A Compiler-Integrated, Extensible, and Efficient Tool for the Mutation Analysis of Java Programs

René Just¹, Franz Schweiggert¹, and Gregory M. Kapfhammer²

¹Ulm University, Germany
²Allegheny College, USA

Saarland University, Saarbrücken, Germany
January 24, 2012
Overview of MAJOR

A Tool for Mutation Analysis
Overview of MAJOR

A Tool for Mutation Analysis

Compiler-Integrated

Just, Kapfhammer, and Schweiggert
Ulm University, Allegheny College

A Compiler-Integrated, Extensible, and Efficient Tool for the Mutation Analysis of Java Programs
Overview of MAJOR

- Compiler-Integrated
- A Tool for Mutation Analysis
- Fast and Scalable
Overview of MAJOR

A Tool for Mutation Analysis

Compiler-Integrated

Domain Specific Language

Fast and Scalable
Overview of MAJOR

A Tool for Mutation Analysis

- Compiler-Integrated
- Domain Specific Language
- Fast and Scalable
- Configurable and Extensible
Overview of MAJOR

- Compiler-Integrated
- Mutation Coverage Information
- Domain Specific Language

A Tool for Mutation Analysis

- Fast and Scalable
- Configurable and Extensible
Overview of MAJOR

- Compiler-Integrated
- Mutation Coverage Information
- Domain Specific Language

- A Tool for Mutation Analysis
  - Fast and Scalable
  - Enables Optimized Workflow
  - Configurable and Extensible
Overview of Mutation Analysis
Overview of Mutation Analysis

Methodically inject small syntactical faults into the program under test

Mutation Analysis
Overview of Mutation Analysis

Methodically inject small syntactical faults into the program under test

Mutation Analysis

Unbiased and powerful method for assessing test oracles and input values
Overview of Mutation Analysis

Methodically inject small syntactical faults into the program under test

Unbiased and powerful method for assessing test oracles and input values

Useful for fault seeding during the empirical study of testing and debugging techniques
Overview of Mutation Analysis

```java
public int eval(int x){
    int a=3, b=1, y;

    y = a * x;

    y += b;
    return y;
}

public int max(int a, int b){
    int max = a;

    if(b>a){
        max=b;
    }

    return max;
}
```

Just, Kapfhammer, and Schweiggert

A Compiler-Integrated, Extensible, and Efficient Tool for the Mutation Analysis of Java Programs
Overview of Mutation Analysis

```java
public int eval(int x) {
    int a = 3, b = 1, y;

    y = a * x;
    y += b;
    return y;
}

public int max(int a, int b) {
    int max = a;

    if (b > a) {
        max = b;
    }

    return max;
}
```
Overview of Mutation Analysis

public int eval(int x) {
    int a=3, b=1, y;
    y = a * x;
    y += b;
    return y;
}

public int max(int a, int b) {
    int max = a;
    if(b>a) {
        max=b;
    }
    return max;
}
## Working Example

```java
public int eval(int x) {
    int a = 3, b = 1, y;

    y = a * x;

    y += b;
    return y;
}
```

1. Define mutation operators: $MOP(x \times y) = \{ x - y, x + y, x/y \}$
2. Determine whether current expression or statement is affected by mutation
3. Apply mutation operators

Just, Kapfhammer, and Schweiggert
Ulm University, Allegheny College

A Compiler-Integrated, Extensible, and Efficient Tool for the Mutation Analysis of Java Programs
Working Example

```java
public int eval(int x) {
    int a = 3, b = 1, y;
    y = a * x;
    y += b;
    return y;
}
```

1. Define mutation operators  \[ MOP(x \times y) = \{ x - y, x + y, x/y \} \]
2. Determine whether current expression or statement is affected by mutation
3. Apply mutation operators

Just, Kapfhammer, and Schweiggert

A Compiler-Integrated, Extensible, and Efficient Tool for the Mutation Analysis of Java Programs
Working Example

```java
public int eval(int x) {
    int a = 3, b = 1, y;

    y = a * x;

    y += b;
    return y;
}
```

1. Define mutation operators \( MOP(x \times y) = \{x - y, x + y, x/y\} \)
2. Determine whether current expression or statement is affected by mutation
3. Apply mutation operators
Working Example

```java
public int eval(int x) {
    int a=3, b=1, y;

    y = (M_NO==1)? a - x :
        a * x;

    y += b;
    return y;
}
```

1. Define mutation operators
   \[ MOP(x \times y) = \{x - y, x + y, x/y\} \]
2. Determine whether current expression or statement is affected by mutation
3. Apply mutation operators
Working Example

```java
public int eval(int x) {
    int a = 3, b = 1, y;

    y = (M_NO==2) ? a + x :
        (M_NO==1) ? a - x :
                    a * x;

    y += b;
    return y;
}
```

1. Define mutation operators $MOP(x \times y) = \{x - y, x + y, x/y\}$
2. Determine whether current expression or statement is affected by mutation
3. Apply mutation operators

Just, Kapfhammer, and Schweiggert

Ulm University, Allegheny College

A Compiler-Integrated, Extensible, and Efficient Tool for the Mutation Analysis of Java Programs
Working Example

```java
public int eval(int x) {
    int a = 3, b = 1, y;

    y = (M_NO == 3) ? a / x :
        (M_NO == 2) ? a + x :
        (M_NO == 1) ? a - x :
            a * x;
    y += b;
    return y;
}
```

1. Define mutation operators \( MOP(x \ast y) = \{ x - y, x + y, x/y \} \)
2. Determine whether current expression or statement is affected by mutation
3. Apply mutation operators
Working Example

```java
public int eval(int x){
    int a=3, b=1, y;

    y = (M_NO==3)? a / x :
        (M_NO==2)? a + x :
        (M_NO==1)? a - x :
                      a * x;

    y += b;
    return y;
}
```

1. Define mutation operators \( MOP(x \times y) = \{ x - y, x + y, x/y \} \)
2. Determine whether current expression or statement is affected by mutation
3. Apply mutation operators

Mutants that are not executed cannot be killed
**Mutation Coverage**

```java
public int eval(int x) {
    int a=3, b=1, y;

    y = (M_NO==3)? a / x : 
        (M_NO==2)? a + x : 
        (M_NO==1)? a - x : 
                    a * x;

    y += b;
    return y;
}
```

Mutants that are not executed cannot be killed.
public int eval(int x) {
    int a=3, b=1, y;

    y = (M_NO==3)? a / x : (M_NO==2)? a + x : (M_NO==1)? a - x :
        (M_NO==0 && COVERED(1,3))? a * x : a * x;
        
        y += b;

    return y;
}
Mutation Coverage

```java
public int eval(int x) {
    int a = 3, b = 1, y;

    y = (M_NO == 3) ? a / x :
        (M_NO == 2) ? a + x :
        (M_NO == 1) ? a - x :
        (M_NO == 0 && COVERED(1, 3)) ? a * x :
        a * x;

    y += b;

    return y;
}
```

Mutants that are not executed cannot be killed

Determine covered mutants with additional instrumentation

Only execute and investigate the covered mutants

Just, Kapfhammer, and Schweiggert
Ulm University, Allegheny College

A Compiler-Integrated, Extensible, and Efficient Tool for the Mutation Analysis of Java Programs
MAJOR’s Compiler

Just, Kapfhammer, and Schweiggert

Ulm University, Allegheny College

A Compiler-Integrated, Extensible, and Efficient Tool for the Mutation Analysis of Java Programs
MAJOR’s Compiler

Enhanced Standard Java Compiler
MAJOR’s Compiler

Source Files → MAJOR’s Compiler → Enhanced Standard Java Compiler

Just, Kapfhammer, and Schweiggert
Ulm University, Allegheny College

A Compiler-Integrated, Extensible, and Efficient Tool for the Mutation Analysis of Java Programs
MAJOR’s Compiler

Common Compiler Options

Source Files

MAJOR’s Compiler

Enhanced Standard Java Compiler
MAJOR’s Compiler

Common Compiler Options ➔ Source Files ➔ MAJOR’s Compiler ➔ Enhanced Standard Java Compiler

Domain Specific Language
MAJOR's Compiler

Source Files → MAJOR's Compiler → Bytecode with Embedded Mutants

Common Compiler Options

Domain Specific Language

Enhanced Standard Java Compiler

A Compiler-Integrated, Extensible, and Efficient Tool for the Mutation Analysis of Java Programs

Just, Kapfhammer, and Schweiggert

Ulm University, Allegheny College
Integration into the Java Compiler

```
Compiler

Parse  Attribute  Flow  Lower  Generate
```

Just, Kapfhammer, and Schweiggert
Ulm University, Allegheny College

A Compiler-Integrated, Extensible, and Efficient Tool for the Mutation Analysis of Java Programs
Integration into the Java Compiler

![Diagram of the Java Compiler process]

- Parse
- Attribute
- Flow
- Lower
- Generate

Conditional Mutation

Just, Kapfhammer, and Schweiggert
Ulm University, Allegheny College

A Compiler-Integrated, Extensible, and Efficient Tool for the Mutation Analysis of Java Programs
Integration into the Java Compiler

A Compiler-Integrated, Extensible, and Efficient Tool for the Mutation Analysis of Java Programs
Integration into the Java Compiler

A Compiler-Integrated, Extensible, and Efficient Tool for the Mutation Analysis of Java Programs
Integration into the Java Compiler

What are the challenges with enhancing existing tools?
Integration into the Java Compiler

What are the challenges with enhancing existing tools?
Integration into the Java Compiler

What are the challenges with enhancing existing tools?

Q: “I was wondering if there is a documentation for the Tree transformation. Is the method documentation is all that is available?”

Java compiler developer mailing list
Integration into the Java Compiler

What are the challenges with enhancing existing tools?

Q: “I was wondering if there is a documentation for the Tree transformation. Is the method documentation is all that is available?”

A: “That and looking at the examples embodied in the existing code.”
MAJOR’s Domain Specific Language

// variable declaration
listCOR={&&, ||, ==, !=};
// Define replacement list
BIN(+)<'org'> -> {-, *};
BIN(*)<'org'> -> {/, %};
// Define own operator
myOp{
    BIN(&&) -> listCOR;
    BIN(||) -> listCOR;
    COR;
    LVR;
}
// Enable built-in operator AOR
AOR<'org'>;
// Enable operator myOp
myOp<'java.lang.System@println'>;

Just, Kapfhammer, and Schweiggert
Ulm University, Allegheny College

A Compiler-Integrated, Extensible, and Efficient Tool for the Mutation Analysis of Java Programs
MAJOR’s Domain Specific Language

// variable declaration
listCOR={&&, ||, ==, !=};

// Define replacement list
BIN(<"org">) -> {-, *};

// Define own operator
myOp{
    BIN(&&) -> listCOR;
    BIN(||) -> listCOR;
    COR;
    LVR;
}

// Enable built-in operator AOR
AOR<"org">;

// Enable operator myOp
myOp<"java.lang.System@println">;
MAJOR’s Domain Specific Language

// variable declaration
listCOR={&&, ||, ==, !=};

// Define replacement list
BIN (+)"org" -> {−,∗};
BIN (∗)"org" -> {/,%};

// Define own operator
myOp{
  BIN (&&) -> listCOR;
  BIN (||) -> listCOR;
  COR;
  LVR;
}

// Enable built-in operator AOR
AOR "org";

// Enable operator myOp
myOp "java.lang.System@println";

Specify mutation operators in detail

Define own mutation operator groups
MAJOR’s Domain Specific Language

// variable declaration
listCOR={&&, ||, ==, !=};

// Define replacement list
BIN(+)<"org"> -> {-,*};
BIN(*)<"org"> -> {/,%};

// Define own operator
myOp{
  BIN(&&) -> listCOR;
  BIN(||) -> listCOR;
  COR;
  LVR;
}

// Enable built-in operator AOR
AOR<"org">;

// Enable operator myOp
myOp<"java.lang.System@println">;
Performance Analysis

Overhead for generating and compiling mutants is negligible
Overhead for generating and compiling mutants is negligible
## Performance Analysis

<table>
<thead>
<tr>
<th>Application</th>
<th>Mutants</th>
<th>Runtime of test suite</th>
<th>Memory consumption</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>original</td>
<td>instrumented</td>
</tr>
<tr>
<td>wcs wcs+cov</td>
<td></td>
<td>wcs</td>
<td>wcs+cov</td>
</tr>
<tr>
<td>aspectj</td>
<td>406,382</td>
<td>4.3</td>
<td>4.8</td>
</tr>
<tr>
<td>apache ant</td>
<td>60,258</td>
<td>331.0</td>
<td>335.0</td>
</tr>
<tr>
<td>jfreechart</td>
<td>68,782</td>
<td>15.0</td>
<td>18.0</td>
</tr>
<tr>
<td>itext</td>
<td>124,184</td>
<td>5.1</td>
<td>5.6</td>
</tr>
<tr>
<td>java pathfinder</td>
<td>37,331</td>
<td>17.0</td>
<td>22.0</td>
</tr>
<tr>
<td>commons math</td>
<td>67,895</td>
<td>67.0</td>
<td>83.0</td>
</tr>
<tr>
<td>commons lang</td>
<td>25,783</td>
<td>10.3</td>
<td>11.8</td>
</tr>
<tr>
<td>numerics4j</td>
<td>5,869</td>
<td>1.2</td>
<td>1.3</td>
</tr>
</tbody>
</table>

- Runtime overhead is application dependent
  - Larger for CPU-bound applications
  - Small for I/O-bound applications
  - Even for large projects, applicable on commodity workstations

Just, Kapfhammer, and Schweiggert

Ulm University, Allegheny College

A Compiler-Integrated, Extensible, and Efficient Tool for the Mutation Analysis of Java Programs
## Performance Analysis

<table>
<thead>
<tr>
<th>Application</th>
<th>Mutants</th>
<th>Runtime of test suite</th>
<th>Memory consumption</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>original</td>
<td>instrumented</td>
</tr>
<tr>
<td>aspectj</td>
<td>406,382</td>
<td>4.3</td>
<td>4.8</td>
</tr>
<tr>
<td>apache ant</td>
<td>60,258</td>
<td>331.0</td>
<td>335.0</td>
</tr>
<tr>
<td>jfreechart</td>
<td>68,782</td>
<td>15.0</td>
<td>18.0</td>
</tr>
<tr>
<td>itext</td>
<td>124,184</td>
<td>5.1</td>
<td>5.6</td>
</tr>
<tr>
<td>java pathfinder</td>
<td>37,331</td>
<td>17.0</td>
<td>22.0</td>
</tr>
<tr>
<td>commons math</td>
<td>67,895</td>
<td>67.0</td>
<td>83.0</td>
</tr>
<tr>
<td>commons lang</td>
<td>25,783</td>
<td>10.3</td>
<td>11.8</td>
</tr>
<tr>
<td>numerics4j</td>
<td>5,869</td>
<td>1.2</td>
<td>1.3</td>
</tr>
</tbody>
</table>

- Runtime overhead is application dependent
  - Larger for CPU-bound applications
  - Small for I/O-bound applications
  - Even for large projects, applicable on commodity workstations
Performance Analysis

<table>
<thead>
<tr>
<th>Application</th>
<th>Mutants</th>
<th>Runtime of test suite</th>
<th>Memory consumption</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>original</td>
<td>instrumented</td>
</tr>
<tr>
<td>wcs wcs+cov</td>
<td></td>
<td></td>
<td>WCS</td>
</tr>
<tr>
<td>aspectj</td>
<td>406,382</td>
<td>4.3</td>
<td>4.8</td>
</tr>
<tr>
<td>apache ant</td>
<td>60,258</td>
<td>331.0</td>
<td>335.0</td>
</tr>
<tr>
<td>jfreechart</td>
<td>68,782</td>
<td>15.0</td>
<td>18.0</td>
</tr>
<tr>
<td>itext</td>
<td>124,184</td>
<td>5.1</td>
<td>5.6</td>
</tr>
<tr>
<td>java pathfinder</td>
<td>37,331</td>
<td>17.0</td>
<td>22.0</td>
</tr>
<tr>
<td>commons math</td>
<td>67,895</td>
<td>67.0</td>
<td>83.0</td>
</tr>
<tr>
<td>commons lang</td>
<td>25,783</td>
<td>10.3</td>
<td>11.8</td>
</tr>
<tr>
<td>numerics4j</td>
<td>5,869</td>
<td>1.2</td>
<td>1.3</td>
</tr>
</tbody>
</table>

- Runtime overhead is application dependent
  - Larger for CPU-bound applications
  - Small for I/O-bound applications
- Even for large projects, applicable on commodity workstations
## Performance Analysis

<table>
<thead>
<tr>
<th>Application</th>
<th>Mutants</th>
<th>Runtime of test suite</th>
<th>Memory consumption</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>original</td>
<td>instrumented</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>wcs</td>
</tr>
<tr>
<td>aspectj</td>
<td>406,382</td>
<td>4.3</td>
<td>4.8</td>
</tr>
<tr>
<td>apache ant</td>
<td>60,258</td>
<td>331.0</td>
<td>335.0</td>
</tr>
<tr>
<td>jfreechart</td>
<td>68,782</td>
<td>15.0</td>
<td>18.0</td>
</tr>
<tr>
<td>itext</td>
<td>124,184</td>
<td>5.1</td>
<td>5.6</td>
</tr>
<tr>
<td>java pathfinder</td>
<td>37,331</td>
<td>17.0</td>
<td>22.0</td>
</tr>
<tr>
<td>commons math</td>
<td>67,895</td>
<td>67.0</td>
<td>83.0</td>
</tr>
<tr>
<td>commons lang</td>
<td>25,783</td>
<td>10.3</td>
<td>11.8</td>
</tr>
<tr>
<td>numerics4j</td>
<td>5,869</td>
<td>1.2</td>
<td>1.3</td>
</tr>
</tbody>
</table>

- Runtime overhead is application dependent
  - Larger for CPU-bound applications
  - Small for I/O-bound applications

- Even for large projects, applicable on commodity workstations
## Performance Analysis

<table>
<thead>
<tr>
<th>Application</th>
<th>Mutants</th>
<th>Runtime of test suite</th>
<th>Memory consumption</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>original</td>
<td>instrumented wcs</td>
</tr>
<tr>
<td>aspectj</td>
<td>406,382</td>
<td>4.3</td>
<td>4.8</td>
</tr>
<tr>
<td>apache ant</td>
<td>60,258</td>
<td>331.0</td>
<td>335.0</td>
</tr>
<tr>
<td>jfreechart</td>
<td>68,782</td>
<td>15.0</td>
<td>18.0</td>
</tr>
<tr>
<td>itext</td>
<td>124,184</td>
<td>5.1</td>
<td>5.6</td>
</tr>
<tr>
<td>java pathfinder</td>
<td>37,331</td>
<td>17.0</td>
<td>22.0</td>
</tr>
<tr>
<td>commons math</td>
<td>67,895</td>
<td>67.0</td>
<td>83.0</td>
</tr>
<tr>
<td>commons lang</td>
<td>25,783</td>
<td>10.3</td>
<td>11.8</td>
</tr>
<tr>
<td>numerics4j</td>
<td>5,869</td>
<td>1.2</td>
<td>1.3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>559</td>
<td>813</td>
</tr>
<tr>
<td></td>
<td></td>
<td>237</td>
<td>293</td>
</tr>
<tr>
<td></td>
<td></td>
<td>220</td>
<td>303</td>
</tr>
<tr>
<td></td>
<td></td>
<td>217</td>
<td>325</td>
</tr>
<tr>
<td></td>
<td></td>
<td>182</td>
<td>217</td>
</tr>
<tr>
<td></td>
<td></td>
<td>153</td>
<td>225</td>
</tr>
<tr>
<td></td>
<td></td>
<td>104</td>
<td>149</td>
</tr>
<tr>
<td></td>
<td></td>
<td>73</td>
<td>90</td>
</tr>
</tbody>
</table>

- Runtime overhead is application dependent
  - Larger for CPU-bound applications
  - Small for I/O-bound applications
- Even for large projects, applicable on commodity workstations

---

A Compiler-Integrated, Extensible, and Efficient Tool for the Mutation Analysis of Java Programs
Optimized Mutation Analysis Process

1. Embed and compile all mutants
2. Run test suite on instrumented program
3. Sort tests according to their runtime
4. Perform mutation analysis with reordered test suite
Optimized Mutation Analysis Process

1. Embed and compile all mutants
2. Run test suite on instrumented program
3. Sort tests according to their runtime
4. Perform mutation analysis with reordered test suite
Optimized Mutation Analysis Process

1. Embed and compile all mutants
2. Run test suite on instrumented program
3. Sort tests according to their runtime
4. Perform mutation analysis with reordered test suite

Just, Kapfhammer, and Schweiggert

Ulm University, Allegheny College

A Compiler-Integrated, Extensible, and Efficient Tool for the Mutation Analysis of Java Programs
Optimized Mutation Analysis Process

1. Embed and compile all mutants
2. Run test suite on instrumented program
3. Sort tests according to their runtime
4. Perform mutation analysis with reordered test suite

Just, Kapfhammer, and Schweiggert

A Compiler-Integrated, Extensible, and Efficient Tool for the Mutation Analysis of Java Programs
Performance Analysis

- Mutation analysis is not feasible without coverage information.
- Reordering the test suite significantly speeds up the process, especially if runtimes of tests differ by orders of magnitude.

Just, Kapfhammer, and Schweiggert

Ulm University, Allegheny College

A Compiler-Integrated, Extensible, and Efficient Tool for the Mutation Analysis of Java Programs
Performance Analysis

- Mutation analysis is not feasible without coverage information
- Reordering the test suite significantly speeds up the process, especially if runtimes of tests differ by orders of magnitude

Just, Kapfhammer, and Schweiggert
Ulm University, Allegheny College
A Compiler-Integrated, Extensible, and Efficient Tool for the Mutation Analysis of Java Programs
Performance Analysis

- Mutation analysis is not feasible without coverage information
- Reordering the test suite significantly speeds up the process, especially if runtimes of tests differ by orders of magnitude
Conclusion

Key Concepts and Features:

- Compiler-integrated solution
- Furnishes its own domain specific language
- Provides mutation coverage information
Conclusion

Key Concepts and Features:

- Compiler-integrated solution
- Furnishes its own domain specific language
- Provides mutation coverage information

Characteristics of MAJOR:

- Fast and scalable technique
- Configurable and extensible mutation tool
- Enables an optimized workflow for mutation analysis
MAJOR’s Compiler

So, what comes next in this talk?
MAJOR’s Compiler

So, what comes next in this talk?

Live Demonstration
MAJOR’s Compiler

So, what comes next in this talk?

Using MAJOR standalone

Live Demonstration
MAJOR’s Compiler

So, what comes next in this talk?

Using MAJOR standalone

Live Demonstration

Configuration of MAJOR
MAJOR’s Compiler

So, what comes next in this talk?

Using MAJOR standalone

Live Demonstration

Configuration of MAJOR

Using MAJOR’s DSL
MAJOR’s Compiler

So, what comes next in this talk?

Using MAJOR standalone

Live Demonstration

Integration in Apache Ant

Configuration of MAJOR

Using MAJOR’s DSL
A Compiler-Integrated, Extensible, and Efficient Tool for the Mutation Analysis of Java Programs

Thank you for your attention!

Questions?