Using mutation analysis for assessing and comparing testing coverage criteria
(+Research idea)

James H. Andrews, Lionel C. Briand, Yvan Labiche, and Akbar Siami Namin

*IEEE Transactions on software engineering, 2006*

2011. 08. 10. (Wed)
Donghwan Shin
Contents

- Introduction
- Related work
- Experimental description
- Analysis results
- Conclusion
- Discussion (research idea)
  - Motivation
  - Goal & contribution
  - Plan & issues
Introduction (1/3)

- Test coverage criteria
  - Measure used to identify which source code of a program has been tested
    » Block, Decision, C-Use, P-Use, and more coverage exist
  - Important consideration in the safety certification of safety-critical systems
    » The U.S. Department of Transportation Federal Aviation Administration (FAA) has formal requirements for structural coverage in the certification of safety-critical airborne systems [DO-178B]

Subsumption relation among test coverage criteria
Introduction (2/3)

- Mutation analysis
  - Measure the fault detection abilities of given test suites
  - **Mutation operators** define rule of making faulty variants of program
    - Important factor in mutant generation
  - **Mutants** are the resulting faulty versions of the original program
    - If the test case fail (i.e. detect) mutants, we say the test case “kill” the mutants

Simple mutation analysis example

Original
```c
add(int a, int b) {
    return c = a + b;
}
```

Mutant
```c
add(int a, int b) {
    return c = a - b;
}
```

Mutant operator: replace + as -

Test case [a=1, b=2, c=3] kill the mutant!
Introduction (3/3)

- Motivation
  - Insufficient study of comparing the relative cost and effectiveness of the testing coverage criteria
    - Fundamental questions regarding the relationships between fault detection, test suite size, and control/data flow coverage level
  - No empirical study that has directly assessed the use of mutants by comparing them with results obtained on real faults

- Goal
  - Investigate the feasibility of using mutation analysis to assess the testing coverage criteria
    - Compare the behavior of mutants and real faults
  - Assessing and comparing four common control and data flow criteria
    - Block, Decision, C-Use, and P-Use coverage criteria
# Related work

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>All-Use, Decision</td>
<td>Def-Use (variant of All-Use), Decision</td>
<td>All-use, Decision</td>
<td>Block, Decision, C-Use, P-Use</td>
</tr>
<tr>
<td>Subject programs</td>
<td>9 tiny program (33–60 LOC)</td>
<td>7 small program (141–512 LOC)</td>
<td>One middle size industrial program (over 5000 LOC)</td>
<td>One middle size industrial program (over 5000 LOC)</td>
</tr>
<tr>
<td>Faulty versions</td>
<td>7 hand seeded faults</td>
<td>130 hand seeded faults</td>
<td>11 real faults</td>
<td>34 real faults + 736 mutants</td>
</tr>
<tr>
<td>Cost-effectiveness study</td>
<td>X</td>
<td>Δ</td>
<td>Δ</td>
<td>O</td>
</tr>
<tr>
<td>Feasibility study for mutation analysis</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>O</td>
</tr>
</tbody>
</table>
Experimental description (1/6)

- Experimental question
  
  〉 Q1. Are the mutation scores **good predictors** of actual fault detection ratio?
  
  〉 Q2. What is the **cost** of achieving given **levels of the coverage** criteria and how do the criteria compare to each other in this respect?
  
  〉 Q3. Can we determine what **levels of coverage** should be achieved to obtain reasonable levels of **fault detection** effectiveness?
  
  〉 Q4. What is the relative **cost-effectiveness** of the investigated control and data flow coverage criteria?
  
  〉 Q5. What is the gain of using coverage criteria compared to **random testing**?
  
  〉 Q6. How is the above results affected by **fault detection difficulty**?
Experimental description (2/6)

- Overview of experiments

### Test programs
- **Subject program** (clean version)
- **Mutant generation**
  - 4 types mutant operators
- **736 mutants**
  - Uniform distribution over all the possible mutants
- **Faulty programs** (34 faulty versions)
  - Subject program for feasibility study of mutation analysis

### Test cases
- **Test pool** (13850 test cases)
- **Suite gen algorithm**
  - Coverage check by ATAC
- **Random selection**
  - Making groups
- **Coverage suite** (230 test suites)
  - Meet specific coverage level for each criteria
- **Random suite** (1700 test suites)
  - No consideration for coverage

### Test results
- **For each test suite** (S)
  - Coverage levels (each criteria)
  - Suite size (# of test cases)
  - # of faults detected by S
  - # of mutants detected by S

- **Analysis for test suites**
  - $Af \sim Am$
  - % coverage $\sim Am$
  - % coverage $\sim$ size
  - Am $\sim$ size
  - Coverage suite vs. random
  - Am $\sim$ % coverage, for “easy”, “hard”, and “very hard” mutants

Af: fault detection ratio, Am: mutant detection ratio

© Donghwan Shin, 2011
Experimental description (3/6)

- Subject program
  - C program originally developed by the European Space Agency, and used subsequently in other experiments
  - During “operational use and testing”, 35 faults were identified and eliminated and details were preserved
    - In this study, use 34 faulty versions of the program for the “real” faults

- Test pool
  - Total 13,850 test cases were generated in the previous related works
    - [F.I. Vokolos and P.G. Frankl] generated 10,000 randomized test cases
    - [G. Rothermel et al.] added 3,850 test cases to ensure that executable Decision was covered by at least 30 test cases in each direction
  - Resulting test pool covers 90, 85, 85, and 80 percent of all Block, Decision, C-Use, and P-Use respectively (baseline for feasible coverage)

---

Experimental description (4/6)

- Mutant generation
  - Mutation operators
    - Replace an integer constant $C$ by 0, 1, -1, $C+1$, or $C-1$
    - Replace an arithmetic, relational, logical, bitwise logical, inclement/decrement, or assignment operator by another operator from the same class
    - Negate the decision in an if or while statement
    - Delete a statement
  - Mutants number control
    - Select every 10th mutant to randomly selecting 10 percent of the mutants
      - Total 11,379 mutants were generated and compiled, but too many
    - Remove mutants not killed by any test cases
    - Thus, we have 736 mutants actually used
Experimental description (5/6)

- Coverage suite generation

  - Algorithm to generate test suites
    - Four coverage criteria: Block, Decision, C-Use, and P-Use

```
Algorithm generate_testsuite
Input: Coverage criterion C, target coverage T
Output: Test suite achieving at least T% coverage of criterion C

Set suite = {};
Set suite_coverage = 0;
While (suite_coverage < T) {
    Choose a test case "tc_chosen" randomly from the test pool;
    Add tc_chosen to suite;
    Set new_coverage_value = coverage percentage for suite on criterion C;
    If (new_coverage_value <= suite_coverage) {
        Remove tc_chosen from suite;
    } Else {
        Set suite_coverage = new_coverage_value;
    }
}
Return suite;
```
Experimental description (6/6)

- Coverage suite generation (cont’d)
  - Using the previous algorithm, generate test suites from 50 percent to 95 percent for each of the four coverage criteria
    - Within xx.00-xx.99 percent coverage, generate 5 test suites
      - For the Block coverage criteria, five suites for 50.00-50.99 percent level, the other five for 51.00-51.99 percent level, and so on
    - Total 230 test suites were generated and selected, and refer to these as the CS (coverage suite)

- Random suite generation
  - Generate 1,700 random test suites from the same pool
    - Each test suite size from one test case to 150 test cases
    - Without any consideration for coverage criteria, and refer to these as the RS (random suite)
Analysis results (1/8)

- Q1. Are mutant scores good predictors for actual fault detection?
  - Calculate the difference \((Af-Am)\) for each test suites
    - \(Af\): fault detection ratio, \(Am\): mutant detection ratio
  - Calculate significant difference p-value
    - p-value < 0.001

<table>
<thead>
<tr>
<th></th>
<th>Mean</th>
<th>Median</th>
<th>2.5%</th>
<th>97.5%</th>
<th>Min</th>
<th>Max</th>
<th>Std Dev</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block</td>
<td>0.013</td>
<td>0.011</td>
<td>-0.092</td>
<td>0.139</td>
<td>-0.14</td>
<td>0.16</td>
<td>0.057</td>
</tr>
<tr>
<td>C-Use</td>
<td>0.018</td>
<td>0.012</td>
<td>-0.078</td>
<td>0.131</td>
<td>-0.14</td>
<td>0.16</td>
<td>0.057</td>
</tr>
<tr>
<td>Decision</td>
<td>0.016</td>
<td>0.023</td>
<td>-0.104</td>
<td>0.139</td>
<td>-0.22</td>
<td>0.23</td>
<td>0.065</td>
</tr>
<tr>
<td>P-Use</td>
<td>0.015</td>
<td>0.014</td>
<td>-0.104</td>
<td>0.132</td>
<td>-0.154</td>
<td>0.227</td>
<td>0.063</td>
</tr>
</tbody>
</table>

Am will not significantly and systematically under/over-estimate Af
Q1. Are mutant scores good predictors for actual fault detection? (cont’d)

- Calculate the magnitude of relative error (MRE) value
  - \[ MRE = \frac{|A_f - A_m|}{A_f} \text{ over } A_f, \] which means the normalized error

In practice, high coverage level makes more accurate prediction of \( A_f \) based on \( A_m \)

Average MRE tends to decrease as the coverage level increases
Q1. Are mutant scores good predictors for actual fault detection? (cont’d)

- Model the linear regression between \( A_f \) and \( A_m \)
  - \( R^2 = 0.908 \), slope = 0.967

Am is an unbiased and accurate prediction of \( A_f \)

Because \( A_m \) is a good predictor of \( A_f \), we use the large number of mutants to the analysis from Q2 to Q7
Analysis results (4/8)

Q2. Comparing the cost of coverage criteria

- Model the relationships between the coverage level of all four criteria and test suite size for the CS test suites
  - Use appropriate nonlinear regression analysis
  - Assume that the effort associated with a coverage level is proportional to test suite size (number of test cases)

Comparing coverage level-suite size

Achieving higher level becomes increasingly expensive

Cost of criteria (for given level):
Block < C-Use < Decision < P-Use
Analysis results (5/8)

- Q3. Are there minimal, required coverage level?
  - Model the relationships between \( A_m \) and the coverage level
    - Focus on the higher part of the coverage level range

Comparing \( A_m \)-coverage level

No strong exponential relation at the higher range of coverage

Mutant score (for given level):
Block < C-Use < Decision < P-Use
Analysis results (6/8)

Q4. Cost-effectiveness of coverage criteria

- Model the relationships between $A_m$ and the test suites size for the CS
  - Use nonlinear regression model
  - $A_m = a \cdot \text{size}^b$

Regression analysis results for $A_m$-size model

<table>
<thead>
<tr>
<th>$A_m$-Size (Model)</th>
<th>a</th>
<th>b</th>
<th>$R^2$</th>
<th>Area under curve (0-100)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Block</td>
<td>0.213</td>
<td>0.309</td>
<td>0.976</td>
<td>67.52</td>
</tr>
<tr>
<td>C-use</td>
<td>0.219</td>
<td>0.300</td>
<td>0.977</td>
<td>67.66</td>
</tr>
<tr>
<td>Decision</td>
<td>0.226</td>
<td>0.292</td>
<td>0.973</td>
<td>67.12</td>
</tr>
<tr>
<td>P-use</td>
<td>0.226</td>
<td>0.287</td>
<td>0.972</td>
<td>65.85</td>
</tr>
</tbody>
</table>

Non of the four criteria is more cost-effective than the others
Analysis results (7/8)

Q5. Comparing coverage criteria to random test suites

- Model the relationships between Am and the **test suites size** for the RS and compare fitted graph with the result of CS
  - Little difference depending on the test pool

**Comparing cost-effectiveness with RS**

Coverage criteria test suites are clearly more cost-effective than random test and more so for higher coverage level.
Q6. Impact of fault detection probabilities

- Compare the relationships between $A_m$ and coverage level for subsets of seeded mutants that are relatively more difficult to detect given our test pool
  - Hard mutants: detected by less than 5 percent of the test cases in the pool
  - Very hard mutants: detected by less than 1.5 percent of the test cases in the pool

Fault detection and the achieved coverage level is very sensitive to the detection difficulty of the faults
Conclusion

- Contribution
  - Show the feasibility of using mutation analysis to assess the cost-effectiveness of common control and data flow criteria
    - Mutant score is the unbiased estimator of actual fault detection
  - Give reliable experimental answers to the fundamental questions regarding the relationships between fault detection, test suite size, and control/data flow coverage
    - Successfully reduce random variation with middle size industrial program and comprehensive test pool
Discussion

- **Research motivation**
  - In safety-critical, software testing adequacy and certification is very important issue
    - Certain coverage level must be satisfied in software testing
    - Lustre (aviation), FBD (nuclear energy) is widely used data flow language in safety-critical system

---

Small FBD program example for calculating `th_X_logic_Trip`

```
(1) ADD_INT
    k_X_Trip_Setpoint
    k_X_Trip_Hys

f_X[ ]
ADD1

(2) LE_INT
    th_Prev_X_Trip[ ]

(3) LE_INT
    f_X[ ]

(4) SEL
    G
    IN0
    IN1

SEL4

(5) TON
    IN
    Q

th_X_Logic_Trip
```
Discussion

- Research motivation (cont’d)
  - Testing coverage criteria defined in data flow language is not totally comparable with coverage criteria defined in procedural language
    - CFGs (procedural language) and DFGs (data flow language) are based on different computational model
  - Unclear relationship between coverage criteria based on procedural language and data flow language

![Diagram showing coverage criteria comparison between data flow and procedural languages.](image-url)
Discussion

- Research goal & contribution

  - Using mutation analysis, assess and compare the fault detection power between coverage criteria based on procedural language and data flow language

  - Suggest theoretical background for testing objective and certification in data flow language by experiments
    - Help to make clear testing objective in FBD programs
    - Do not need to use unclear coverage criteria defined in procedural language, not only for FBD programs but also the other data flow programs
Discussion

- Research plan & issues

**Subject program** (Data flow language)

- Mutant generation
  - Defined mutant operators
- Test case generation
  - For specific coverage level

**Mutants**

**Transformed program** (procedural language)

- Test case generation
  - For specific coverage level

**DF-suite** (Data flow based)

**CF-suite** (Control flow based)

Transformation by tool

**Issues**

- How to define mutant operators? -> Paper survey, inspection for data flow programs
- What criteria to be compared? -> At first, Basic coverage for FBD and Decision for C
- How to make test cases with achieving target coverage level? -> Fully use existing tools
- How to run (simulate) such test cases? -> Big problem

**Test results**

- **For DF-suite (D)**
  - 100% coverage level for specific criteria for data flow language
  - Suite size (# of test cases)
  - # of mutants detected by D

- **For CF-suite (C)**
  - 100% coverage level for specific criteria for procedural language
  - Suite size (# of test cases)
  - # of mutants detected by C

- **Compare mutant score**
  - Rank coverage criteria for 100% coverage level
Thank you!