bj Rollison
Test Architect
Microsoft

http://www.TestingMentor.com
http://blogs.msdn.com/imtesty
Parameters with multiple interdependent variable states cause a significant number of issues!

Imagine...

- 24 parameters
- 2 – 5 variable states per parameter
- > 500,000,000,000 combinations
- 1 test/ms > 3300 years
Software testing

Software testing is...

- Any activity designed to evaluate an attribute or capability of a software program to determine that it meets applicable standards or guidelines.

The primary objectives include:

- Provide information
- Measure quality

Approaches to functional testing

- Verification/scripted testing
- Exploratory/ad hoc testing
- In-depth systematic analysis
Testing challenge

- Design an extremely small subset of tests from all possibilities
- Evaluate important attributes and capabilities of the application under test
- Identify issues and other potential risks
- Exercise large portions of the code to reduce risk
- Provide accurate information to the decision makers for informed risk-assessment
- Within a limited amount of time
- Provides a high degree of confidence for the entire team
Why combinatorial analysis

- **Functional testing technique** in which the tester:
  - systematically analyze parameter interaction in a complex feature set
  - in order to **methodically select** a finite subset of **tests** from all possible combinations

- Uses a **mathematical approach** based on historical **failure indicators** and **empirical fault models**

- Helps **reduce overall risk** by
  - High defect detection effectiveness
  - High levels of structural coverage
Combinatorial testing is useful when testing a feature in which:

- **Parameters** are directly interdependent
- **Parameters** are semi-coupled
- Parameter **input** is unordered
Combinatorial test approaches

- Random evaluation approaches
  - Best guess
    - Intuition and luck
  - Random selection
    - Randomly selected from all combinations
- Systematic evaluation approaches
  - Each choice (EC)
  - Base choice (BC)
  - Orthogonal arrays (OA)
  - Coverage arrays (CA)
  - Exhaustive (AC)
Orthogonal arrays (OA)

- Finding **OA is non-trivial**
- Unrealistic expectations equal number of variables per parameter
- **Redundant tuples** in output

L₈ Orthogonal Array

<table>
<thead>
<tr>
<th>Test</th>
<th>B</th>
<th>I</th>
<th>U</th>
<th>S</th>
<th>Col.5</th>
<th>Col.6</th>
<th>Col.7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>unck</td>
<td>unck</td>
<td>unck</td>
<td>unck</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>unck</td>
<td>unck</td>
<td>unck</td>
<td>chk</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>unck</td>
<td>chk</td>
<td>chk</td>
<td>unck</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>unck</td>
<td>chk</td>
<td>chk</td>
<td>chk</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>chk</td>
<td>unck</td>
<td>chk</td>
<td>unck</td>
<td>2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>chk</td>
<td>unck</td>
<td>chk</td>
<td>chk</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>chk</td>
<td>chk</td>
<td>unck</td>
<td>unck</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>chk</td>
<td>chk</td>
<td>unck</td>
<td>chk</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

Taguchi Array Selector
http://www.freequality.org
**Coverage arrays**

- **Pairs of parameters** (BI, BU, BS, IU, IS, US)
- **Every variable combination** for each pair of parameters
- **Multiple pair combinations per test**
- **Minimizes redundancy**

<table>
<thead>
<tr>
<th>Test</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
<th>16</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bold</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Italic</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Underline</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Strikethrough</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
</tbody>
</table>
Benefits

FREE SEX
no strings attached

Greatest number (> 50%) of combinatorial defects are simple pair interactions

- Randomizing pair-wise combinations with PICT may improve \( t=2 \) detection effectiveness*

Subtle defects still being discovered at \( t=6 \)

- Effects are cumulative

Number of tests increase exponentially, but don’t stop at pair-wise (\( t=2 \))
Defect detection effectiveness

- Pair-wise found 98 percent of all detectable faults seeded in five applications
  - All faults in four applications

<table>
<thead>
<tr>
<th>Test Subject</th>
<th>Faults</th>
<th>Combination Strategy</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>known</td>
<td>detectable</td>
</tr>
<tr>
<td>count</td>
<td>15</td>
<td>12</td>
</tr>
<tr>
<td>tokens</td>
<td>15</td>
<td>11</td>
</tr>
<tr>
<td>series</td>
<td>20</td>
<td>19</td>
</tr>
<tr>
<td>nametbl</td>
<td>49</td>
<td>49</td>
</tr>
<tr>
<td>ntree</td>
<td>29</td>
<td>29</td>
</tr>
<tr>
<td>total</td>
<td>128</td>
<td>120</td>
</tr>
<tr>
<td>% of detectable</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 10: Number of faults revealed by the combination strategies.

Evaluation of Combination Strategies for Test Case Selection (Grindal, et. al.)
“In this study we found no significant difference in the FDE of n-way and random combinatorial test suites. …the result is not unexpected.”

Tests selected randomly from all possible combinations

# of tests equal to n-way # of tests

(Patrick J. Schroeder, Pankaj Bolaki, and Vijayram Gopu)
"… a set of 29 pairwise tests gave 90% block coverage for the UNIX sort command. We also compared pairwise testing with random input testing and found that pairwise testing gave better coverage."

Table 9
Code Coverage Results for Module A

<table>
<thead>
<tr>
<th>Method</th>
<th>No of tests</th>
<th>Block</th>
<th>Decision</th>
<th>P-uses</th>
<th>C-uses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pair-wise</td>
<td>200</td>
<td>92</td>
<td>85</td>
<td>49</td>
<td>72</td>
</tr>
<tr>
<td>All</td>
<td>436</td>
<td>92</td>
<td>85</td>
<td>49</td>
<td>72</td>
</tr>
<tr>
<td>Random</td>
<td>300</td>
<td>67</td>
<td>58</td>
<td>36</td>
<td>55</td>
</tr>
</tbody>
</table>

The AETG System: An Approach to Testing Based on Combinatorial Design (Cohen, et. al.)
## Coverage effectiveness

### Attrib.exe – path + 6 optional argv.

<table>
<thead>
<tr>
<th>Total # of blocks = 483</th>
<th>&quot;Default&quot; test suite</th>
<th>Exhaustive coverage</th>
<th>Pair-wise coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of test cases</td>
<td>9</td>
<td>972</td>
<td>13</td>
</tr>
<tr>
<td>Blocks covered</td>
<td>358</td>
<td>370</td>
<td>370</td>
</tr>
<tr>
<td>Code coverage</td>
<td>74%</td>
<td>77%</td>
<td>77%</td>
</tr>
<tr>
<td>Functions not covered</td>
<td>15</td>
<td>15</td>
<td>15</td>
</tr>
</tbody>
</table>

### Findstr.exe – str + 19 optional argv. (421,200 EC tests)

<table>
<thead>
<tr>
<th>Total # of blocks = 1317</th>
<th>Hand-crafted tests</th>
<th>N=2 coverage</th>
<th>N=3 coverage</th>
<th>N=4 coverage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of test cases</td>
<td>236</td>
<td>136</td>
<td>800</td>
<td>3533</td>
</tr>
<tr>
<td>Blocks covered</td>
<td>960</td>
<td>979</td>
<td>994</td>
<td>1006</td>
</tr>
<tr>
<td>Code coverage</td>
<td>73%</td>
<td>74%</td>
<td>75%</td>
<td>76%</td>
</tr>
<tr>
<td>Functions not covered</td>
<td>11</td>
<td>11</td>
<td>10</td>
<td>10</td>
</tr>
</tbody>
</table>
Efficiency and cost savings

- Regression test suite time reduced (104 man-days to 32 hours)
- Test design and implementation reduced (44 man-months to 4 man-months)
- Reduced OS configuration testing 50% with zero bug bounce (ZBB)
- Increase in automated test development adopting data-driven output of PICT (@ 30 tests/week to 235 tests/week)
- 3 ‘show-stopper’ defects detected prior to release saved @ $375,000 (each QFE @ $125K)
Common problems...

- Misuse of technique (wrong fault model)
- Unfamiliarity with technique
- Right values not tested
- Highly probable values get too little attention
- Complex interactions are missed
Misuse of the technique

Sometimes we just can’t stop stupid people from doing stupid things!

But, if you understand the fault model…
Not combinatorial testing

- The combinatorial fault model does not apply if...
  - Parameters are independent

- Mathematical formulas
  - \( \text{Var}_{\text{Param1}} + \text{Var}_{\text{Param2}} = \text{Var}_{\text{ParamOut}} \)

- Sequential operations
- Ordered input
Unfamiliarity with technique / tool
Feature decomposition

- Interdependent parameters
  - Fonts
  - Bold
  - Italic

- Semi-coupled parameters
  - Color (*Black, white, red, green, blue, yellow*)
  - Size (*1 – 1638, including half sizes*)
  - Strikethrough
  - Underline

*Total number of tests = Cartesian product of variables = 1,257,600 tests (assuming all font size values are tested)*
Model the parameter variables

- **File format** (for PICT)
  - *param*: \( \text{var}_1, \text{var}_2, \text{var}_3 \)
- Use equivalence classes!

---

File content:

**Font:** Arial, Tahoma, BrushScript, MonotypeCorsive
**Bold:** Check, Uncheck
**Italic:** Check, Uncheck
**Strikethrough:** Check, Uncheck
**Underline:** Check, Uncheck
**Colors:** Black, White, Red, Green, Blue, Yellow
**Size:** Small, Nominal, Large, VeryLarge, HalfSizes
Right values not tested

YOU'RE DOING IT WRONG
Verify Output

- Identify mutually exclusive variable combinations
- Analyze output for “happy path”
- Analyze output for failure indicators
Modify model

- Conditional or invariant relationships

Exclude mutually exclusive variable states; often increases total number of tests
Highly probable values get too little attention
Weighting variables, Variable preference

Give increased weighted value to important variables; MAY increase probability of occurrence
Seeded output
- Specify important combinations
- ‘Happy path’
- BC tests

Tahoma seed not used because Tahoma is equivalent to Arial

Seeded input ensures important combinations are tested
Modify Model

- Sub-Models
  - Bundle specific parameters into groups
  - Each groups gets own combinatory orders
  - Increases thoroughness of tested combinations

- Syntax
  - \{<ParamName1>, <ParamName2>, <ParamName3>, ... \} @ <Order>
  - Combinatory order cannot exceed number of its parameters
Complex interactions missed
Randomize output to increase breadth of testing and minimize probability of error due to missed combinations.
Increase order of combinations for greater depth of testing and minimize probability of error due to complex interactions.

<table>
<thead>
<tr>
<th>Font</th>
<th>Bold</th>
<th>Italic</th>
<th>Strikethrough</th>
<th>Underline</th>
<th>Colors</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arial</td>
<td>Uncheck</td>
<td>Uncheck</td>
<td>Uncheck</td>
<td>Uncheck</td>
<td>Black</td>
<td>Nominal</td>
</tr>
<tr>
<td>BrushScript</td>
<td>Uncheck</td>
<td>Uncheck</td>
<td>Uncheck</td>
<td>Uncheck</td>
<td>Yellow</td>
<td>Large</td>
</tr>
<tr>
<td>MonotypeCors</td>
<td>Check</td>
<td>Check</td>
<td>Check</td>
<td>Check</td>
<td>Green</td>
<td>HalfSizes</td>
</tr>
<tr>
<td>MonotypeCors</td>
<td>Check</td>
<td>Check</td>
<td>Check</td>
<td>Check</td>
<td>Red</td>
<td>VeryLarge</td>
</tr>
<tr>
<td>MonotypeCors</td>
<td>Check</td>
<td>Check</td>
<td>Check</td>
<td>Check</td>
<td>White</td>
<td>Large</td>
</tr>
<tr>
<td>MonotypeCors</td>
<td>Check</td>
<td>Check</td>
<td>Check</td>
<td>Check</td>
<td>Green</td>
<td>HalfSizes</td>
</tr>
<tr>
<td>MonotypeCors</td>
<td>Check</td>
<td>Check</td>
<td>Check</td>
<td>Check</td>
<td>Red</td>
<td>Small</td>
</tr>
<tr>
<td>MonotypeCors</td>
<td>Check</td>
<td>Check</td>
<td>Check</td>
<td>Check</td>
<td>White</td>
<td>Small</td>
</tr>
<tr>
<td>MonotypeCors</td>
<td>Check</td>
<td>Check</td>
<td>Check</td>
<td>Check</td>
<td>Green</td>
<td>Large</td>
</tr>
<tr>
<td>MonotypeCors</td>
<td>Check</td>
<td>Check</td>
<td>Check</td>
<td>Check</td>
<td>Black</td>
<td>Nominal</td>
</tr>
<tr>
<td>MonotypeCors</td>
<td>Check</td>
<td>Check</td>
<td>Check</td>
<td>Check</td>
<td>Yellow</td>
<td>Small</td>
</tr>
<tr>
<td>MonotypeCors</td>
<td>Check</td>
<td>Check</td>
<td>Check</td>
<td>Check</td>
<td>Red</td>
<td>Small</td>
</tr>
<tr>
<td>MonotypeCors</td>
<td>Check</td>
<td>Check</td>
<td>Check</td>
<td>Check</td>
<td>White</td>
<td>HalfSizes</td>
</tr>
<tr>
<td>MonotypeCors</td>
<td>Check</td>
<td>Check</td>
<td>Check</td>
<td>Check</td>
<td>Yellow</td>
<td>Small</td>
</tr>
<tr>
<td>MonotypeCors</td>
<td>Check</td>
<td>Check</td>
<td>Check</td>
<td>Check</td>
<td>Red</td>
<td>Small</td>
</tr>
<tr>
<td>MonotypeCors</td>
<td>Check</td>
<td>Check</td>
<td>Check</td>
<td>Check</td>
<td>White</td>
<td>HalfSizes</td>
</tr>
</tbody>
</table>

- 2-wise = 33 tests
- 3-wise = 120 tests

PICT
/o:n
switch
Overcoming obstacles

- Misuse of technique
  - Domain knowledge beyond the user interface
- Unfamiliarity with technique or tool
  - Training, in-depth technical knowledge, use the right tool for the right job!
- Right values not tested
  - In-depth system knowledge, stochastic data
- Highly probable values get too little attention
  - Weighted variables, include seeded inputs
- Complex interactions are missed
  - Random output, increase n-wise combinations
Combinatorial analysis process with PICT

Analysis & feature decomposition → Model parameter variables

Seeded Input → PICT Tool

Modify model → Re-validate Output

Test up to t=6

REMEMBER: The output of the tool is based on the tester’s input...it’s only a tool!
Example: Setup/upgrade testing

# Machine configurations
#
Maker: Dell, HP, Sony, Compaq, Other
CPU: Pentium, PentiumPro, PII, PIII, P4, IA-64
CPU#: Single, Dual, Quad
RAM: 128MB, 256MB, 512MB, 1024MB
SvcPack: 1, 2, 3, 4
IE: IE5, IE5.5, IE6.0, IE7.0
Disk: SCSI, IDE
DualMonitor: yes, no
Example: Interoperability testing

```plaintext
# Machine 1
OS_1:  win2000, winXP
SKU_1:  Professional, Server, Datacenter, WinPowered
LANG_1:  EN, DE

# Machine 2
OS_2:  win2000, winXP
SKU_2:  Professional, Server, Datacenter
LANG_2:  EN, DE

[OS_1] <> [OS_2] and [SKU_1] <> [SKU_2] and [LANG_1] <> [LANG_2];
```
Example: Configuration testing

```
# Create and format a volume
# Focus on primary partition formatted with NTFS

TYPE: Primary (10), Logical, Single, Span, Stripe, Mirror, RAID-5
SIZE: 10, 1000, 5000, 40000
FORMAT: quick, slow
FSYSTEM: FAT, FAT32, NTFS (10)
CLUSTER: 512, 1024, 2048, 4096, 8192, 16384, 32768, 65536
COMPRESSION: on, off

# File systems have constraints on volume size
#
IF [FSYSTEM] = "FAT" THEN [SIZE] <= 4096;
IF [FSYSTEM] = "FAT32" THEN [SIZE] <= 32000;

# Compression can be applied only for volumes
# formatted as NTFS and with cluster size <= 4K
#
IF [FSYSTEM] in "FAT", "FAT32" or
   ([FSYSTEM] = "NTFS" and [CLUSTER] >4096)
THEN [COMPRESSION] = "off";
```
Example: Game testing


Player1: Aztecs, Bantu, British, Chinese, Egyptians, French, Germans, Greek, Inca, Japanese, Koreans, Maya, Mongols, Nubians, Romans, Russians, Spanish, Turks, Random

Player2: Aztecs, Bantu, British, Chinese, Egyptians, French, Germans, Greek, Inca, Japanese, Koreans, Maya, Mongols, Nubians, Romans, Russians, Spanish, Turks, Random

Player3: Aztecs, Bantu, British, Chinese, Egyptians, French, Germans, Greek, Inca, Japanese, Koreans, Maya, Mongols, Nubians, Romans, Russians, Spanish, Turks, Random

Stance: Random, Offensive, Defensive, Economic

RushRules: Standard, Classical, Medieval, Gunpowder, Enlightenment, Industrial, Modern, Information, NonViolent

Teams: Diplomacy, Survival of the Fittest, Assassin, Barbarians at the Gates, Team Diplomacy, Cooperative Diplomacy, Cooperative Survival, Cooperative Teams, Custom Diplomacy, Custom Survival

Resources: Low, Standard, Deathmatch, Infinite, VarLow, VarMedium, VarHigh, Random
Example: API testing (Black box)
Example: Utility testing

```
# Testing of findstr.exe

PLACES_IN_FILE_ARG: /B, /E,
TEXT_TYPE_ARG: /L, /R,
DIRECTORY_ARG: /D:1;2, /S,
CASE_SENS_ARG: /I,
LINES_MATCHING_ARG: /V, /X,
PRINT_ARG: /N, /M, /O, /A:07,
FILE_TYPE_ARG: /F:index.txt, /G:normal_onlyinmiddle.txt, /OFF, /P,
WHERE: */*, *.txt, *.bin, *.log
TEXT: A.*, AA.*, Aaa.*, *.B.[1-6]*.*,.*, ^A, ^A.*, $123, \<Abc, \<ABC, 23>, f123
```
Example: Compiler testing

```plaintext
NestedTypeKind: Interface, Class, Delegate, Struct, Enum, Primitive, Unmanaged
NestedTypeFlag: abstract, sealed, None
NestingModuleFlag: Mixed, PureManaged
ClientInteraction: ImplementsNested, ImplementsBoth, ExtendsNested, ExtendsEnclImplementsNested
NestDependsOnEncl: None, Implements, StaticField, InstanceField, MethodReturn, MethodParam, Encl
EnclDependsOnNest: None, StaticField, MethodReturn, MethodParam, Implements, Extends, Instance
EntryPointLocation: NestedType, EnclosingType, MainManagedType|GlobalManaged, GlobalUnmanaged
NestingLocation: DLL, Netmodule, MainModule
ClientModuleMixed: Separate, Same
ClientLocation: DLL, Netmodule, MainModule
ModuleConfig: Separate, Same
EnclosingTypeGeneric: Generic, Nongeneric
NestedTypeGeneric: Generic, Nongeneric
EnclosingTypeVis: public, private
NestedTypeVis: public, family, assembly, famandassem, famorassem
NestMemberVis: private, famandassem, family, famorassem, public
EnclosingTypeKind: Interface, Class, Delegate, Struct, Enum, Primitive, Unmanaged
EnclosingTypeFlag: abstract, sealed, None
EncMemberVis: private, famandassem, assembly, family, famorassem, public

if [NestedTypeKind] = "Interface" then [NestedTypeFlag] = "abstract";
if [EnclosingTypeKind] = "Interface" then [EnclosingTypeFlag] = "abstract";
```
Combinatorial analysis

- **Best Practice** compared to other approaches for testing complex parameter interactions
- **Provides critical information earlier**
  - High probability of early defect detection
  - High structural coverage; $t=6 \approx$ exhaustive
- **Increases** depth and breadth of **test coverage**
- **Easy** to integrate into **automation** (data-driven)
- **But,**
  - Requires in-depth technical knowledge
  - Must verify output – it’s only a tool!
  - **Don’t forget Beizer’s Pesticide Paradox!**
Questions?
References

- http://www.pairwise.org
- Pairwise Testing in the Real World: Practical Extensions to Test-Case Scenarios
  Jacek Czerwonka
- Comparing the Fault Detection Effectiveness of N-way and Random Test Suites
  Patrick J. Schroeder, Pankaj Bolaki, and Vijayram Gopu
  http://portal.acm.org/citation.cfm?doid=1159733.1159742
- Software and Hardware Testing Using Combinatorial Covering Suites
  Alan Hartman, IBM Haifa Research Laboratory
References

- Pseudo-Exhaustive Testing for Software
  D. Richard Kuhn and Vadim Okun
  (http://csrc.nist.gov/acts/PID258305.pdf)

- Failure Modes in Medical Device Software
  Dolores R. Wallace and D. Richard Kuhn
  (http://csrc.nist.gov/staff/Kuhn.final-rqse.pdf)

- The Combinatorial Design Approach to Automatic Test Generation
  (http://www.argreenhouse.com/papers/gcp/AETGissre96.shtml)

- An Evaluation of Combination Strategies for Test Case Selection
  M. Grindal, B. Lindström, J. Offutt, and S.F. Andler
  (http://www.his.se/upload/19352/EvalCombStratTechRepFinal.pdf)