Software Testing and Validation
2023/24

MuJava

an automated class mutation system

Group 13

100731 - Edson da Veiga
110857 - André Alves
110906 - Francisco Teixeira

Professor João Pereira

Contet

- Introduction
- Related Work
- Proposed Solution
- Evaluation
- Conclusion
- Future Work

Object-oriented Testing

Research confirms that testing methods proposed for procedural approach are not adequate for Object-oriented approach.

Ex. Statement coverage

Object-oriented software testing poses additional problems due to Object-oriented fetchers.

Ex. Inheritance, polymorphism, and encapsulation

Mutation Testing

Mutation testing is a fault-based testing technique that measures the effectiveness of test cases.

A better way for generating software tests and evaluating the quality of software

Surge of interest in the 1980s

Terminology: Mutation

A mutation is a (small) change in your codebase, for example:

```
public int getPrice( int amountOfThings, boolean hasCoupon) {
    if (amountOfThings >= DISCOUNT_AMOUNT || hasCoupon) {
        return amountOfThings * DISCOUNT_AMOUNT;
    }
    return amountOfThings * NORMAL_COST;
}
```

```
no usages
public int getPrice( int amountOfThings, boolean hasCoupon) {
    if (amountOfThings > DISCOUNT_AMOUNT || hasCoupon) {
        return amountOfThings * DISCOUNT_AMOUNT;
    }
    return amountOfThings * NORMAL_COST;
}
```

Terminology: Mutant

A mutant is a mutated version of your class or method.

```
DebitCard >= anotherDebitCard
        ^(type = anotherDebitCard type)
          and: [ number = anotherDebitCard number]
```

Operator: Change #and: by #or:

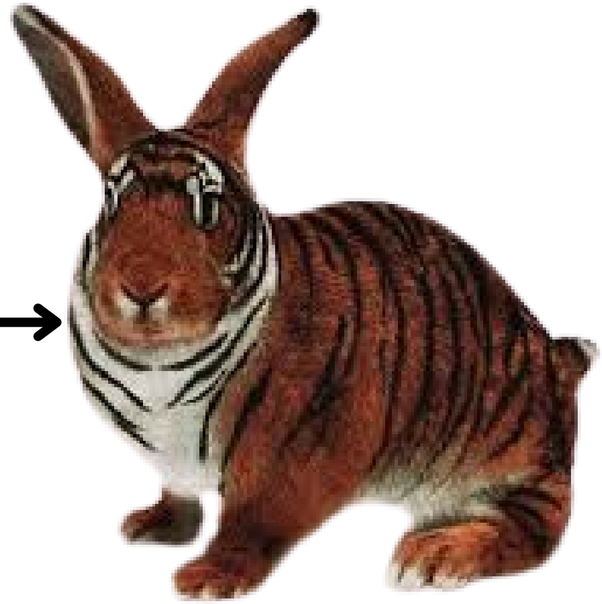
```
DebitCard >= anotherDebitCard
        ^(type = anotherDebitCard type)
          or: [ number = anotherDebitCard number]
```

Process of creating the Mutant

The Source Code

Mutation Process

The Mutation “Operator”



The “Mutant”

Related Work

Previous Solutions Problems

- The research was developed before the class mutation operators were well established and did not support all of the current mutation operators.

- Their tool separately compiles each mutated class, a very slow process.

- Their tool generates mutants, but does not support the execution of mutants and tests.

Mutation

many executions
of programs

COST

Proposed Solution

Proposed Solution

Objective: Make an more efficient mutation testing

Done by a meta-mutant that only needs to compile once

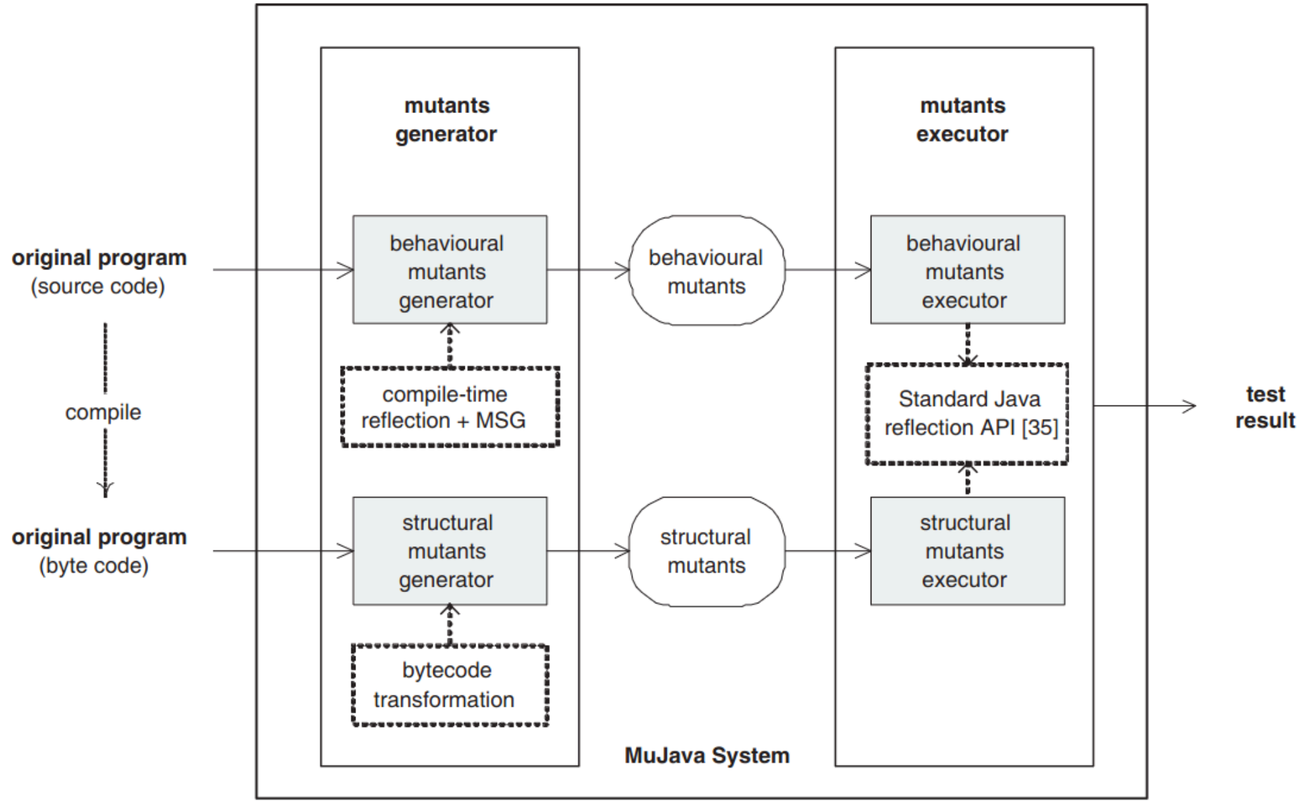


Figure 1. Overall structure of MuJava.

Proposed Solution

MSG

Mutant Schemata Generation

Behavioural Mutants

+

Bytecode Translation

Structural Mutants

Behavioural Mutants

Behavioral mutants already existed in previous work

Mujava uses java specific optimizations to make the process faster

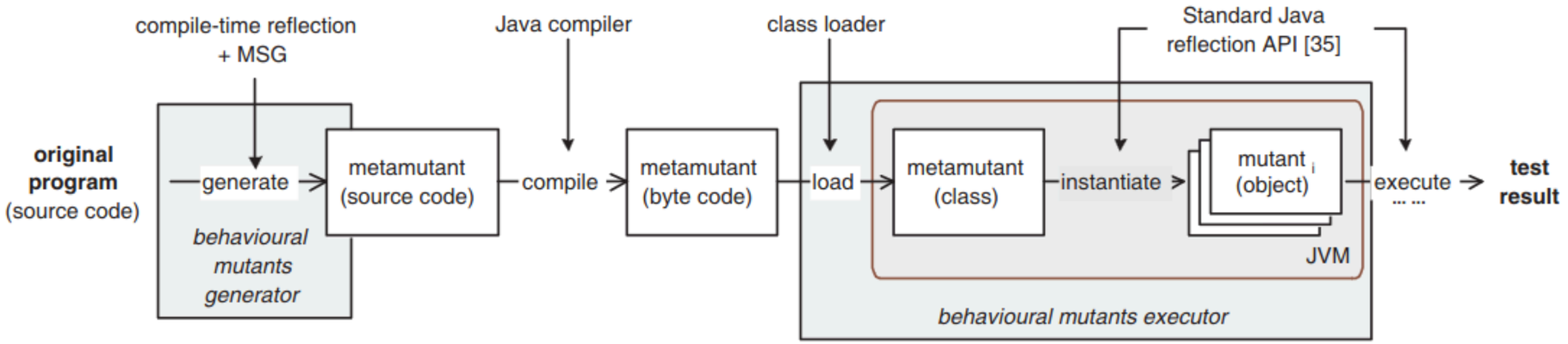


Figure 2. Mutation process for behavioural mutants.

Structural Mutants

Structural mutants are OO specific

So MuJava uses the BCEL library to handle structural mutants

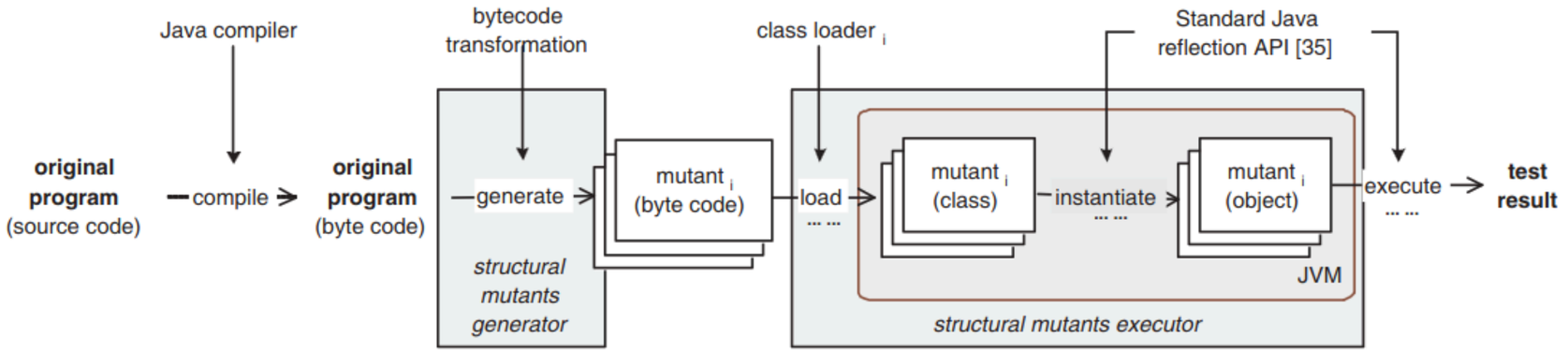


Figure 3. Mutation process for structural mutants.

Evaluation of MuJava

Evaluation of MuJava

Objective: Assess efficiency and effectiveness of the MuJava System

Methodology:

- Selection of Benchmark Programs
- Generation of Mutants
- Execution of Test Cases
- Performance Metrics

Results:

- Efficiency Gained (Table IV/V/VI of the Paper)
- Effectiveness:
  - High Detection Rate
  - Comprehensive Mutation Operators
- Scalability

Conclusion & Future Work

Conclusion

- **Effectiveness of Mutation Testing**
  - High Fault Detection
- **Efficiency Improvements**
  - Innovative Techniques
  - Performance Gains
- **Comprehensive Mutation Operators**
  - OO-Specific Operators
  - Wide Range of Faults

Future Work

- Extension to other languages
- Integration with other testing tools
- Automation and usability enhancement
- Performance optimization
- Empirical studies

Thank You

For Your Attention