04834580 Software Engineering (Honor Track) 2024-25

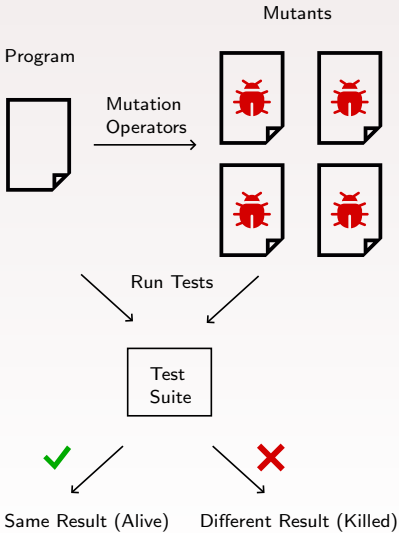
Mutation Testing

Sergey Mechtaev
mechtaev@pku.edu.cn
School of Computer Science, Peking University

Definition (Mutation Testing [1])

A technique to assess the quality of a given test set, where faults are deliberately seeded into the original program, by a simple syntactic change, to create a set of faulty programs called mutants, each containing a different syntactic change. These mutants are executed against the input test set to see if the seeded faults can be detected.

Mutation Testing



Mutation Testing Assumption

Definition (Competent Programmer Hypothesis [2])

The version of program produced by a competent programmer is close to the final correct version of a program.

Mutation Testing Example

北京大学

Original Code:

```
def is_even(num):
    return num % 2 == 0
```

Test Suite:

```
def test_is_even():
    assert is_even(2) == True
    assert is_even(3) == False
```

Mutant Example: Incorrect Relational Operator

北京大学

Mutated Code:

```
def is_even(num):
    return num % 2 != 0
```

Test Results:

- `test_is_even()` fails because the mutant changes the behavior.
- **Mutant is killed!**

Definition

A mutant is considered equivalent if it behaves the same as the original program for every input.

Original Code:

```
if x > 0:
  return 1
else:
  return -1
```

Mutant:

```
if x > 0 and x != 0:
  return 1
else:
  return -1
```

- Both versions are semantically identical for all inputs. No test case can distinguish them.
- Testers waste time and computational resources attempting to "kill" non-killable mutants.

Mutation Coverage:

$$\text{Mutation Coverage} = \frac{\text{Number of killed mutants}}{\text{Total number of mutants}}$$

Mutation Coverage (Non-Equivalent Mutants):

$$\text{Mutation Coverage} = \frac{\text{Number of killed mutants}}{\text{Total number of non-equivalent mutants}}$$

References I

[1] Yue Jia and Mark Harman.
An analysis and survey of the development of mutation testing.
IEEE transactions on software engineering, 37(5):649–678, 2010.

[2] Timothy A Budd, Richard J Lipton, Richard A DeMillo, and Frederick G Sayward.
Mutation analysis.
Yale University. Department of Computer Science, 1979.