Mutation testing: practical aspects and cost analysis

Macario Polo and Mario Piattini
Alarcos Group
Department of Information Systems and Technologies
University of Castilla-La Mancha (Spain)

Macario.Polo@uclm.es
Contents

• What is mutation testing?
• Mutation operators
• Steps of mutation testing
• Cost reduction
• Conclusions
What is mutation testing?

• The goal of testing is to find faults on the System Under Test (SUT)
• Thus, a test suite is more or less effective depending on its ability to find faults on the SUT
A possible “SUT”

To be, or not to be: that is de question:
Whether 'tis novler in the mind to sufer
The slings and arrows of outrageus fortune,
Or to take arms against a sea of troubles,
And by opposing end them? To die: to sleep;
No more; and by a slept to say the end
The heart-ache and the thousand natural socks
That flesh is heir to, 'tis a consummation
Deboutly to be wish'd. To die, to sleep; ...

(Hamlet, by William Shakespeare)
What is mutation testing?

- If I need to select an English reviewer, maybe I would select that of you who more faults have found
- Mutation works in this way:
  - A set of “mutants” are generated from a given SUT
  - A test suite is executed against the original SUT and its mutants
  - The adequacy of the suite is measured in terms of the “mutation score”
What is mutation testing?

• The “mutation score” measures the ability of the test suite to find faults on the SUT

\[ MS(P,T) = \frac{K}{(M - E)} \]

• …where:
  • \( P \): program under test
  • \( T \): test suite
  • \( K \): number of mutants “killed”
  • \( M \): number of mutants
  • \( E \): number of functionally-equivalent mutants
What is mutation testing?

• Mutants maybe “killed” or “alive”
  – A mutant \( m \) is “killed” when it shows a different behavior that \( P \), for one or more \( t \in P \):
    \[
    \exists t \in T \ / \ f(m,t) \neq f(P,t)
    \]
  – Otherwise, the mutant is “alive”
    \[
    f(m,t) = f(P,t) \quad \forall t \in T
    \]
What is mutation testing?

• Each mutant is a copy of the program under test, usually with a small syntactic change, which is interpreted as a fault

• Mutants with $n$ faults are call $n$-order mutants
What is mutation testing?

- Each mutant is a copy of the program under test, usually with a small syntactic change, which is interpreted as a fault.
- Mutants with $n$ faults are called $n$-order mutants.

...it was a SUT with seeded faults.

To be, or not to be: that is the question:

Whether 'tis nobler in the mind to suffer

The slings and arrows of outrageous fortune,

Or to take arms against a sea of troubles,

And by opposing end them? To die, to sleep,

Devoutly to be wished.

That flesh is heir to, 'tis a consummation

More; and by a sleep to say No more; and by a sleep to say No more; and by a sleep to say No more; and by a sleep to say...
A program and four mutants

<table>
<thead>
<tr>
<th>Version</th>
<th>Code</th>
<th>Test data (a,b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P (original)</td>
<td>int sum(int a, int b) { return a + b; }</td>
<td>(1, 1) 0 -1 -2</td>
</tr>
<tr>
<td>Mutant 1</td>
<td>int sum(int a, int b) { return a - b; }</td>
<td></td>
</tr>
<tr>
<td>Mutant 2</td>
<td>int sum(int a, int b) { return a * b; }</td>
<td></td>
</tr>
<tr>
<td>Mutant 3</td>
<td>int sum(int a, int b) { return a / b; }</td>
<td></td>
</tr>
<tr>
<td>Mutant 4</td>
<td>int sum(int a, int b) { return a + b++; }</td>
<td></td>
</tr>
</tbody>
</table>
What is mutation testing?

- Faults are introduced ("seeded") by mutation operators, and try to imitate actual programmer errors.
- Mutation relies on the "Coupling effect": a test suite that detects all simple faults in a program is so sensitive that it also detects more complex faults.
Some mutation operators

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABS</td>
<td>Substitution of a variable $x$ by $\text{abs}(x)$</td>
</tr>
<tr>
<td>ACR</td>
<td>Substitution of a variable array reference by a constant</td>
</tr>
<tr>
<td>AOR</td>
<td>Arithmetic operator replacement ($a+b$ by $a-b$)</td>
</tr>
<tr>
<td>CRP</td>
<td>Substitution of a constant value</td>
</tr>
<tr>
<td>ROR</td>
<td>Relational operator replacement ($A \text{ and } B$ by $A \text{ or } B$)</td>
</tr>
<tr>
<td>RSR</td>
<td>Return statement substitution ($\text{return 5}$ by $\text{return 0}$)</td>
</tr>
<tr>
<td>SDL</td>
<td>Removal of a sentence</td>
</tr>
<tr>
<td>UOI</td>
<td>Unary operator insertion (instead of $x$, write $-x$)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Steps of mutation testing

- Mutation testing has three main steps:
  1. Mutant generation
  2. Mutant execution
  3. Result analysis
Mutant generation

- Almost each executable instruction of the original program can be mutated with several mutation operators.
- Therefore, the number of mutants generated for a normal program may be huge.
- The cost of compilation of all mutants may be also significant.
Mutant generation

- Offutt et al. (1996): 10 programs from 10 to 48 executable sentences produce from 183 to 3010 mutants
- Mresa and Botaci (1999): 11 programs with 43,7 LOC produce 3211 mutants
- One of our experiments: mean of 76,7 LOC produce a mean of 150 mutants
Mutant generation

- Java version of Myers (1979) triangle-type problem: 61 LOC, 262 mutants

- A widely-used mutation tool is MuJava (Ma, Offutt and Kwon, 2005)
Mutant generation: the MuJava tool

Mutants (and operators)

Number of mutants per operator

Number of mutants

Fault introduced

Modified sentence

Mutant

23    public static final int SCALENE = 1;
24    public static final int ISOSCELES = 2;
25    public static final int EQUILATERAL = 3;
26    public static final int NO_TRIANGLE = 0;
27    public TriangleType()
28    {
29    }
30    {
31    public void set(int i)
32    {
33    i = i + i;
34    }
Mutant generation: the MuJava tool

• In general, a parser is required to generate mutants:
  – $a+b$ is translated into $a-b$, $a\times b$, $a/b$
  – Then, these program versions are compiled

• MuJava uses “Mutant Schemata Generation”
  – With some operators, it substitutes (at bytecode level) $a+b$ by $a\ OPERATOR\ b$
  – Then, all the program versions are directly generated with no need of compiling
Mutant execution

• In this case, the problem is the huge number of test cases that must be executed: each case is executed against the original program and the mutants.

• For testing a simple `BankingAccount` class, with 96 mutants and 300 test cases, $96 \times 300 = 28,800$ executions are required (with at least 28.800 accesses to the database, etc.)
Mutant execution

• All the outputs must be compared to detect which mutants are killed:
  – In the *BankingAccount* example, the outputs of the 300 test cases with the original and the 96 mutants
  – Actually, killed mutants can be removed for further comparisons
Mutant execution: \textit{testooj}

- \textit{testooj} is a relatively user-friendly research tool developed in the University of Castilla-La Mancha
- Generates test cases in several formats and according to several generation strategies
- Executes test cases against versions and gives some additional results
Mutant execution: testooj
### The killing matrix

|   | A               | B   | C   | D   | E   | F   | G   | H   | I   | J   | K   | L   | M   | N   | O   | P   | Q   | R   | S   | T   | U   | V   |
|---|----------------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 1 | Mutants        | t76 | t77 | t78 | t54 | t73 | t74 | t75 | t81 | t27 | t49 | t50 | t51 | t22 | t23 | t24 | t46 | t47 | t48 | t19 | t20 | t21 |
| 2 | IOD_1          | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   |
| 3 | PRV_1          | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   |
| 4 | PRV_2          | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   |
| 5 | PRV_3          | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   |
| 6 | PRV_4          | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   |
| 7 | PRV_5          | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   |
| 8 | AOIS_1         | X   |   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   |
| 9 | AOIS_10        | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   |
| 10| AOIS_11        | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   |
| 11| AOIS_12        | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   |
| 12| AOIS_13        | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   |
| 13| AOIS_14        | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   |
| 14| AOIS_17        | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   |
| 15| AOIS_18        | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   |
| 16| AOIS_2         | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   |
| 17| AOIS_21        | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   |
| 18| AOIS_22        | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   |
| 19| AOIS_25        | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   |
| 20| AOIS_26        | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   |
| 21|                | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   | X   |
Result analysis

• The major difficulties appear with the detection of functionally equivalent mutants
Result analysis

• A functionally equivalent mutant is a mutant which never will be killed.
• Actually, the “fault” introduced is not a fault, but a code de-optimization.

<table>
<thead>
<tr>
<th>Version</th>
<th>Code</th>
<th>Test data (a,b)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P (original)</td>
<td><code>int sum(int a, int b){   return a + b;};</code></td>
<td>(1, 1) 0 (-1, 0) (-1, -1)</td>
</tr>
<tr>
<td>Mutant 4</td>
<td><code>int sum(int a, int b){   return a + b++;};</code></td>
<td>2 0 -1 -2</td>
</tr>
</tbody>
</table>

| Mutant 4 | 2 0 -1 -2 |
Result analysis

- Mutation operators have different pronoeness for producing equivalent mutants

<table>
<thead>
<tr>
<th>Program</th>
<th># of 1-order mutants</th>
<th>Total equivalent mutants</th>
<th>Equivalent mutants per mutation operator</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>AOIS</td>
</tr>
<tr>
<td>Bisect</td>
<td>63</td>
<td>19</td>
<td>14</td>
</tr>
<tr>
<td>Bub</td>
<td>82</td>
<td>12</td>
<td>9</td>
</tr>
<tr>
<td>Find</td>
<td>179</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Fourballs</td>
<td>212</td>
<td>44</td>
<td>42</td>
</tr>
<tr>
<td>Mid</td>
<td>181</td>
<td>43</td>
<td>38</td>
</tr>
<tr>
<td>TriTyp</td>
<td>309</td>
<td>70</td>
<td>54</td>
</tr>
</tbody>
</table>

83.5%  7.4%  3.7%  3.7%  1.6%
Result analysis

- The example is an occurrence of the AOIS operator

<table>
<thead>
<tr>
<th>Version</th>
<th>Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>P (original)</td>
<td><code>int sum(int a, int b) { return a + b; }</code></td>
</tr>
<tr>
<td>Mutant 4</td>
<td><code>int sum(int a, int b) { return a + b++; }</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test data (a, b)</th>
<th>(1, 1)</th>
<th>(0, 0)</th>
<th>(-1, 0)</th>
<th>(-1, -1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>2</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
</tr>
<tr>
<td>M4</td>
<td>2</td>
<td>0</td>
<td>-1</td>
<td>-2</td>
</tr>
</tbody>
</table>
Result analysis

• Detection of equivalent mutants is usually manual
• Grüen, Schuler and Zeller (2009) report that some equivalent mutants require up to 15 minutes to be detected
• Offutt and Pan (1997) demonstrated that it is possible to automatically detect almost 50% of functionally equivalent mutants if the program under test is annotated with constraints
Result analysis

• Once more, the selection of the best mutation operators is essential (this is *selective mutation*):
  – Replacement of numerical constants
  – Negate jump conditions
  – Replacement of arithmetic operators
  – Omission of method calls (instead of \( x=\text{foo}() \), write \( x=0.0 \))
Result analysis

• Other strategies rely on *weak mutation*:
  – “Strong” mutation has three conditions:
    • Reachability (the instruction must be reached)
    • Necessity (once the sentences has been reached, the test case must cause an erroneous state on the mutant)
    • Sufficiency (the erroneous state must be propagated to the output)
  – Instead of observing the output of each test case, the idea of *weak mutation* is to detect changes in intermediate states (reachability+necessity)
Cost reduction

• Summarizing, in mutant generation:
  – Selective mutation (use of the best operators)
  – Mutant Schemata Generation/Mutation at bytecode level

• In mutant execution:
  – Reduction of the test suite
  – N-order mutation

• In result analysis:
  – To take advantage of the previous techniques
Reduction of the test suite

• The optimal test-suite reduction problem:
  – *Given*: Test Suite $T$, a set of test-case requirements $r_1, r_2, ..., r_n$, that must be satisfied to provide the desired test coverage of the program.
  – *Problem*: Find $T' \subset T$ such that $T'$ satisfies all $r_i$ and ($\forall T'' \subset T$, $T''$ satisfies all $r \Rightarrow |T'| \leq |T''|$)

• It is NP-hard (no solution in polynomial time): solutions approached with greedy algorithms
Reduction of the test suite

- Example: 6 test cases, 7 mutants

<table>
<thead>
<tr>
<th></th>
<th>tc1</th>
<th>tc2</th>
<th>tc3</th>
<th>tc4</th>
<th>tc5</th>
<th>tc6</th>
</tr>
</thead>
<tbody>
<tr>
<td>m1</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>m2</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>m3</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>m4</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>m5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>m6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>m7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
</tr>
</tbody>
</table>
Reduction of the test suite

- The greedy approach selects the test case killing more mutants

<table>
<thead>
<tr>
<th></th>
<th>tc1</th>
<th>tc2</th>
<th>tc3</th>
<th>tc4</th>
<th>tc5</th>
<th>tc6</th>
</tr>
</thead>
<tbody>
<tr>
<td>m1</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>m2</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>m3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>m4</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>m5</td>
<td></td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>m6</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>m7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>

T' = tc2, tc3, tc4
Reduction of the test suite in testooj
N-order mutation

• The idea is to have mutants with more than one fault
N-order mutation

• Thus:
  – The number of mutants is closed to half the size of the original suite
  – Each equivalent mutant will be *probably* combined with a non-equivalent mutant, what implies a reduction of the number of equivalent mutants
N-order mutation

• However, there exists the possibility of having a test case that only finds one of the two faults injected

```java
class Operations {
    int sum(int a, int b) {
        return a+b; a-b;
    }
    int mult(int a, int b) {
        return a*b; a-b;
    }
}
```

```java
public int test1() {
    Operations op=new Operations();
    return op.sum(5, 3);
}
```

Original: 8
Mutant 1: 5 (killed)
Mutant 2: 8 (alive)
Mutant 1,2 : 5 ("partially" killed)
N-order mutation

- Algorithms for mutants combination
  - LastToFirst
  - DifferentOperators
  - RandomMix
LastToFirst

- First with the last, second with the penultime, etc.
- AOIS_1-ROR_6, AOIS_10-ROR_4, etc.
- It gets a half of the original mutant set
Different Operators

- Combines mutants obtained with different operators
- Mutants proceeding from the least frequent operator are used more than once
RandomMix

- Makes a pure random combination

<table>
<thead>
<tr>
<th>AOIS_1</th>
<th>AOIS_35</th>
<th>AOIS_56</th>
<th>AOIU_11</th>
<th>AORB_1</th>
<th>ROR_1</th>
</tr>
</thead>
<tbody>
<tr>
<td>AOIS_10</td>
<td>AOIS_37</td>
<td>AOIS_58</td>
<td>AOIU_12</td>
<td>AORB_10</td>
<td>ROR_10</td>
</tr>
<tr>
<td>AOIS_12</td>
<td>AOIS_38</td>
<td>AOIS_59</td>
<td>AOIU_13</td>
<td>AORB_3</td>
<td>ROR_4</td>
</tr>
<tr>
<td>AOIS_13</td>
<td>AOIS_39</td>
<td>AOIS_60</td>
<td>AOIU_2</td>
<td>AORB_5</td>
<td>ROR_6</td>
</tr>
<tr>
<td>AOIS_14</td>
<td>AOIS_4</td>
<td>AOIS_71</td>
<td>AOIU_3</td>
<td>AORB_7</td>
<td></td>
</tr>
<tr>
<td>AOIS_15</td>
<td>AOIS_40</td>
<td>AOIS_72</td>
<td>AOIU_4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AOIS_16</td>
<td>AOIS_41</td>
<td>AOIS_73</td>
<td>AO_6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AOIS_17</td>
<td>AOIS_68</td>
<td>AOIS_74</td>
<td>AOIU_7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AOIS_18</td>
<td>AOIS_44</td>
<td>AOIS_75</td>
<td>AOIU_8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AOIS_19</td>
<td>AOIS_45</td>
<td>AOIS_76</td>
<td>AOIU_9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>AOIS_20</td>
<td>AOIS_47</td>
<td>AOIS_77</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AOIS_25</td>
<td>AOIS_48</td>
<td>AOIS_78</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AOIS_27</td>
<td>AOIS_5</td>
<td>AOIS_79</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AOIS_3</td>
<td>AOIS_52</td>
<td>AOIS_80</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AOIS_31</td>
<td>AOIS_54</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Results with benchmark programs

<table>
<thead>
<tr>
<th>Program</th>
<th>LOC</th>
<th># of 1-order mutants</th>
<th>Equivalent 1-order mutants</th>
<th># of test cases</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Number</td>
<td>%</td>
<td></td>
</tr>
<tr>
<td>Bisect</td>
<td>31</td>
<td>63</td>
<td>19</td>
<td>30.15%</td>
</tr>
<tr>
<td>Bub</td>
<td>54</td>
<td>82</td>
<td>12</td>
<td>14.63%</td>
</tr>
<tr>
<td>Find</td>
<td>79</td>
<td>179</td>
<td>0</td>
<td>0.00%</td>
</tr>
<tr>
<td>Fourballs</td>
<td>47</td>
<td>212</td>
<td>44</td>
<td>20.75%</td>
</tr>
<tr>
<td>Mid</td>
<td>59</td>
<td>181</td>
<td>43</td>
<td>23.75%</td>
</tr>
<tr>
<td>TriTyp</td>
<td>61</td>
<td>309</td>
<td>70</td>
<td>22.65%</td>
</tr>
<tr>
<td><strong>Mean</strong></td>
<td></td>
<td></td>
<td></td>
<td>18.66%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Program</th>
<th>LastToFirst</th>
<th>DifferentOperators</th>
<th>RandomMix</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># mutants ((^{(*)}))</td>
<td>Equivalent</td>
<td># mutants ((^{(*)}))</td>
</tr>
<tr>
<td></td>
<td>#</td>
<td>%</td>
<td>#</td>
</tr>
<tr>
<td>Bisect</td>
<td>32 (50,8%)</td>
<td>5</td>
<td>15,63</td>
</tr>
<tr>
<td>Bub</td>
<td>41 (50%)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Find</td>
<td>90 (50,3%)</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Fourballs</td>
<td>107 (50,4%)</td>
<td>5</td>
<td>4,67</td>
</tr>
<tr>
<td>Mid</td>
<td>91 (50,3%)</td>
<td>8</td>
<td>8,79</td>
</tr>
<tr>
<td>TriTyp</td>
<td>155 (50,2%)</td>
<td>7</td>
<td>4,51</td>
</tr>
<tr>
<td><strong>Means</strong></td>
<td></td>
<td>5,6%</td>
<td>4,4%</td>
</tr>
</tbody>
</table>
## Results with benchmark programs

<table>
<thead>
<tr>
<th>Bisect</th>
<th>Bub</th>
<th>Find</th>
<th>Mid</th>
<th>TriTyp</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>KM</strong></td>
<td><strong>TC</strong></td>
<td><strong>KM</strong></td>
<td><strong>TC</strong></td>
<td><strong>KM</strong></td>
</tr>
<tr>
<td>17</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>18</td>
<td>5</td>
<td>3</td>
<td>75</td>
<td>14</td>
</tr>
<tr>
<td>20</td>
<td>1</td>
<td>4</td>
<td>20</td>
<td>17</td>
</tr>
<tr>
<td>22</td>
<td>1</td>
<td>178</td>
<td>81</td>
<td>19</td>
</tr>
<tr>
<td>23</td>
<td>1</td>
<td>29</td>
<td>9</td>
<td>20</td>
</tr>
<tr>
<td>24</td>
<td>3</td>
<td>30</td>
<td>3</td>
<td>21</td>
</tr>
<tr>
<td>25</td>
<td>1</td>
<td>59</td>
<td>5</td>
<td>22</td>
</tr>
<tr>
<td>26</td>
<td>1</td>
<td>61</td>
<td>4</td>
<td>23</td>
</tr>
<tr>
<td>28</td>
<td>1</td>
<td>62</td>
<td>3</td>
<td>25</td>
</tr>
<tr>
<td>30</td>
<td>1</td>
<td>63</td>
<td>5</td>
<td>26</td>
</tr>
<tr>
<td>31</td>
<td>1</td>
<td>64</td>
<td>5</td>
<td>28</td>
</tr>
<tr>
<td>35</td>
<td>1</td>
<td>65</td>
<td>8</td>
<td>29</td>
</tr>
<tr>
<td>37</td>
<td>2</td>
<td>66</td>
<td>11</td>
<td>30</td>
</tr>
<tr>
<td>40</td>
<td>2</td>
<td>67</td>
<td>19</td>
<td>31</td>
</tr>
<tr>
<td>42</td>
<td>1</td>
<td>68</td>
<td>6</td>
<td>32</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Fourballs</th>
<th><strong>KM</strong></th>
<th><strong>TC</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>43</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>46</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>49</td>
<td>14</td>
<td></td>
</tr>
<tr>
<td>50</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>52</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>54</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>55</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>56</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>59</td>
<td>10</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>67</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mid</th>
<th><strong>KM</strong></th>
<th><strong>TC</strong></th>
<th><strong>KM</strong></th>
<th><strong>TC</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>3</td>
<td>67</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>6</td>
<td>62</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>9</td>
<td>64</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>39</td>
<td>3</td>
<td>67</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TriTyp</th>
<th><strong>KM</strong></th>
<th><strong>TC</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>86</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>88</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>89</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>91</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>94</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>96</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>97</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
## Results with industrial software

<table>
<thead>
<tr>
<th>Program</th>
<th># of sentences</th>
<th>WMC</th>
<th># of methods</th>
<th># of 1-order mutants</th>
<th># of available test cases</th>
<th>1-order mutants killed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ciudad</td>
<td>177</td>
<td>65</td>
<td>23</td>
<td>203</td>
<td>37</td>
<td>92%</td>
</tr>
<tr>
<td>IgnoreList</td>
<td>26</td>
<td>8</td>
<td>3</td>
<td>27</td>
<td>6</td>
<td>85%</td>
</tr>
<tr>
<td>PluginTokenizer</td>
<td>157</td>
<td>27</td>
<td>16</td>
<td>94</td>
<td>14</td>
<td>31%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ciudad</th>
<th>KM</th>
<th>TC</th>
<th>KM</th>
<th>TC</th>
<th>KM</th>
<th>TC</th>
<th>KM</th>
<th>TC</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>9</td>
<td></td>
<td>94</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td></td>
<td>95</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td></td>
<td>96</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>69</td>
<td>1</td>
<td></td>
<td>97</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>75</td>
<td>1</td>
<td></td>
<td>98</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>86</td>
<td>3</td>
<td></td>
<td>99</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>88</td>
<td>1</td>
<td></td>
<td>100</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>89</td>
<td>1</td>
<td></td>
<td>101</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>91</td>
<td>1</td>
<td></td>
<td>102</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>92</td>
<td>2</td>
<td></td>
<td>118</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>93</td>
<td>1</td>
<td></td>
<td>181</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PluginTokenizer</th>
<th>KM</th>
<th>TC</th>
<th>KM</th>
<th>TC</th>
</tr>
</thead>
<tbody>
<tr>
<td>18</td>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IgnoreList</th>
<th>KM</th>
<th>TC</th>
<th>KM</th>
<th>TC</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Cost-risk analysis

<table>
<thead>
<tr>
<th>K</th>
<th>Benchmark</th>
<th>Industrial</th>
<th>Benchmark</th>
<th>Industrial</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Cost</td>
<td>TC</td>
<td>Risk</td>
<td>TC</td>
</tr>
<tr>
<td>1</td>
<td>1.00</td>
<td>174</td>
<td>0.00</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0.50</td>
<td>2</td>
<td>0.21</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0.33</td>
<td>123</td>
<td>0.21</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>0.25</td>
<td>68</td>
<td>0.36</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0.20</td>
<td>0</td>
<td>0.44</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0.17</td>
<td>0</td>
<td>0.44</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>0.14</td>
<td>0</td>
<td>0.44</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0.13</td>
<td>0</td>
<td>0.44</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>0.11</td>
<td>0</td>
<td>0.44</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>0.10</td>
<td>0</td>
<td>0.44</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>0.09</td>
<td>1</td>
<td>0.44</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>0.08</td>
<td>0</td>
<td>0.44</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>0.08</td>
<td>0</td>
<td>0.44</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>0.07</td>
<td>4</td>
<td>0.44</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>0.07</td>
<td>0</td>
<td>0.44</td>
<td>1</td>
</tr>
</tbody>
</table>

\[
\text{risk}(k) = 1 - \frac{\# \text{ of test cases killing } k \text{ mutants or more}}{\text{total } \# \text{ of test cases}}
\]
Cost-risk analysis

Cost and risk of k-order mutation

- Cost
- Benchmarks
- Industrial
Conclusions

• Mutation is an excellent testing technique
• From the point of view of research, it is mature
• From the industry point of view, user-friendly tools are required
• Mutation is also applied at other levels: black-box, components, web services, models…
Mutation testing: practical aspects and cost analysis

Macario Polo and Mario Piattini
Alarcos Group
Department of Information Systems and Technologies
University of Castilla-La Mancha (Spain)

Macario.Polo@uclm.es