Customized Program Mutation
(aka Mutation analysis for the real world: Effectiveness, efficiency and proper tool support)

René Just

UMass, Amherst
Laboratory of Advanced Software Engineering Research

March 13, 2017
A quick poll

What is a good mutation score?
A quick poll

What is a good mutation score?

~100% is good if the mutants are good proxies for real faults.

Everything else is meaningless: the mutation score is heavily inflated due to a high degree of redundancy.
Big picture: the past, the present, and the future

**Past:** manual fault seeding

**Present:** generic program mutation

```
0 1 0 0 0 1 1
1 1 1 0 0 1 0
0 0 1 0 1 1 0
0 1 0 0 0 1 1
```

```
0 1 0 0 0 1 1
1 1 1 0 0 1 0
0 0 1 0 1 1 0
0 1 0 0 0 1 1
```

```
0 1 0 0 0 1 1
1 1 1 0 0 1 0
0 0 1 0 1 1 0
0 1 0 0 0 1 1
```

```
0 1 0 0 0 1 1
1 1 1 0 0 1 0
0 0 1 0 1 1 0
0 1 0 0 0 1 1
```

ROR, COR, AOR, …
Big picture: the past, the present, and the future

**Problem:** not all mutants are equally strong, and program context affects mutant utility.

**Solution:** customize program mutations to program context.
Big picture: the past, the present, and the future

**Past:** manual fault seeding

```
0 1 0 0 0 1 1
1 1 1 0 0 1 0
0 0 1 0 1 1 0
0 1 0 0 0 1 1
```

**Present:** generic program mutation

```
0 1 0 0 0 1 1
1 1 1 0 0 1 0
0 0 1 0 1 1 0
0 1 0 0 0 1 1
```

**Future:** customized program mutation

```
0 1 0 0 0 1 1
1 1 1 0 0 1 0
0 0 1 0 1 1 0
0 1 0 0 0 1 1
```

---

![Diagram](image_url)

ROR, COR, AOR, …
Some terminology

**Mutation operator** vs. **mutation operator group**

\[ lhs < rhs \rightarrow lhs \neq rhs \]

\[ lhs < rhs \rightarrow lhs \leq rhs \]

\[ lhs < rhs \rightarrow \ldots \]

ROR
Some terminology

**Mutation operator** vs. **mutation operator group**

\[ \text{lhs} < \text{rhs} \rightarrow \text{lhs} \neq \text{rhs} \]

\[ \text{lhs} < \text{rhs} \rightarrow \text{lhs} \leq \text{rhs} \]

\[ \text{lhs} < \text{rhs} \rightarrow \ldots \]

**An effective mutant:**
- is coupled to one or more real faults
- is NOT equivalent
- is NOT dominated by other mutants
- is NOT redundant or trivial
High-level goal: effective mutation operators
High-level goal: effective mutation operators

An effective mutation operator generates a large ratio of non-equivalent, non-trivial, fault-coupled dominators.
Fault-coupled mutants

- Mutants are not similar to real faults.
- BUT most real faults are coupled to some mutants.
- Number of mutants increases superlinear when fault-coupling is increased.

Gopinath et al., ISSRE’14, Pearson et al., ICSE’17, Just et al., FSE’14
Is selective mutation the solution?

No free lunch
- No selection strategy for mutation operator groups works equally well for all programs.

Program context matters!

Zhang et al., ICSE’10, Gopinath et al., ICSE’16, Kurtz et al., FSE’16
public double getAbsAvg(int[] nums) {
    double avg = 0;

    for (int i = 0; i < nums.length; ++i) {
        if (nums[i] < 1) {
            avg -= (double)nums[i] / nums.length;
        } else {
            avg += (double)nums[i] / nums.length;
        }
    }
    return avg;
}
Program context: motivational example (1)

Original program

```java
public double getAbsAvg(int[] nums) {
    double avg = 0;
    for (int i = 0; i < nums.length; ++i) {
        if (nums[i] < 1) {
            avg -= (double)nums[i] / nums.length;
        } else {
            avg += (double)nums[i] / nums.length;
        }
    }
    return avg;
}
```

Mutation operator

`lhs < rhs` → `lhs != rhs`
Program context: motivational example (1)

Original program

```java
global double getAbsAvg(int[] nums) {
    double avg = 0;
    for (int i = 0; i < nums.length; ++i) {
        if (nums[i] < 1) {
            avg -= (double)nums[i] / nums.length;
        } else {
            avg += (double)nums[i] / nums.length;
        }
    }
    return avg;
}
```

Mutation operator

 lhs < rhs ➡️ lhs != rhs

equivalent mutant

dominator mutant

Context: different kinds of lexically enclosing statements (for vs. if)
Program context: motivational example (2)

Original program

```java
public double getAbsAvg(int[] nums) {
    double avg = 0;

    for (int i = 0; i < nums.length; ++i) {
        if (nums[i] < 1) {
            avg -= (double)nums[i] / nums.length;
        } else {
            avg += (double)nums[i] / nums.length;
        }
    }

    return avg;
}
```

Mutation operator

0 → -1
public double getAbsAvg(int[] nums) {
    double avg = 0;
    for (int i = 0; i < nums.length; ++i) {
        if (nums[i] < 1) {
            avg -= (double)nums[i] / nums.length;
        } else {
            avg += (double)nums[i] / nums.length;
        }
    }
    return avg;
}
public double getAbsAvg(int[] nums) {
    double avg = 0;

    for (int i = 0; i < nums.length; ++i) {
        if (nums[i] < 1) {
            avg -= (double)nums[i] / nums.length;
        } else {
            avg += (double)nums[i] / nums.length;
        }
    }

    return avg;
}
Program context: motivational example (3)

**Original program**

```java
public double getAbsAvg(int[] nums) {
    double avg = 0;

    for (int i = 0; i < nums.length; ++i) {
        if (nums[i] < 1) {
            avg -= (double)nums[i] / nums.length;
        } else {
            avg += (double)nums[i] / nums.length;
        }
    }

    return avg;
}
```

**Mutation operator**

lhs < rhs → lhs <= rhs

- *trivial mutant*
- *dominator mutant*

**Context:** different kinds of operands (variable vs. literal)
Program context: summary

- Program context affects mutant utility
  - Utility of mutation operators differs, even within a single mutation operator group (e.g., ROR).
  - Utility of a mutation operator differs, even within a single method.

- Different dimensions of program context
  - Kind of lexically enclosing statement
  - Kind and data type of operator and operands
  - Scope and visibility
  - Coding style and syntactic sugar
  - ...
Customized program mutation
Modeling program context using the AST

- The abstract syntax tree (AST) provides relevant context information for:
  - Mutated nodes
  - Parent nodes
  - Children nodes

- Can be augmented with project-specific context information:
  - Coding guidelines
Some promising results

● “Fresh out of the oven”

● Preliminary study
  ○ 100,000 mutants (5 open source projects)*.
  ○ Approximation of equivalent/dominator/trivial mutants, using thorough test suites*.

● Comparison of tree-based classifiers for mutant utility
  ○ Mutation operator groups
  ○ Mutation operators
  ○ Program context

*http://www.defects4j.org
Classifiers for mutant utility (non-equivalent)

- **Perfect prediction**
- **Better prediction**

- 3-dim context
- 1-dim context
- Mutation operator
- Mutation operator group
- Random
Classifiers for mutant utility (non-equivalent)

- Mutation operator group is marginally better than random.
- Program context improves over mutation operator.
- Similar results for trivial mutants and dominator strength.
Error rate of 3-dim context classifier (non-equivalent)
- Training error shows room for improvements.
- Overfitting is NOT (yet) a problem.
- Similar results for trivial mutants and dominator strength.
Recall the high-level goal

**Goal**: generate a large ratio of non-equivalent, non-trivial, fault-coupled dominators.
Effectiveness: mutation operator groups
Effectiveness: mutation operators
Effectiveness: mutation operators + program context
Customized program mutation
Customized program mutation

- Generate **dominator** mutants.
- Don’t generate **equivalent** mutants.
- Avoid **redundant** mutants.
- Avoid **trivial** mutants.

**Consider 4th dimension (fault-coupling) to determine thresholds!**
Tool support
Major: overview

Why compiler-integrated mutation?
- Orders of magnitude faster than source code mutation.
- Mutate what the developer actually wrote (no desugared/simplified code).

Just et al., ASE’11, ISSTA’14
http://www.mutation-testing.org
Major: customized program mutation

class A {
  ...
}

Compiler

Embedded mutants

Just et al., ASE’11, ISSTA’14
http://www.mutation-testing.org
Acknowledgments

Bob Kurtz

Paul Ammann

Huzefa Rangwala

Jeff Offutt

Andrew McCallum

Miltos Allamanis
Customized program mutation

- Effectiveness of mutation operators differs even within operator groups
- Program context affects mutant utility
- Different dimensions of program context

http://www.mutation-testing.org
http://www.defects4j.org