Presenter Introduction: Gregory M. Kapfhammer

Efficient and Effective Mutation Testing: Supporting the Implementation of Quality Software by Purposefully Inserting Defects
Inspiration and Motivation

The magic of myth and legend has come true in our time. One types the correct incantation on a keyboard, and a display screen comes to life, showing things that never were nor could be.

Frederick P. Brooks, Jr.
Inspiration and Motivation

The magic of myth and legend has come true in our time. One types the correct incantation on a keyboard, and a display screen comes to life, showing things that never were nor could be.

Frederick P. Brooks, Jr.

In reference to software!
Inspiration and Motivation

I believe the hard part of building software to be the specification, design, and testing of this conceptual construct, not the labor of representing it and testing the fidelity of the representation.

Frederick P. Brooks, Jr.
I believe the hard part of building software to be the specification, design, and testing of this conceptual construct, not the labor of representing it and testing the fidelity of the representation.

Frederick P. Brooks, Jr.

What happens if the “incantation” is incorrect?
I believe the hard part of building software to be the specification, design, and testing of this conceptual construct, not the labor of representing it and testing the fidelity of the representation.

Frederick P. Brooks, Jr.

How do we efficiently and effectively test software?
What is a Test Case?

Method
Under Test
What is a Test Case?

Input → Method Under Test
What is a Test Case?

Input → Method Under Test → Output
What is a Test Case?

- Test Set Up
- Method Under Test
What is a Test Case?

- **Input**: Test Set Up
- **Method Under Test**: Test Set Up

Kapfhammer Allegheny College

Efficient and Effective Mutation Testing: Supporting the Implementation of Quality Software by Purposefully Inserting Defects
What is a Test Case?

- **Input**
- **Method Under Test**
- **Output**

Diagram:
- **Test Set Up**
What is a Test Case?

Input → Method Under Test → Output

Test Set Up → Method Under Test → Clean Up

Kapfhammer
Allegheny College
Efficient and Effective Mutation Testing: Supporting the Implementation of Quality Software by Purposefully Inserting Defects
What is a Test Case?

Input → Method Under Test → Output → Test Oracle → Test Clean Up → Test Set Up → Method Under Test

Test Set Up

Input

Method Under Test

Output

Test Oracle

Test Clean Up

Kapfhammer

Efficient and Effective Mutation Testing: Supporting the Implementation of Quality Software by Purposefully Inserting Defects
What is a Test Case?

Input → Method Under Test → Output → Expected Output → Test Oracle

Test Set Up → Method Under Test → Output → Test Oracle

Test Clean Up
What is a Test Case?

![Test Case Diagram]

- **Input**
- **Method Under Test**
- **Test Set Up**
- **Output**
- **Expected Output**
- **Test Oracle**
- **Test Clean Up**
- **Test Verdict**
What is a Test Case?

Input → Method Under Test → Output

Test Set Up → Test Under Test → Test Clean Up

Expected Output → Test Oracle

Test Verdict
What is a Test Case?

- **Input**
- **Method Under Test**
- **Test Set Up**
- **Output**
- **Test Oracle**
- **Test Clean Up**
- **Expected Output**
- **Test Verdict**

The test case passes and the code is correct!
Software Testing

What is a Test Case?

1. **Input**
2. **Method Under Test**
3. **Expected Output**
4. **Output**
5. **Test Verdict**
6. **Test Clean Up**
7. **Test Oracle**
8. **Test Set Up**
What is a Test Case?

The test case fails and a defect is found!
What is a Test Suite?

$T_1$  $T_2$
What is a Test Suite?

\[ T_1 \quad T_2 \quad T_3 \quad T_4 \]
# What is a Test Suite?

\[ T_1 \quad T_2 \quad T_3 \quad T_4 \quad T_5 \quad T_6 \]
Software Testing

What is a Test Suite?

\[ T_1 \quad T_2 \quad T_3 \quad T_4 \quad T_5 \quad T_6 \quad T_7 \quad T_8 \]
What is a Test Suite?

\[ T_1, T_2, T_3, T_4, T_5, T_6, T_7, T_8, T_9, T_{10} \]
What is a Test Suite?

Test Suite \( T = \langle T_1, T_2, \ldots, T_9, T_{10} \rangle \)
What is a Test Suite?

Test Suite $T = \langle T_1, T_2, \ldots, T_9, T_{10} \rangle$
What is a Test Suite?

Test Suite \( T = \langle T_1, T_2, \ldots, T_9, T_{10} \rangle \)
What is a Test Suite?

Test Suite \( T = \langle T_1, T_2, \ldots, T_9, T_{10} \rangle \)
What is a Test Suite?

Test Suite \( T = \langle T_1, T_2, \ldots, T_9, T_{10} \rangle \)
What is a Test Suite?

Test Suite $T = \langle T_1, T_2, \ldots, T_9, T_{10} \rangle$
What is a Test Suite?

Test Suite $T = \langle T_1, T_2, \ldots, T_9, T_{10} \rangle$
What is a Test Suite?

Test Suite $T = \langle T_1, T_2, \ldots, T_9, T_{10} \rangle$

Requirements $R = \{ R_1, \ldots, R_6 \}$, Features $F = \{ F_1, \ldots, F_4 \}$, Bug Fixes $B = \{ B_1, B_2 \}$
What is a Test Suite?

Test Suite $T = \langle T_1, T_2, \ldots, T_9, T_{10} \rangle$

Requirements $R = \{ R_1, \ldots, R_6 \}$, Features $F = \{ F_1, \ldots, F_4 \}$, Bug Fixes $B = \{ B_1, B_2 \}$
What is a Test Suite?

Test Suite $T = \langle T_1, T_2, \ldots, T_9, T_{10} \rangle$

Requirements $R = \{ R_1, \ldots, R_6 \}$, Features $F = \{ F_1, \ldots, F_4 \}$, Bug Fixes $B = \{ B_1, B_2 \}$
What is a Test Suite?

Test Suite $T = \langle T_1, T_2, \ldots, T_9, T_{10} \rangle$

Requirements $R = \{ R_1, \ldots, R_6 \}$, Features $F = \{ F_1, \ldots, F_4 \}$, Bug Fixes $B = \{ B_1, B_2 \}$
What is a Test Suite?

Test Suite $T = \langle T_1, T_2, \ldots, T_9, T_{10} \rangle$

Requirements $R = \{R_1, \ldots, R_6\}$, Features $F = \{F_1, \ldots, F_4\}$, Bug Fixes $B = \{B_1, B_2\}$
What is a Test Suite?

Test Suite $T = \langle T_1, T_2, \ldots, T_9, T_{10} \rangle$

How Good is Test Suite $T$?

Requirements $R = \{ R_1, \ldots, R_6 \}$, Features $F = \{ F_1, \ldots, F_4 \}$, Bug Fixes $B = \{ B_1, B_2 \}$

Kapfhammer

Efficient and Effective Mutation Testing: Supporting the Implementation of Quality Software by Purposefully Inserting Defects

Allegheny College
What is a Test Suite?

Test Suite $T = \langle T_1, T_2, \ldots, T_9, T_{10} \rangle$

How Good is Test Suite $T$?

Coverage Analysis

Requirements $R = \{ R_1, \ldots, R_6 \}$, Features $F = \{ F_1, \ldots, F_4 \}$, Bug Fixes $B = \{ B_1, B_2 \}$
What is a Test Suite?

Test Suite \( T = \langle T_1, T_2, \ldots, T_9, T_{10} \rangle \)

How Good is Test Suite \( T \)?

Coverage Analysis

Mutation Analysis

Requirements \( R = \{R_1, \ldots, R_6\} \), Features \( F = \{F_1, \ldots, F_4\} \), Bug Fixes \( B = \{B_1, B_2\} \)
Conceptual Faults

if(a > 10)
Conceptual Faults

```java
if(a > 10)
```

```java
if(a >= 10)
```
Conceptual Faults

\[ \text{if}(a > 10) \]

\[ \text{if}(a \geq 10) \]

Implemented
Fundamental Concepts

Conceptual Faults

\[ \text{if}(a > 10) \quad \text{if}(a \geq 10) \]

Implemented

Potential Fault
Conceptual Faults

if(a > 10)

if(a >= 10)

true  false

Kapfhammer Allegheny College

Efficient and Effective Mutation Testing: Supporting the Implementation of Quality Software by Purposefully Inserting Defects
Conceptual Faults

If \( a > 10 \)

\[
\begin{align*}
&\text{true} \\
&\text{false}
\end{align*}
\]

If \( a \geq 10 \)

\[
\begin{align*}
&\text{true} \\
&\text{false}
\end{align*}
\]
Conceptual Faults

- if(a > 10)
- if(a >= 10)
- a = 0
- a

true  false
true  false
Conceptual Faults

- **a**: if \( a > 10 \)
  - true
  - false

- **a = 0**: if \( a >= 10 \)
  - true
  - false
Conceptual Faults

\begin{align*}
\text{if}(a > 10) & \quad \text{if}(a \geq 10) \\
\text{true} & \quad \text{true} \\
\text{false} & \quad \text{false}
\end{align*}
Conceptual Faults

- If $a > 10$
  - True
  - False
- If $a \geq 10$
  - True
  - False

Kapfhammer Allegheny College

Efficient and Effective Mutation Testing: Supporting the Implementation of Quality Software by Purposefully Inserting Defects
Conceptual Faults

- \( if(a > 10) \)
  - true
  - false

- \( if(a \geq 10) \)
  - true
  - false
Conceptual Faults

```
a
if(a > 10)
  true false
```

```
a = 10
if(a >= 10)
  true false
```

```
a
```

Kapfhammer Allegheny College

Efficient and Effective Mutation Testing: Supporting the Implementation of Quality Software by Purposefully Inserting Defects
Conceptual Faults

Can the tests differentiate between *implemented* and *potential fault*?
Conceptual Faults

If yes, then the tests are *adequate*!
Conceptual Faults

If \( a > 10 \), then the tests must be \textit{improved}!
Conceptual Faults

Purposefully insert faults in order to implement quality software!
Overview of Mutation Analysis

Mutation Operator
Overview of Mutation Analysis

Mutation Operator

Mutation Operator
Overview of Mutation Analysis

Mutation Operator

Mutation Operator

Mutation Operator

Mutation Operator
Overview of Mutation Analysis
Overview of Mutation Analysis

Methodically inject small syntactical faults into the program under test.
Overview of Mutation Analysis

- Methodically inject small syntactical faults into the program under test.
Overview of Mutation Analysis

Mutation Operator

Mutation Operator

Mutation Operator

Mutation Operator

Methodically inject small syntactical faults into the program under test

Kapfhammer

Efficient and Effective Mutation Testing: Supporting the Implementation of Quality Software by Purposefully Inserting Defects
Overview of Mutation Analysis

Methodically inject small syntactical faults into the program under test.
Overview of Mutation Analysis

Mutation Operator

Mutation Operator

Mutation Operator

Mutation Operator
Overview of Mutation Analysis

- Test Case $T_1$
- Test Case $T_2$
- Test Case $T_3$
- Test Case $T_4$

Execute the test suite after enabling a single mutant in the program under test.
Overview of Mutation Analysis

Test Case $T_1$ | Test Case $T_2$ | Test Case $T_3$ | Test Case $T_4$

Execute the test suite after enabling a single mutant in the program under test.
Overview of Mutation Analysis

Execute the test suite after enabling a single mutant in the program under test.
Overview of Mutation Analysis

Execute the test suite after enabling a single mutant in the program under test.
Overview of Mutation Analysis

Execute the test suite after enabling a single mutant in the program under test.
Overview of Mutation Analysis

Execute the test suite after enabling a single mutant in the program under test.
Overview of Mutation Analysis

Execute the test suite after enabling a single mutant in the program under test.
Overview of Mutation Analysis

Test Case $T_1$  Test Case $T_2$  Test Case $T_3$  Test Case $T_4$

Execute the test suite after enabling a single mutant in the program under test
Overview of Mutation Analysis

Execute the test suite after enabling a single mutant in the program under test.
## Overview of Mutation Analysis

<table>
<thead>
<tr>
<th>Test Case $T_1$</th>
<th>Test Case $T_2$</th>
<th>Test Case $T_3$</th>
<th>Test Case $T_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The test suite *cannot* kill the mutant – either a test suite weakness or an equivalent mutant!
Overview of Mutation Analysis

Test Case $T_1$  Test Case $T_2$  Test Case $T_3$  Test Case $T_4$

Repeat this process for all of the test cases and mutants – calculate mutation score when finished.
Contributions of this Presentation

Efficient Mutation Analysis
Contributions of this Presentation
Contributions of this Presentation

- Efficient Mutation Analysis
- Challenges
- Solutions
Contributions of this Presentation

Efficient Mutation Analysis

Challenges

Solutions

Conditional Mutation

Kapfhammer
Allegheny College

Efficient and Effective Mutation Testing: Supporting the Implementation of Quality Software by Purposefully Inserting Defects
Contributions of this Presentation

- Efficient Mutation Analysis
- Challenges
- Solutions

Conditional Mutation

Syntax Tree Transformation

Kapfhammer
Allegheny College
Efficient and Effective Mutation Testing: Supporting the Implementation of Quality Software by Purposefully Inserting Defects
Contributions of this Presentation

- Efficient Mutation Analysis
- Challenges
- Solutions
- Conditional Mutation
- Syntax Tree Transformation
- Expressions and Statements

Kapfhammer Allegheny College
Efficient and Effective Mutation Testing: Supporting the Implementation of Quality Software by Purposefully Inserting Defects
Contributions of this Presentation

- Efficient Mutation Analysis
- Challenges
- Solutions
- Conditional Mutation
- Syntax Tree Transformation
- Compiler Integrated
- Expressions and Statements

Efficient and Effective Mutation Testing: Supporting the Implementation of Quality Software by Purposefully Inserting Defects

Kapfhammer
Allegheny College
Contributions of this Presentation

- Comprehensive Empirical Study
- Efficient Mutation Analysis
- Conditional Mutation
- Syntax Tree Transformation
- Compiler Integrated
- Expressions and Statements
- Challenges
- Solutions
Efficient and Effective Mutation Testing: Supporting the Implementation of Quality Software by Purposefully Inserting Defects
Fundamental Concepts

Understanding 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;
}
```
Fundamental Concepts

Understanding 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;
}
```

Methodically inject small syntactical faults into the program under test.
Understanding 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;
}
```

Kapfhammer Allegheny College
Efficient and Effective Mutation Testing: Supporting the Implementation of Quality Software by Purposefully Inserting Defects
Understanding 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;
}
```

- \( y = a - x \)
- \( y = a + x \)
- \( y = a / x \)
- \( \text{if}(b < a) \)
- \( \text{if}(b \neq a) \)
- \( \text{if}(b == a) \)
Understanding 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;
}
```

Unbiased and powerful method for assessing oracles and input values.
Understanding 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;
}
```

Unbiased and powerful method for assessing oracles and input values

Useful method for fault seeding during the empirical study of testing techniques
Mutation Analysis Challenges

Mutant Generation
Mutation Analysis Challenges

Fundamental Concepts

Mutation Analysis

Kapfhammer
Allegheny College

Efficient and Effective Mutation Testing: Supporting the Implementation of Quality Software by Purposefully Inserting Defects
Mutation Analysis Challenges

- Program
- Mutant Generation
- Mutation Operators
- Mutant Generation
- Program
Mutation Analysis Challenges

Fundamental Concepts

Mutation Operators → Program → Mutant Generation → Mutants
Mutation Analysis Challenges

Often Yields a Substantial Number of Mutants
Mutation Analysis Challenges

Program → Mutant Generation → Mutants

Often Yields a Substantial Number of Mutants

High Time Overhead for Generation

Kapfhammer Allegheny College

Efficient and Effective Mutation Testing: Supporting the Implementation of Quality Software by Purposefully Inserting Defects
Mutation Analysis Challenges

- Program
- Mutant Generation
- Mutants
- Mutation Analysis
- High Time Overhead for Generation
- Often Yields a Substantial Number of Mutants

Fundamental Concepts
Mutation Analysis Challenges

- Program
- Mutant Generation
- Tests
- Mutants
- Mutation Analysis

High Time Overhead for Generation

Often Yields a Substantial Number of Mutants
Mutation Analysis Challenges

Program → Mutant Generation → Mutants → Mutation Analysis → Results

- Often Yields a Substantial Number of Mutants
- High Time Overhead for Generation

Introduction
Mutation Analysis
Empirical Evaluation
Conclusion

Fundamental Concepts

Mutation Analysis Challenges

- Mutant Generation
- Mutation Operators
- Tests

Kapfhammer
Allegheny College

Efficient and Effective Mutation Testing: Supporting the Implementation of Quality Software by Purposefully Inserting Defects
Mutation Analysis Challenges

- High Time Overhead for Generation
- Individually Executing the Mutants is Too Expensive

```
Program

Mutation Operators

Mutant Generation

Tests

Mutants

Mutation Analysis

Often Yields a Substantial Number of Mutants

Results
```
Mutation Analysis Challenges

- Prior Solutions?
- Mutation Operators
- Program
- Mutant Generation
- Mutants
- Tests
- Mutation Analysis
- Results

- Often Yields a Substantial Number of Mutants
- High Time Overhead for Generation
- Individually Executing the Mutants is Too Expensive

Kapfhammer
Allegheny College
Efficient and Effective Mutation Testing: Supporting the Implementation of Quality Software by Purposefully Inserting Defects
Prior Work in Mutation Analysis

Improving Mutation Analysis
Prior Work in Mutation Analysis

Improving Mutation Analysis

Offutt and Untch
Prior Work in Mutation Analysis

Improving Mutation Analysis

Do Fewer

Offutt and Untch
Prior Work in Mutation Analysis

- Improving Mutation Analysis
- Do Fewer
  - Sampling
  - Selection

Offutt and Untch
Prior Work in Mutation Analysis

Improving Mutation Analysis

Offutt and Untch

Do Fewer

Do Smarter

Kapfhammer
Allegheny College
Efficient and Effective Mutation Testing: Supporting the Implementation of Quality Software by Purposefully Inserting Defects
Prior Work in Mutation Analysis

- Improving Mutation Analysis
  - Do Fewer
  - Distributed
- Offutt and Untch
  - Do Smarter
  - Weak Mutation
Prior Work in Mutation Analysis

Improving Mutation Analysis

Do Fewer

Do Smarter

Do Faster

Offutt and Untch
Prior Work in Mutation Analysis

- Improving Mutation Analysis
  - Do Fewer
  - Do Smarter
  - Do Faster
    - Compiler Integrated
    - Bytecode Transformation
    - Mutant Schemata
- Offutt and Untch
Prior Work in Mutation Analysis

- Offutt and Untch: Do Fewer, Do Smarter
- Jia and Harman: Do Faster
- Higher Order Mutation
Conditional Mutation
Conditional Mutation

Encapsulates all mutants within the same block
Conditional Mutation

- Encapsulates all mutants within the same block
- Can be integrated within the compiler
Mutation Analysis with MAJOR

Conditional Mutation

- Encapsulates all mutants within the same block
- Transforms the abstract syntax tree (AST)
- Can be integrated within the compiler
Conditional Mutation

- Encapsulates all mutants within the same block
- Transforms the abstract syntax tree (AST)
- Can be integrated within the compiler

Stmt $\rightarrow$ Conditional Stmt (if-then-else, switch)

Expr $\rightarrow$ Conditional Expr (conditional operator ?::)
Transforming the AST

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

    y = a * x;

    y += b;
    return y;
}
```
Transforming the AST

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

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

\[
\text{ASSIGN}
\]
\[
\text{IDENT}
\quad \text{BINARY}
\]
\[
\text{ASSIGN}
\quad \text{IDENT}
\quad \text{BINARY}
\]
\[
y = a \times x
\]
Transforming the AST

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

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

![AST diagram]
Transforming the AST

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

![Diagram showing the transformation of the AST tree](image)
Mutation Analysis with MAJOR

Source Code View of Inserting Mutants

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

    y = a * x;

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

1. Define mutation operators \( MOP(x \ast y) = \{x - y, x + y, x/y\} \)
2. Determine whether current expression or statement is affected by mutation
3. Apply mutation operators
Source Code View of Inserting Mutants

```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
Source Code View of Inserting Mutants

```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

Efficient and Effective Mutation Testing: Supporting the Implementation of Quality Software by Purposefully Inserting Defects
Source Code View of Inserting Mutants

```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 \( MOP(x \cdot y) = \{x - y, x + y, x/y\} \)
2. Determine whether current expression or statement is affected by mutation
3. Apply mutation operators
Mutation Analysis with MAJOR

Source Code View of Inserting Mutants

```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
Source Code View of Inserting Mutants

```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

Kapfhammer
Allegheny College

Efficient and Effective Mutation Testing: Supporting the Implementation of Quality Software by Purposefully Inserting Defects
Source Code View of Inserting Mutants

```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

Mutants that are not executed cannot be killed
Collecting and Using 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

Kapfhammer
Allegheny College

Efficient and Effective Mutation Testing: Supporting the Implementation of Quality Software by Purposefully Inserting Defects
Collecting and Using 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
Mutation Analysis

Collecting and Using 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

Efficient and Effective Mutation Testing: Supporting the Implementation of Quality Software by Purposefully Inserting Defects
MAJOR’s Compiler
MAJOR’s Compiler

Enhanced the Standard Java Compiler
MAJOR’s Compiler

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

Efficient and Effective Mutation Testing: Supporting the Implementation of Quality Software by Purposefully Inserting Defects
MAJOR’s Compiler

Common Compiler Options

Source Files

MAJOR’s Compiler

Enhanced the Standard Java Compiler

Efficient and Effective Mutation Testing: Supporting the Implementation of Quality Software by Purposefully Inserting Defects
MAJOR’s Compiler

Source Files → Common Compiler Options → MAJOR’s Compiler → Domain Specific Language → Enhanced the Standard Java Compiler

Efficient and Effective Mutation Testing: Supporting the Implementation of Quality Software by Purposefully Inserting Defects
MAJOR’s Compiler

Common Compiler Options

Domain Specific Language

Source Files

MAJOR’s Compiler

Bytecode with Embedded Mutants

Enhanced the Standard Java Compiler

Kapfhammer
Allegheny College
Efficient and Effective Mutation Testing: Supporting the Implementation of Quality Software by Purposefully Inserting Defects
Integration into the Java Compiler
Integration into the Java Compiler

![Compiler Diagram]

- Parse
- Attribute
- Flow
- Lower
- Generate

Conditional Mutation
Mutation Analysis with MAJOR

Integration into the Java Compiler

Compiler

Parse → Attribute → Flow → Lower → Generate

Conditional Mutation

Configuration

Compiler options

Domain specific language

<< use >>
Integration into the Java Compiler

Mutation Analysis with MAJOR

Kapfhammer Allegheny College

Efficient and Effective Mutation Testing: Supporting the Implementation of Quality Software by Purposefully Inserting Defects
MAJOR’s Domain Specific Language

```java
// 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">;
```
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";
MAJOR’s Domain Specific Language

```java
// 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

```java
// 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">;
```
### 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

---

**Kapfhammer Allegheny College**

Efficient and Effective Mutation Testing: Supporting the Implementation of Quality Software by Purposefully Inserting Defects
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
Compilation Efficiency

Mutant Generation and Compilation

Overhead for generating and compiling mutants is negligible

Kapfhammer Allegheny College

Efficient and Effective Mutation Testing: Supporting the Implementation of Quality Software by Purposefully Inserting Defects
Compilation Efficiency

Mutant Generation andCompilation

Overhead for generating and compiling mutants is negligible

Kapfhammer
Allegheny College

Efficient and Effective Mutation Testing: Supporting the Implementation of Quality Software by Purposefully Inserting Defects
## Time and Space Overhead

<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
Time and Space Overhead

<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
### Time and Space Overhead

<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
### Time and Space Overhead

<table>
<thead>
<tr>
<th>Application</th>
<th>Mutants</th>
<th>Runtime of test suite original</th>
<th>Instrumented wcs</th>
<th>Instrumented wcs + cov</th>
<th>Memory consumption original</th>
<th>Instrumented</th>
</tr>
</thead>
<tbody>
<tr>
<td>aspectj</td>
<td>406,382</td>
<td>4.3</td>
<td>4.8</td>
<td>5.0</td>
<td>559</td>
<td>813</td>
</tr>
<tr>
<td>apache ant</td>
<td>60,258</td>
<td>331.0</td>
<td>335.0</td>
<td>346.0</td>
<td>237</td>
<td>293</td>
</tr>
<tr>
<td>jfreechart</td>
<td>68,782</td>
<td>15.0</td>
<td>18.0</td>
<td>23.0</td>
<td>220</td>
<td>303</td>
</tr>
<tr>
<td>itext</td>
<td>124,184</td>
<td>5.1</td>
<td>5.6</td>
<td>6.3</td>
<td>217</td>
<td>325</td>
</tr>
<tr>
<td>java pathfinder</td>
<td>37,331</td>
<td>17.0</td>
<td>22.0</td>
<td>29.0</td>
<td>182</td>
<td>217</td>
</tr>
<tr>
<td>commons math</td>
<td>67,895</td>
<td>67.0</td>
<td>83.0</td>
<td>98.0</td>
<td>153</td>
<td>225</td>
</tr>
<tr>
<td>commons lang</td>
<td>25,783</td>
<td>10.3</td>
<td>11.8</td>
<td>14.8</td>
<td>104</td>
<td>149</td>
</tr>
<tr>
<td>numerics4j</td>
<td>5,869</td>
<td>1.2</td>
<td>1.3</td>
<td>1.6</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
## Time and Space Overhead

<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
Evaluating and Improving Mutation 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.
Mutation Analysis Efficiency

Evaluating and Improving Mutation 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.
Evaluating and Improving Mutation 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
Improving Test Suite Quality

Mutation Analysis

Kapfhammer Allegheny College

Efficient and Effective Mutation Testing: Supporting the Implementation of Quality Software by Purposefully Inserting Defects
Improving Test Suite Quality

1. **Program** → **Mutation Analysis** → **Test Suite**

Kapfhammer

Efficient and Effective Mutation Testing: Supporting the Implementation of Quality Software by Purposefully Inserting Defects
Improving Test Suite Quality

Program → Mutation Analysis → Test Suite

Mutation Score

Kapfhammer
Allegheny College
Efficient and Effective Mutation Testing: Supporting the Implementation of Quality Software by Purposefully Inserting Defects
Improving Test Suite Quality

1. Program → Mutation Analysis → Mutation Score → Test Suite
2. Improve Tests

Kapfhammer
Allegheny College

Efficient and Effective Mutation Testing: Supporting the Implementation of Quality Software by Purposefully Inserting Defects
Improving Test Suite Quality

- Program → Mutation Analysis → Mutation Score → Test Suite
- Improve Tests:
  - Automated
  - Manual

Efficient and Effective Mutation Testing: Supporting the Implementation of Quality Software by Purposefully Inserting Defects
Improving Test Suite Quality

Efficient and Effective Mutation Testing: Supporting the Implementation of Quality Software by Purposefully Inserting Defects
Improving Test Suite Quality

Program → Mutation Analysis → Test Suite

Mutation Score

Improve Tests → Use Tests

Test improvement is only effective if mutation analysis is efficient!

Kapfhammer

Efficient and Effective Mutation Testing: Supporting the Implementation of Quality Software by Purposefully Inserting Defects
# Reviewing MAJOR’s Contributions

<table>
<thead>
<tr>
<th>Mutation Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kapfhammer Allegheny College</td>
</tr>
</tbody>
</table>

Efficient and Effective Mutation Testing: Supporting the Implementation of Quality Software by Purposefully Inserting Defects
Reviewing MAJOR’s Contributions

**Efficiency:** MAJOR has acceptable time and space overheads and scales to large, real-world programs.
Reviewing MAJOR’s Contributions

**Efficiency**: MAJOR has acceptable time and space overheads and scales to large, real-world programs

**Usability**: MAJOR’s integration into the Java SE compiler makes it a no-hassle, drop-in tool
Reviewing MAJOR’s Contributions

**Efficiency**: MAJOR has acceptable time and space overheads and scales to large, real-world programs

**Usability**: MAJOR’s integration into the Java SE compiler makes it a no-hassle, drop-in tool

We will release MAJOR as free and open source software
Conclusion

Key Concepts and Features:

- Compiler-integrated solution
- Conditional mutation with the abstract syntax tree
- Furnishes its own domain specific language
- Collects and leverages mutation coverage information
Conclusion

Key Concepts and Features:

- Compiler-integrated solution
- Conditional mutation with the abstract syntax tree
- Furnishes its own domain specific language
- Collects and leverages mutation coverage information

Characteristics of MAJOR:

- Fast and scalable technique
- Configurable and extensible mutation tool
- Enables an optimized workflow for mutation analysis
Recently Published Papers


Efficient and Effective Mutation Testing: Supporting the Implementation of Quality Software by Purposefully Inserting Defects

Gregory M. Kapfhammer

Department of Computer Science
Allegheny College
http://www.cs.allegheny.edu/~gkapfham/

Thank you for your attention!
I welcome your questions and comments.