Combinatorial Testing and Its Applications

W. Eric Wong
Department of Computer Science
The University of Texas at Dallas
http://www.utdallas.edu/~ewong
Speaker Biographical Sketch

- Professor & Director
  *Advanced Research for Software Testing & Quality Assurance*
  Department of Computer Science
  University of Texas at Dallas

- Guest Researcher
  Computer Security Division
  National Institute of Standards and Technology (NIST)

- Editor-in-Chief, IEEE Transactions on Reliability

- Engineer of the Year, 2014, IEEE Reliability Society

- Vice President for Publication, IEEE Reliability Society

- Founder & Steering Committee Chair of the QRS conference
  *(IEEE International Conference on Software Quality, Reliability and Security)* & IWPD *(IEEE International Workshop on Program Debugging)*
STQA: Advanced Research Center on Software Testing and Quality Assurance
Software has become fundamental to our society and our very way of life. Regardless of age, gender, occupation, or nationality, each one of us depends on software, either directly or indirectly. Yet the disappointing truth is that software is far from defect-free and that very large sums of money are spent each year only to fix and maintain defective software. According to the National Institute of Standards and Technology, software bugs cost the U.S. economy an estimated $59.5 billion annually, or about 0.6 percent of the nation’s gross domestic product (GDP). The same study also found that more than one-third of these costs, or an estimated $22.2 billion, could be eliminated by an improved testing infrastructure. Furthermore, these estimates do not take into account any potential deaths or catastrophic financial loss associated with the failure of mission critical software.

It is notable that software testing continues to be the primary approach used in practice to ensure the development of high quality software. In fact, the majority of
Why Does A System Fail?

Software Failures VS Hardware Failures
**AT&T Network Outage, January 1990 (1)**

- At 2:20 pm on January 15, 1990, only 50% of calls placed through AT&T were connected, the other half heard a prerecorded message saying, “Sorry, all circuits are busy now.”

- The network remained down until a team of 100 telephone technicians discovered and corrected the problem at 11:30 pm that night.
While (ring receive buffer | empty and side buffer | empty) {
  Initialize pointer to first message in side buffer or ring received buffer
  Get a copy of buffer
  Switch (message) {
    Case incoming message: if (sending switch = out of service)
      if (ring write buffer = empty)
        Send in service to states map manager;
      Else
        Break;
    Process incoming message, set up pointers to optional parameters
    Break;
    : 
  }
  Do optional parameter work
}
The spacecraft was launched on December 11, 1998.

Instead of 460,000 to 500,000 feet above the Martian surface, it was only ~190,000 feet due to a *programming error*.

- Imperial units (pound-seconds) were used instead of the metric units (Newtons) in the navigation software.

The spacecraft was destroyed and the total loss was about $85 million (not including the launch vehicle).
**Explosion of Ariane 5**

- On June 4, 1996, the Ariane 5 rocket exploded *37 seconds* after lift-off.

- A *software failure* occurred when a 64-bit floating point number was converted to a 16-bit signed integer
  - The input value was larger than 32,767 and outside the range that could be represented by a 16-bit signed integer.
  - The conversion failed due to an overflow.
We Agree . . . .

Some accidents are due to a combination of software & hardware failures!
Software Enabled System

- Software has been used extensively in almost every critical infrastructure (e.g., transportation, communication, finance, medicine, aeronautics, astronautics, and nuclear power generation).

- A failure will not only merely cause inconvenience, but may also introduce severe consequences including monetary loss, property damage, injury, and even human casualty.
A Simple View of Software Assurance

• Test generation & execution

• Before the observation of any failure
  – Analysis
  – Prediction
    □ Reliability
    □ Fault-prone modules
    □ Security/safety vulnerability
    □ etc.

• After a failure is observed
  – Bug fixing
  – Regression testing
  – etc.
Why Software (System) Testing?

- Well Professor Qiang Miao told us we should… Congratulations! You will probably get an A 😊
- Testing is an investment with proven returns
- Testing is as fundamental a software development activity as any, and often can drive other activities
- Testing lets you realize things about your product that you wouldn’t otherwise
- Testing can help avoid embarrassment
A Study by NIST in 2002

• Software bugs costed the U.S. economy an estimated $59.5 billion annually, or about 0.6 percentage of the nation’s gross domestic product (GDP).
  – These numbers will be much higher if measured in 2015

• More than one third of these costs, or an estimated $22.2 billion, could be eliminated by an improved testing infrastructure.
  – Did not reflect the “costs” associated with mission critical software where failure can lead to extremely costs such as loss of life or catastrophic financial and reputation loss.
Testing and debugging activities constitute one of the most expensive aspects of software development

– Often more than 50% of the cost [Hailpern & Santhanam, 2003]

- Software testing should be effective
  - Testing is not really a question of doing or not doing
    - It is about understanding
    - It is *as much an art as it is a science*
  - We need to be familiar with the software we are testing
  - We need to plan, scope and prioritize and measure
  - *We need to treat our tests as first class citizens*
  - We need to leverage automation as appropriate
  - We need to be creative!
Why Is Reliability Overestimated?

You have good confidence about your product!

Why didn’t your customers feel the same way?
Here Is the Reason

Execution ≠ Fault Detection
**Test Case Generation**

- **When to stop testing?**
  - Has a significant impact on *reliability* and . . . . .
  - What do we want to achieve?
  - Here is an ideal solution

Entire input domain  Test cases used

where x are failure causing inputs
Do You Know... 

- How many failure causing inputs?
- Which one is a failure causing input?

What should we do?
Challenges to be Overcome

- Testing individual component is not enough
- We should also focus on interactions on different components

How to do it

Combinatorial Testing
Combinatorial Testing – What Is It?

• CT is an effective test generation technique focuses on testing various combinations among different components of a system.
  – Based on combinational design – a well-found statistical technique
  – NIST (National Institute of Standards and Technology)
Do We Have Any Tool Support?

- ACTS is one of the tools which support CT
Combinatorial Testing – Example (1)

- A system with 5 input parameters and their corresponding possible values
  - Parameter A \((a_1, a_2, a_3, a_4)\)
  - Parameter B \((b_1, b_2, b_3)\)
  - Parameter C \((c_1, c_2)\)
  - Parameter D \((d_1, d_2, d_3)\)
  - Parameter E \((e_1, e_2)\)

- How many 2-way combinations?
  \[
  4 \times 3 + 4 \times 2 + 4 \times 3 + 4 \times 2 + 3 \times 2 + 3 \times 3 + 3 \times 2 + 2 \times 3 + 2 \times 2 + 3 \times 2 = 77
  \]

\[
\begin{array}{cccccc}
A \& B & A \& C & A \& D & A \& E & B \& C & B \& D & B \& E & C \& D & C \& E & D \& E \\
A = a_1, & B = b_1 & A = a_1, & B = b_2 & A = a_1, & B = b_3 & A = a_2, & B = b_1 \\
A = a_2, & B = b_2 & A = a_2, & B = b_3 & A = a_3, & B = b_1 & \ldots
\end{array}
\]
### Combinatorial Testing – Example (2)

**• How many tests are needed to cover all 77 2-way combinations?**

<table>
<thead>
<tr>
<th>A = a₁, B = b₁, C = c₂, D = d₂, E = e₂</th>
<th>A = a₁, B = b₂, C = c₁, D = d₃, E = e₁</th>
</tr>
</thead>
<tbody>
<tr>
<td>A = a₁, B = b₃, C = c₂, D = d₁, E = e₁</td>
<td>A = a₂, B = b₁, C = c₁, D = d₃, E = e₂</td>
</tr>
<tr>
<td>A = a₂, B = b₂, C = c₂, D = d₁, E = e₂</td>
<td>A = a₂, B = b₃, C = c₁, D = d₂, E = e₁</td>
</tr>
<tr>
<td>A = a₃, B = b₁, C = c₁, D = d₁, E = e₁</td>
<td>A = a₃, B = b₂, C = c₂, D = d₂, E = e₂</td>
</tr>
<tr>
<td>A = a₃, B = b₃, C = c₂, D = d₃, E = e₂</td>
<td>A = a₄, B = b₁, C = c₁, D = d₁, E = e₁</td>
</tr>
<tr>
<td>A = a₄, B = b₂, C = c₂, D = d₂, E = e₂</td>
<td>A = a₄, B = b₃, C = c₂, D = d₃, E = e₂</td>
</tr>
</tbody>
</table>

**Only Twelve!**

One test case can cover multiple 2-way combinations
Combinatorial Testing – Example (3)

• What about 3-way combinations?

\[
\begin{align*}
4 \times 3 \times 2 & + 4 \times 3 \times 3 + 4 \times 3 \times 2 + 4 \times 2 \times 3 + 4 \times 2 \times 2 + 4 \times 3 \times 2 \\
3 \times 2 \times 3 & + 3 \times 2 \times 2 + 3 \times 3 \times 2 \\
B & \& C & \& D & B & \& C & \& E & B & \& D & \& E \\
2 \times 3 \times 2 \\
C & \& D & \& E
\end{align*}
\]

208

• How many tests are needed to cover all 208 3-way combinations?
  – Only 36 test cases!
What about 4-way?
- Totally 276 4-way combinations
- 72 test cases to cover all of them

In summary
- 2-way (77 combinations covered by 12 test cases)
- 3-way (208 combinations covered by 36 test cases)
- 4-way (276 combinations covered by 72 test cases)

In practice, 2-way (pairwise) and 3-way combinatorial testing are the most frequently used and provide good results.
Two Parts of This Talk

• Application of *black-box requirement-based* combinatorial testing to two real-life industry applications
  – Results and lessons learned

• Extension of combinatorial testing to a *white-box structure-based setting*
  – Combinatorial Decision Testing
Part I

Applying Black-box Combinatorial Testing in Industrial Settings

Two Industrial Projects

Subway Control System
Test whether the dashboard correctly presents the data based on different input values

Kylin Linux
An OS developed by National University of Defense Technology since 2001
Schedule

Timeline

1/4 1/7 1/18 1/20 1/25

Arrived in Guangzhou

Finished testing Subway Control System

Finished testing Kylin Linux

Subway Control System
• Environment setup
• Requirement Analysis
• Test generation

Kylin Linux
• OS install
• Requirement Analysis
• Test generation

Combinatorial Testing and Its Applications @ 2016 PHM - Chengdu
**Effort Distribution**

**Major Steps:**
1. Environment set up
2. Requirement Analysis
3. Test case generation
4. Execution
5. Verification

- **Step 1:** 20%
- **Step 2 & 3:** 25%
- **Step 4:** 10%
- **Step 5:** 45%
Test Generation – CT + ECP + BVA (1)

- Equivalence Class Partitioning (ECP)
- Boundary Value Analysis (BVA)

An Example:

Parameter: Pressure sensor
Input domain: 0 ~ 2500kpa

Equivalence classes:
{ 0 ~ 10kpa }, {10 ~ 200kpa }, {200 ~ 2490kpa},
{2490 ~ 2500kpa}, {2500kpa+}

Possible values: 0, 10, 200, 1250, 2500, 3000

The more possible values are selected for each parameter, the more test cases will be generated

Restriction from CEPREI: Generate no more than 100 test cases!!!
Test Generation – CT + ECP + BVA (2)

Specification → Input parameters

ECP + BVA

Possible values for each parameter

Constraints between parameters and their possible values (e.g., If parameter A is greater than 0, the value of parameter B has to be less than 100)

Different input models will generate different sets of test cases.

CT → Test cases

Expert Knowledge
Case Study I:

Subway Control System
The GUI of the System (1)
### The GUI of the System (2)

<table>
<thead>
<tr>
<th>制动控制器</th>
<th>数字输入</th>
<th>数字输出</th>
</tr>
</thead>
<tbody>
<tr>
<td>紧急制动</td>
<td>中立阀(B02.03)</td>
<td>遮断阀(B02.06)</td>
</tr>
<tr>
<td>重联位</td>
<td>重联塞门(B04.21)</td>
<td>紧急制动施加</td>
</tr>
<tr>
<td>抑制位</td>
<td>停放制动压力开关(TC1)</td>
<td>紧急排风电磁阀(B02.03)</td>
</tr>
<tr>
<td>全制动</td>
<td>停放隔离塞门(T01.01)</td>
<td>联锁电磁阀(B04.17)</td>
</tr>
<tr>
<td>常用制动区</td>
<td>切换阀(B03.15)</td>
<td>联锁电磁阀(B04.30)</td>
</tr>
<tr>
<td>初制动</td>
<td>单独缓解电磁阀(B04.1)</td>
<td>Q-URG(EQ)</td>
</tr>
<tr>
<td>运转位</td>
<td>单独制动压力开关(B04.1)</td>
<td>切换阀(B03.15)</td>
</tr>
<tr>
<td></td>
<td>单独制动隔离塞门(B04.1)</td>
<td>单独缓解电磁阀(B04.1)</td>
</tr>
<tr>
<td></td>
<td>单独制动激活</td>
<td>中立阀(B02.03)</td>
</tr>
<tr>
<td></td>
<td>无火回送塞门(B03.06)</td>
<td></td>
</tr>
<tr>
<td>压强(kPa)</td>
<td>无人警惕</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>SERRAGE LKJ1</td>
<td>紧急排风气控阀(B02.02)</td>
</tr>
<tr>
<td></td>
<td>SERRAGE LKJ2</td>
<td>制动缸1隔离塞门(B05.08)</td>
</tr>
<tr>
<td></td>
<td>零速</td>
<td>制动缸2隔离塞门(B05.09)</td>
</tr>
<tr>
<td></td>
<td>遮断阀(B02.06)</td>
<td>联锁电磁阀(B04.17)</td>
</tr>
<tr>
<td></td>
<td>紧急制动</td>
<td>联锁电磁阀(B04.30)</td>
</tr>
<tr>
<td></td>
<td>紧急制动隔离塞门(B02.0)</td>
<td>转向架1电制动信号</td>
</tr>
<tr>
<td></td>
<td>紧急制动压力开关(B02.0)</td>
<td>转向架2电制动信号</td>
</tr>
<tr>
<td></td>
<td>0 km/h (来自CCU)</td>
<td>紧急制动(来自CCU)</td>
</tr>
<tr>
<td>BCU通讯</td>
<td>BCU状态</td>
<td>机械分配阀(B03.08)故障</td>
</tr>
<tr>
<td>MVB</td>
<td>错误码: 8437</td>
<td></td>
</tr>
</tbody>
</table>
A Sample Test Case

How to interpret such a monster?
Decompose the Test Case

<table>
<thead>
<tr>
<th>Byte 0 ~ 6</th>
<th>Frame header – identify which package the test case belongs to.</th>
<th>There are totally four packages.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Package 3</strong></td>
<td><strong>16 16 FF 80 00 21 03</strong></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Byte 7</th>
<th>8 Signal Lights</th>
<th>3 Signal Lights</th>
<th>A Pressure Sensor</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Frame header – identify which package the test case belongs to.</td>
<td>There are totally four packages.</td>
<td></td>
</tr>
<tr>
<td>Byte 7</td>
<td>Frame header – identify which package the test case belongs to.</td>
<td>There are totally four packages.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Frame header – identify which package the test case belongs to.</td>
<td>There are totally four packages.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Frame header – identify which package the test case belongs to.</td>
<td>There are totally four packages.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Frame header – identify which package the test case belongs to.</td>
<td>There are totally four packages.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Frame header – identify which package the test case belongs to.</td>
<td>There are totally four packages.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Frame header – identify which package the test case belongs to.</td>
<td>There are totally four packages.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Frame header – identify which package the test case belongs to.</td>
<td>There are totally four packages.</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Byte 0</th>
<th>16 16 FF 80 00 21 03</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Binary → Hexadecimal</strong></td>
<td><strong>10000000 00000010</strong></td>
</tr>
<tr>
<td><strong>80</strong></td>
<td><strong>78</strong></td>
</tr>
</tbody>
</table>

| Byte 39 ~ 41 | Check bit & end of frame – generated by a tool automatically | **7E E9 FF** |

**Decimal → Hexadecimal**

1616FF80002103800278000078E703E703E70340200440400158025802580258025802646458027EE9FF
### Four Data Packages

<table>
<thead>
<tr>
<th>Package 1</th>
<th>Package 2</th>
<th>Package 3</th>
<th>Package 4</th>
</tr>
</thead>
</table>

Instead of selecting possible values for each parameter, we combine parameters into bytes and select possible values for each byte.
### Select Possible Values for Each Byte (1)

#### Package 3 - Byte 7

<table>
<thead>
<tr>
<th>Switching valve</th>
<th>Average Tue Failure</th>
<th>Interlock Solenoid 1</th>
<th>Exhaust Solenoid 2</th>
<th>KEMER</th>
<th>Interceptor Valve</th>
<th>KMAL</th>
<th>KPCS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 0 0 0 0 0 0</td>
<td>0 1 0 0 0 0 0 0</td>
<td>0 0 1 0 0 0 0 0</td>
<td>0 0 0 1 0 0 0 0</td>
<td>0 0 0 0 1 0 0 0</td>
<td>0 0 0 0 0 1 0 0</td>
<td>0 0 0 0 0 0 1 0</td>
<td>0 0 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>

#### Binary → Hexadecimal

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Binary</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Switching Valve</td>
<td>1000 0000</td>
<td>80</td>
</tr>
<tr>
<td>Average Tue Failure</td>
<td>0100 0000</td>
<td>40</td>
</tr>
<tr>
<td>Interlock Solenoid 1</td>
<td>0010 0000</td>
<td>20</td>
</tr>
<tr>
<td>Exhaust Solenoid 2</td>
<td>0001 0000</td>
<td>10</td>
</tr>
<tr>
<td>KEMER</td>
<td>0000 1000</td>
<td>08</td>
</tr>
<tr>
<td>Interceptor Valve</td>
<td>0000 0100</td>
<td>04</td>
</tr>
<tr>
<td>KMAL</td>
<td>0000 0010</td>
<td>02</td>
</tr>
<tr>
<td>LPCS</td>
<td>0000 0001</td>
<td>01</td>
</tr>
<tr>
<td>N/A</td>
<td>0000 0000</td>
<td>00</td>
</tr>
</tbody>
</table>

Possible values for parameter “Package 3 - Byte 7”: 80; 40; 20; 10; 08; 04; 02; 01; 00
Select Possible Values for Each Byte (2)

<table>
<thead>
<tr>
<th>Package 3 - Byte 9</th>
<th>Pressure Sensor of Brake Control Module</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td></td>
</tr>
<tr>
<td>120</td>
<td></td>
</tr>
<tr>
<td>255</td>
<td></td>
</tr>
</tbody>
</table>

Possible values for parameter “Package 3 - Byte 9”: 0A; 78; FF

Decimal → Hexadecimal

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0A</td>
</tr>
<tr>
<td>120</td>
<td>78</td>
</tr>
<tr>
<td>255</td>
<td>FF</td>
</tr>
</tbody>
</table>
Expert Knowledge can also help select possible values for each byte.
### Select Possible Values for Each Byte (3)

**Package 1 - Byte 20**

<table>
<thead>
<tr>
<th></th>
<th>Tube 1</th>
<th>Tube 2</th>
<th>Cylinder 1</th>
<th>Cylinder 2</th>
<th>Valve 1</th>
<th>Valve 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</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>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<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>0</td>
<td>0</td>
<td>0</td>
<td>0</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>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Based on the *expert knowledge*, we also want to turn on the signal lights for Tube 1 & 2, Cylinder 1 & 2, and Valve 1 & 2 at the same time.

#### Binary → Hexadecimal

<table>
<thead>
<tr>
<th></th>
<th>Binary</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tube 1</td>
<td>0010 0000</td>
<td>20</td>
</tr>
<tr>
<td>Tube 2</td>
<td>0001 0000</td>
<td>10</td>
</tr>
<tr>
<td>Cylinder 1</td>
<td>0000 1000</td>
<td>08</td>
</tr>
<tr>
<td>Cylinder 2</td>
<td>0000 0100</td>
<td>04</td>
</tr>
<tr>
<td>Valve 1</td>
<td>0000 0010</td>
<td>02</td>
</tr>
<tr>
<td>Valve 2</td>
<td>0000 0001</td>
<td>01</td>
</tr>
<tr>
<td>NA</td>
<td>0000 0000</td>
<td>00</td>
</tr>
<tr>
<td>Tube 1 &amp; 2</td>
<td>0011 0000</td>
<td>30</td>
</tr>
<tr>
<td>Cylinder 1 &amp; 2</td>
<td>0000 1100</td>
<td>0C</td>
</tr>
<tr>
<td>Valve 1 &amp; 2</td>
<td>0000 0011</td>
<td>03</td>
</tr>
</tbody>
</table>

Possible values for “Package 1 - Byte 20”:

20; 10; 08; 04; 02; 01; 00; 30; 0C; 03
Generate 2-way Test Cases

• Package 1
  – No. of parameters: 30
  – No. of test cases: 49

• Package 2
  – No. of parameters: 58
  – No. of test cases: 77

• Package 3
  – No. of parameters: 61
  – No. of test cases: 80

• Package 4
  – No. of parameters: 50
  – No. of test cases: 90

The number of test cases for each package is less than 100
After the execution of a test case we generated, the status of module IBO2A should be “Abnormal” instead of “Normal”
**Sample Bugs We Have Detected (2)**

After the execution of a test case, the signal light for “Emergency Brake” was turned on and could not be turned off (although it should for the subsequent executions that were not used to test the brake.)

The meter does not display the corresponding data correctly (should be 999 instead of 1,400)
## Bug Detection Results

Less test cases generated, but more bugs detected

<table>
<thead>
<tr>
<th>Package</th>
<th>Method</th>
<th>Number of test cases</th>
<th>Number of bugs found</th>
<th>Did CT find all bugs detected by the practitioners?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package 1</td>
<td>Field Testing</td>
<td>98</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>CT</td>
<td>49</td>
<td>6</td>
<td>Yes</td>
</tr>
<tr>
<td>Package 2</td>
<td>Field Testing</td>
<td>102</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>CT</td>
<td>77</td>
<td>5</td>
<td>Yes</td>
</tr>
<tr>
<td>Package 3</td>
<td>Field Testing</td>
<td>116</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>CT</td>
<td>80</td>
<td>7</td>
<td>Miss 1</td>
</tr>
<tr>
<td>Package 4</td>
<td>Field Testing</td>
<td>122</td>
<td>2</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>CT</td>
<td>90</td>
<td>4</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Due to time limit, combinations between different packages were not included.
Case Study II:

Kylin Linux
Results Conclusion

- We use similar approaches to test three functionalities of Kylin
  - File exploration (similar to Windows Explorer)
  - File search
  - Shortcut creation

<table>
<thead>
<tr>
<th></th>
<th>Number of test cases</th>
<th>Number of bugs found</th>
<th>Did CT find all bugs detected by the practitioners?</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>File Exploration</strong></td>
<td>Field Testing</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>CT</td>
<td>19</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td><strong>File Search</strong></td>
<td>Field Testing</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>CT</td>
<td>17</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Miss 2</td>
</tr>
<tr>
<td><strong>Shortcut creation</strong></td>
<td>Field Testing</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>CT</td>
<td>27</td>
<td>1</td>
</tr>
</tbody>
</table>

Based on the expert knowledge, testers used some special input values to detect these 2 bugs. However, these values were not selected in our study because the special knowledge was not provided to us.
Part II

Combinatorial Decision Testing

Coverage Criteria

Decision coverage  MC/DC coverage
Statement coverage
Condition coverage

Focusing on single “Segment of Code”
**An Example**

```c
input a, b;
int k = 1;
if (a > 0)
    k = k + 1;
if (b < 5){
    k = k + 2;
} else{
    k = k - 2;
}
print 5/k;
```

A test suite with 100% statement, decision, and condition coverage

\[ t_1 = \{ a = 2, b = 3 \} \]
\[ t_2 = \{ a = -1, b = 6 \} \]

Cannot detect the bug (because these two test cases cannot make \( k = 0 \))

To detect the bug, we need to have \( a > 0 \) and \( b \geq 5 \)

Consider the *combinations* of different decisions
What Should We Do?

Cover all paths?

Obviously too expensive
What Should We Do?

How to decide which paths we should cover?

- Combinations of decisions

What inputs should we use to cover a particular path?

- Symbolic Execution

Combinatorial Testing
Extends CT to White-box Settings

• Black-box requirements-based combinatorial testing
  – Testing various combinations among input parameters (components) of a system

• White-box-based combinatorial decision testing
  – Testing various combinations among decisions of a program
Combinatorial Testing & Symbolic Execution

Black-box
Combinatorial Testing

Combinatorial Decision Testing

White-box
Symbolic Execution
**General Procedure**

- Step 1: Identify program decisions
- Step 2: Assignment of decision outcomes
- Step 3: Identification of constraints between decisions
- Step 4: Generation of a t-way path-condition set
- Step 5: Generation of test cases
Black-Box CT versus White-Box CT

- CT in black-box settings
  - Parameters
    - Possible value 1
    - Possible value 2
    - ...
    - Possible value n
  - Constraints between parameters (optional)
  - a CT tool
    - Test cases to cover all t-way possible value combinations

- CT in white-box settings
  - Decisions
    - Possible outcome 1
    - Possible outcome 2
    - Possible outcome 3
      - (true, false, not executed)
  - Constraints between decisions
  - a CT tool
    - Path-conditions to cover all t-way decision combinations

Combinatorial Testing and Its Applications @ 2016 PHM - Chengdu
Question
• Bring together stakeholders responsible for protecting the nation’s key information technologies most of which are *enabled and controlled by software.*