Evolutionary Mutation Testing Applied to Object-Oriented Systems

Pedro Delgado-Pérez, Inmaculada Medina-Bulo, Sergio Segura, Antonio García-Domínguez and Juan José Domínguez-Jiménez

UCA Universidad de Cádiz

UCASE

Summer School Search-Based Software Engineering

June 2016

Outline

Mutation Testing

A brief description

- Involves inserting simple syntactic changes in the program using **mutation operators**.
- This modification creates a new version called **mutant**.
- A non-detected mutant reveals a deficiency in our test suite.
- **Equivalence:** The change cannot be detected by any input.

Original program

```
if (x > 5) { ... }
```

Mutant: relational operator replaced

```
if (x < 5) { ... }
```

Test case	x = 5	x = 10
Original: $x > 5$	false	true
Mutant: $x < 5$	false	false
Classification:	Alive	Dead

Goals in Mutation Testing

Test suite evaluation

Measure how good is a test suite at detecting faults.

Test suite refinement

Improve the test suite to kill (detect) surviving mutants.

Test suite refinement

Search for mutants inducing the design of new test cases.

Goals in Mutation Testing

Test suite evaluation

Measure how good is a test suite at detecting faults.

Test suite refinement

Improve the test suite to kill (detect) surviving mutants.

Test suite refinement

Search for mutants inducing the design of new test cases.

Phases in Mutation Testing

Four main phases:

1. Definition of mutation operators.

2. Development of a mutation framework

3. Evaluation of experimental results

4. Reducing the high cost of applying mutation testing.

Cost Reduction Techniques

Techniques "Do Fewer"

Reduce the number of mutants.

- *Mutant sampling.*
- *Selective mutation.*
- *High order mutation.*

Cost Reduction Techniques

Techniques "Do Fewer"

Reduce the number of mutants.

- *Mutant sampling.*
- *Selective mutation.*
- *High order mutation.*

Recent technique

Evolutionary Mutation Testing

J. J. Domínguez- Jiménez, A. Estero-Botaro, I. Medina-Bulo and A. García-Domínguez
Evolutionary Mutation Testing
Information and Software Technology, 2011.
http://dx.doi.org/10.1016/j.infsof.2011.03.008

Evolutionary Mutation Testing

Evolutionary Mutation Testing *proposes the generation of a subset of the mutants by means of an* ***evolutionary algorithm****.*

This algorithm favors that the subset contains ***mutants with great potential to assist the tester in improving the test suite*** *with new test cases.*

Fitness Function

Execution matrix:

Operator	Mutant	Test cases: $test_1$	$test_2$	$test_3$
o_1	m_1	0	1	0
	m_2	1	0	0
	m_3	1	0	1
o_2	m_4	0	0	0
	m_5	0	0	1
	m_6	1	0	1

Fitness of mutants/individuals

- **Best valued (Strong mutants):**
  - **Potentially equivalent:** not killed by the current test suite.
  - **Resistant hard to kill:** killed by a single test case, which does not kill any other mutants.
- **Worst valued**: killed by many test cases, which in turn kill many other mutants.

Algorithm

Operator	Location	Attribute

Three fields are used to identify a mutant

- **Operator**: code representing the mutation operator.
- **Location**: order of location in the source file.
- **Attribute**: variant used in the location.

Original program

```
if (x > 0) {
   if (x > 5) {...}
```

Mutant

```
if (x > 0) {
   if (x < 5) {...}
```

- **Operator**: set of operators:
  - 1 (relational operator replaced)
  - 2 (arithmetic operator replaced)
- **Location**: possible locations: 1, 2
- **Attribute**: possible variants: >=, <=, <

MUTANT: 1 / 2 / 3

Algorithm

Operator	Location	Attribute

Three fields are used to identify a mutant

- **Operator**: code representing the mutation operator.
- **Location**: order of location in the source file.
- **Attribute**: variant used in the location.

Original program

```
if (x > 0) {
   if (x > 5) {...}
```

Mutant

```
if (x > 0) {
   if (x < 5) {...}
```

- **Operator**: set of operators:
  - 1 (relational operator replaced)
  - 2 (arithmetic operator replaced)
- **Location**: possible locations: 1, 2
- **Attribute**: possible variants: >=, <=, <

MUTANT: 1 / 2 / 3

Algorithm

Operator	Location	Attribute

Three fields are used to identify a mutant

- **Operator**: code representing the mutation operator.
- **Location**: order of location in the source file.
- **Attribute**: variant used in the location.

Original program

```
if (x > 0) {
   if (x > 5) {...}
```

Mutant

```
if (x > 0) {
   if (x < 5) {...}
```

- **Operator**: set of operators:
  - 1 (relational operator replaced)
  - 2 (arithmetic operator replaced)
- **Location**: possible locations: 1, 2
- **Attribute**: possible variants: $>=, <=, <$

MUTANT: 1 / 2 / 3

Algorithm

Individual generation

Individuals in a generation are:

1. **Randomly generated.**
2. **Generated by reproductive operators***:
   - **Mutation operators**.
   - **Crossover operators**.

* *Selection of individuals: roulette wheel method.*

Algorithm

Mutation operators:

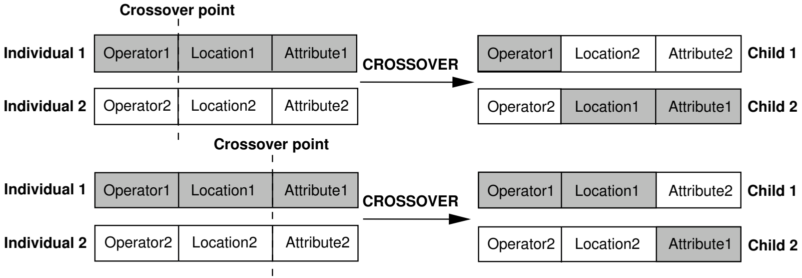
Operator	Location	Attribute

Algorithm

Mutation operators:

Operator	Location	Attribute

Crossover operators:



Research line

Motivation

- One of the most used programming languages [a].
- Search for cost reduction techniques.
- Evolutionary Mutation Testing had only been applied to WS-BPEL.

[a] 3rd position in the TIOBE index in June 2016

Goal

Is Evolutionary Mutation Testing useful in other contexts?

Research line

Motivation

- One of the most used programming languages [a].
- Search for cost reduction techniques.
- Evolutionary Mutation Testing had only been applied to WS-BPEL.

[a] 3^{rd} position in the TIOBE index in June 2016

Goal

Is Evolutionary Mutation Testing useful in other contexts?

Mutation operators

Class mutation operators

- Related to object-oriented features:
  - Inheritance
  - Polymorphism
  - Method overloading

Set of mutation operators

- The set of operators should be defined for each programming language:
  1. Common to other programming languages (Java and C#).
  2. Specific to the language.

P. Delgado-Pérez, I. Medina-Bulo, J. J. Domínguez-Jiménez, A. García-Domínguez and F. Palomo-Lozano.
Class mutation operators for C++ object-oriented systems
Annals of telecommunications, 2015.
http://dx.doi.org/10.1007/s12243-014-0445-4

Example

Example "Inheritance" block: IOD (Overriding method deletion)

Original:

```
class A {                          class B: public A{
    ... ...                            ... ...
    int method(){... ...};             int method(){... ...};
};                                 };
```

Mutant:

```
class A {                          class B: public A{
    ... ...                            ... ...
    int method(){... ...};             /*Deleted*/
};                                 };
```

Example

Example “Inheritance” block: IOD (Overriding method deletion)

Original:

```
class A {                          class B: public A{
    ... ...                            ... ...
    int method(){... ...};             int method(){... ...};
};                                 };
```

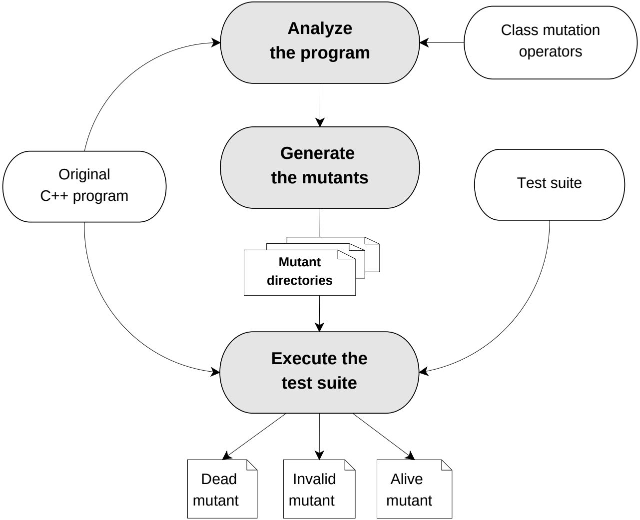
Mutant:

```
class A {                          class B: public A{
    ... ...                            ... ...
    int method(){... ...};             /*Deleted*/
};                                 };
```

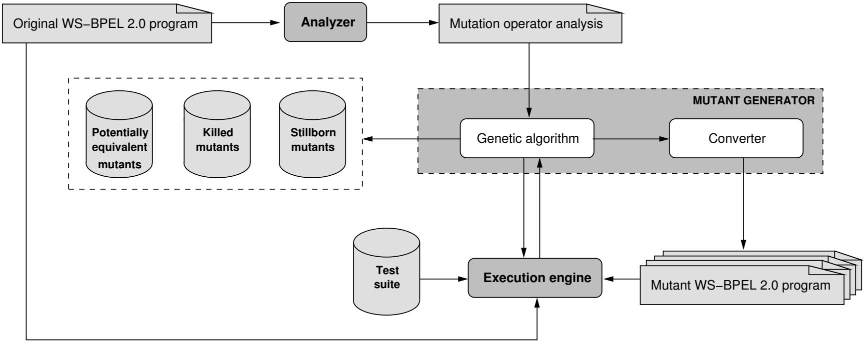
Main differences:

- Class operators are less prolific than traditional operators.
- High percentage of equivalence.

C++ mutation tool flow chart

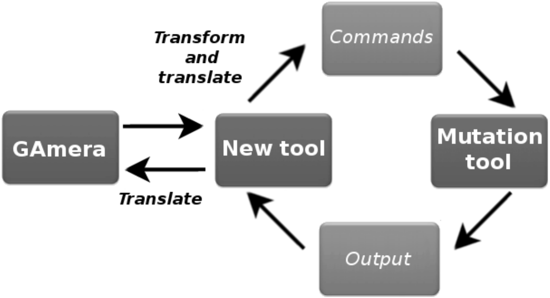


GAmera flow chart



*Its **modular architecture** allows to reuse the the genetic algorithm with different programming languages.*

GAmera / C++ mutation tool



Connection between applications

- Development of a **new application** to connect GAmera and the C++ mutation tool.
- **Tasks of this application**:
  - Transform GAmera execution commands into understandable commands for the mutation tool.
  - Translate the output between both applications.

Experiments

Evolutionary Mutation Testing VS Random Selection

1. **Random selection**: mutants selected one by one randomly.
2. **Evolutionary Mutation Testing**: generations are produced according to these parameters*:

• **Population size**:	5 %
• **New individuals randomly generated**:	10 %
• **New individuals generated by reproductive operators**:	90 %
• - Mutation probability:	30 %
• - Crossover probability:	70 %

* *These parameters were experimentally found as the best.*

Experiments

Evolutionary Mutation Testing VS Random Selection

1. **Random selection**: mutants selected one by one randomly.
2. **Evolutionary Mutation Testing**: generations are produced according to these parameters*:
   - **Population size**: 5 %
   - **New individuals randomly generated**: 10 %
   - **New individuals generated by reproductive operators**: 90 %
   - - Mutation probability: 30 %
   - - Crossover probability: 70 %

* *These parameters were experimentally found as the best.*

Termination

Stop condition: generation of a percentage of the total of **strong mutants**:

- 75 %
- 90 %

30 executions with different seeds.

Experiments

Case studies

- **Three programs** with different number of mutants and strong mutants*.
- We use the test suite distributed with the programs.

* *Strong mutants are known thanks to a previous execution.*

	Program 1	**Program 2**	**Program 3**	**Total**
Total mutants	219	614	1,146	1,979
Valid	208	433	681	1,322
Strong	103	159	348	610
% Strong mutants	49.5%	36.7%	51.1%	46.1%
Test cases	61	57	46	-

Experiments

Result 1

EMT outperforms Random in all the cases.

Result 2

The result is better with a low threshold (75 %).

Result 3

The efficiency does not scale with the number of mutants.

Result 4

The lower the percentage of strong mutants, the more efficient.

Threshold	*75%*		*90%*	
Technique	*EMT*	*Random*	*EMT*	*Random*
Program 1				
Mean	69.87	75.11	85.35	89.07
Median	70.09	75.79	85.38	89.72
MIN	62.55	67.57	78.99	82.64
Max	76.71	81.27	90.41	93.15
SD	3.57	3.57	2.67	2.86
Program 2				
Mean	64.91	74.93	84.32	89.98
Median	64.74	74.83	84.12	90.22
Min	60.58	69.70	77.85	85.01
Max	71.49	80.61	89.73	93.64
SD	2.59	2.78	3.34	1.78
Program 3				
Mean	69.96	74.43	87.84	89.76
Median	70.15	74.34	88.09	89.75
Min.	66.05	71.64	83.33	86.21
Max.	73.38	78.88	90.13	93.71
SD	1.98	2.00	1.60	1.58

Experiments

Result 1

EMT outperforms Random in all the cases.

Result 2

The result is better with a low threshold (75 %).

Result 3

The efficiency does not scale with the number of mutants.

Result 4

The lower the percentage of strong mutants, the more efficient.

Threshold	*75%*		*90%*	
Technique	*EMT*	*Random*	*EMT*	*Random*
Program 1				
Mean	69.87	75.11	85.35	89.07
Median	70.09	75.79	85.38	89.72
MIN	62.55	67.57	78.99	82.64
Max	76.71	81.27	90.41	93.15
SD	3.57	3.57	2.67	2.86
Program 2				
Mean	64.91	74.93	84.32	89.98
Median	64.74	74.83	84.12	90.22
Min	60.58	69.70	77.85	85.01
Max	71.49	80.61	89.73	93.64
SD	2.59	2.78	3.34	1.78
Program 3				
Mean	69.96	74.43	87.84	89.76
Median	70.15	74.34	88.09	89.75
Min.	66.05	71.64	83.33	86.21
Max.	73.38	78.88	90.13	93.71
SD	1.98	2.00	1.60	1.58

Experiments

Result 1

EMT outperforms Random in all the cases.

Result 2

The result is better with a low threshold (75 %).

Result 3

The efficiency does not scale with the number of mutants.

Result 4

The lower the percentage of strong mutants, the more efficient.

Threshold	*75%*		*90%*	
Technique	*EMT*	*Random*	*EMT*	*Random*
Program 1				
Mean	69.87	75.11	85.35	89.07
Median	70.09	75.79	85.38	89.72
MIN	62.55	67.57	78.99	82.64
Max	76.71	81.27	90.41	93.15
SD	3.57	3.57	2.67	2.86
Program 2				
Mean	64.91	74.93	84.32	89.98
Median	64.74	74.83	84.12	90.22
Min	60.58	69.70	77.85	85.01
Max	71.49	80.61	89.73	93.64
SD	2.59	2.78	3.34	1.78
Program 3				
Mean	69.96	74.43	87.84	89.76
Median	70.15	74.34	88.09	89.75
Min.	66.05	71.64	83.33	86.21
Max.	73.38	78.88	90.13	93.71
SD	1.98	2.00	1.60	1.58

Experiments

Result 1

EMT outperforms Random in all the cases.

Result 2

The result is better with a low threshold (75 %).

Result 3

The efficiency does not scale with the number of mutants.

Result 4

The lower the percentage of strong mutants, the more efficient.

Threshold	*75%*		*90%*	
Technique	*EMT*	*Random*	*EMT*	*Random*
Program 1				
Mean	69.87	75.11	85.35	89.07
Median	70.09	75.79	85.38	89.72
MIN	62.55	67.57	78.99	82.64
Max	76.71	81.27	90.41	93.15
SD	3.57	3.57	2.67	2.86
Program 2				
Mean	64.91	74.93	84.32	89.98
Median	64.74	74.83	84.12	90.22
Min	60.58	69.70	77.85	85.01
Max	71.49	80.61	89.73	93.64
SD	2.59	2.78	3.34	1.78
Program 3				
Mean	69.96	74.43	87.84	89.76
Median	70.15	74.34	88.09	89.75
Min.	66.05	71.64	83.33	86.21
Max.	73.38	78.88	90.13	93.71
SD	1.98	2.00	1.60	1.58

Achievements

1. C++ mutation tool connected with the evolutionary algorithm.
2. Results confirms Evolutionary Mutation Testing as an efficient cost reduction technique.

Future work

- Confirm this tendency in new experiments.
- Introduce changes in the genetic algorithm.
- Check how this technique helps refine the test suite.

Summer School SBSE 2016

Pedro Delgado-Pérez

University of Cádiz

pedro.delgado@uca.es
https://ucase.uca.es/pedro-delgado-perez