



University of
Sheffield



Exploring Pseudo-Testedness

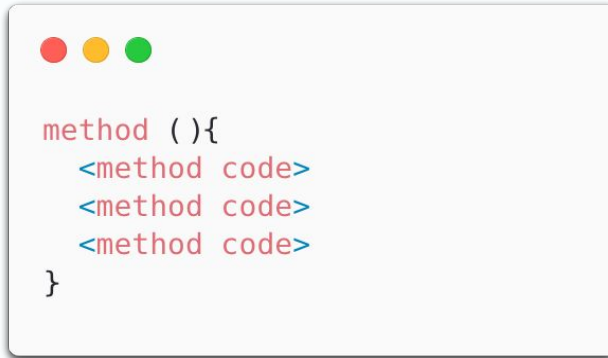
Empirically Evaluating Extreme Mutation
Testing at the Statement Level

Megan Maton*, Gregory M. Kapfhammer, Phil McMinn

* **Lead author and paper presenter**

Extreme Mutation Testing (XMT)

Production Code



```
method (){\n  <method code>\n  <method code>\n  <method code>\n}
```

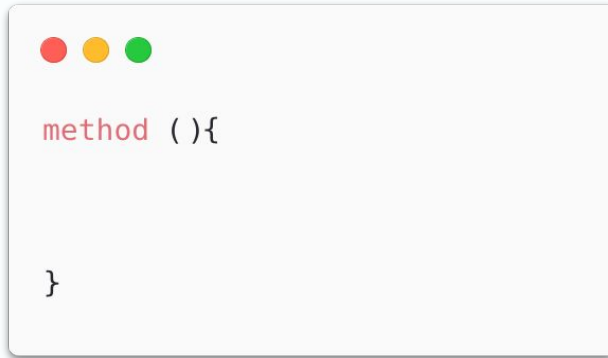
Test Suite



This method is **pseudo-tested**.

Extreme Mutation Testing (XMT)

Production Code



```
method (){  
  
}
```

Test Suite



This method is “**required**” for the test suite to pass.

Does pseudo-testedness
exist within these required
methods?

Identifying pseudo-tested statements

Required Method

```
public String formatTag(String tag, String content) {  
    String html = "<" + tag + ">";  
    html += content;  
    html += "</" + tag + ">";  
    return html;  
}
```

Test Suite

```
@Test  
public void testFormatTag() {  
    String str = formatTag("p", "hello world!");  
    assertThat(str, startsWith("<p>"));  
    assertThat(str, endsWith("</p>"));  
}
```

Identifying pseudo-tested statements

Required Method

```
public String formatTag(String tag, String content) {  
    String html = "<" + tag + ">";  
    html += content;  
    html += "</" + tag + ">";  
    return html;  
}
```

Test Suite

```
@Test  
public void testFormatTag() {  
    String str = formatTag("p", "hello world!");  
    assertThat(str, startsWith("<p>"));  
    assertThat(str, endsWith("</p>"));  
}
```

Identifying pseudo-tested statements

Required Method

```
public String formatTag(String tag, String content) {  
    String html = "<" + tag + ">";  
    html += content;  
    html += "</" + tag + ">";  
    return html;  
}
```

Test Suite

```
@Test  
public void testFormatTag() {  
    String str = formatTag("p", "hello world!");  
    assertThat(str, startsWith("<p>"));  
    assertThat(str, endsWith("</p>"));  
}
```

Identifying pseudo-tested statements

Required Method

```
public String formatTag(String tag, String content) {  
    String html = "<" + tag + ">";  
    html += content;  
    html += "</" + tag + ">";  
    return html;  
}
```

Test Suite

```
@Test  
public void testFormatTag() {  
    String str = formatTag("p", "hello world!");  
    assertThat(str, startsWith("<p>"));  
    assertThat(str, endsWith("</p>"));  
}
```


Identifying pseudo-tested statements

Required Method

```
public String formatTag(String tag, String content) {  
    String html = "<" + tag + ">";  
    html += content;  
    html += "</" + tag + ">";  
    return html;  
}
```

Test Suite

```
@Test  
public void testFormatTag() {  
    String str = formatTag("p", "hello world!");  
    assertThat(str, startsWith("<p>"));  
    assertThat(str, endsWith("</p>"));  
}
```

Identifying pseudo-tested statements

Required Method

```
public String formatTag(String tag, String content) {  
    String html = "<" + tag + ">";  
    html += content;  
    html += "</" + tag + ">";  
    return html;  
}
```

Test Suite

```
@Test  
public void testFormatTag() {  
    String str = formatTag("p", "hello world!");  
    assertThat(str, startsWith("<p>"));  
    assertThat(str, endsWith("</p>"));  
}
```

Identifying pseudo-tested statements

Required Method

```
public String formatTag(String tag, String content) {  
    String html = "<" + tag + ">";  
    html += content;  
    html += "</" + tag + ">";  
    return html;  
}
```

Test Suite

```
@Test  
public void testFormatTag() {  
    String str = formatTag("p", "hello world!");  
    assertThat(str, startsWith("<p>"));  
    assertThat(str, endsWith("</p>"));  
}
```

Identifying pseudo-tested statements

Required Method



```
public String formatTag(String tag, String content) {  
    String html = "<" + tag + ">";  
    html += "</" + tag + ">";  
    return html;  
}
```

Test Suite



```
@Test  
public void testFormatTag() {  
    String str = formatTag("p", "hello world!");  
    assertThat(str, startsWith("<p>"));  
    assertThat(str, endsWith("</p>"));  
}
```

Identifying pseudo-tested statements

Required Method

```
public String formatTag(String tag, String content) {  
    String html = "<" + tag + ">";  
    html += "</" + tag + ">";  
    return html;  
}
```

Test Suite

```
@Test  
public void testFormatTag() {  
    String str = formatTag("p", "hello world!");  
    assertThat(str, startsWith("<p>"));  
    assertThat(str, endsWith("</p>"));  
}
```

Despite the method being “required”, it contains a pseudo-tested statement...

Research Questions

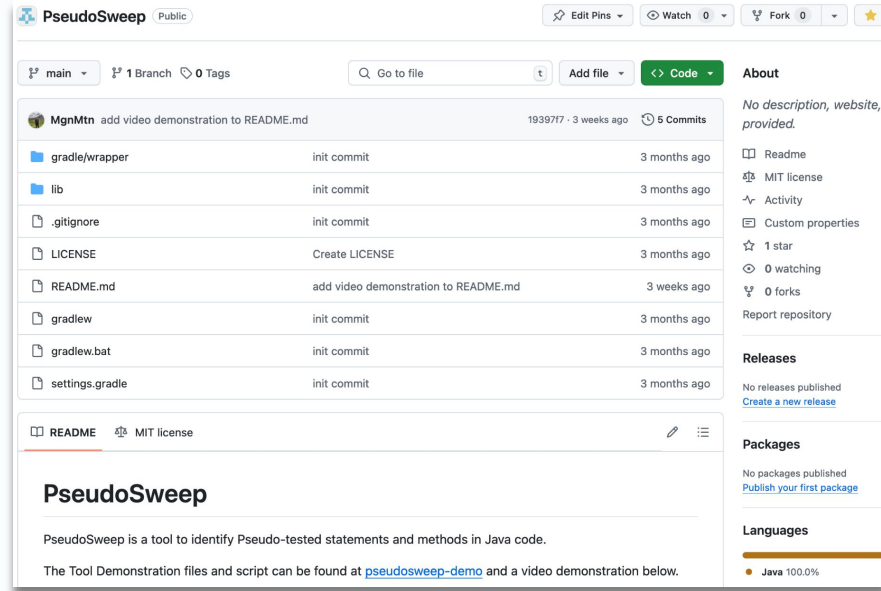
RQ1: How **frequent** are pseudo-tested elements?

RQ2: Do pseudo-tested elements have **low mutation scores**?

RQ3: Does PIT's default set of operators effectively highlight deficient testing with respect to pseudo-tested statements?

RQ4: What are the **causes** of pseudo-tested statements?

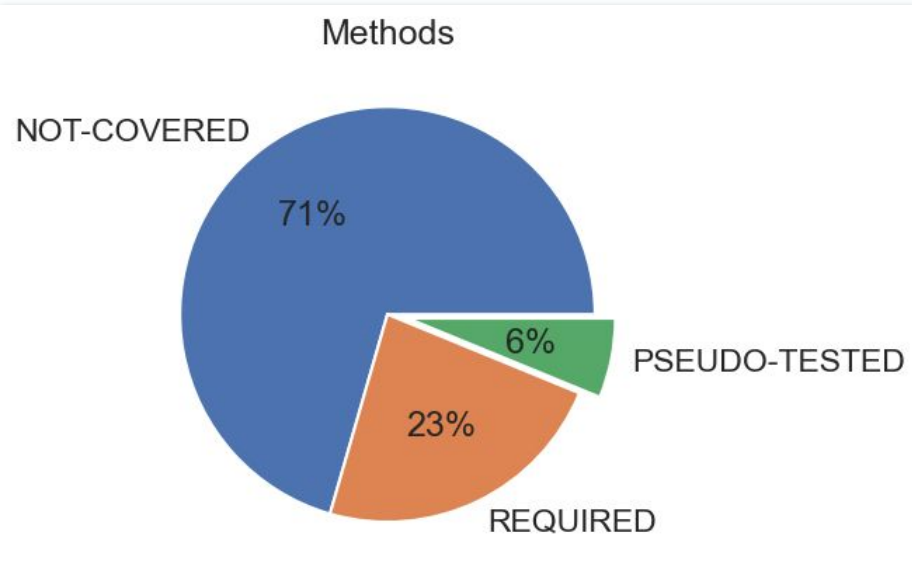
PseudoSweep: A Pseudo-Tested Code Identifier



16:00 Tool Demo in Fremont

<https://github.com/PseudoTested/PseudoSweep>

RQ1: How **frequent** are pseudo-tested elements?



7 projects contained no pseudo-tested methods

6.1% of all methods were pseudo-tested

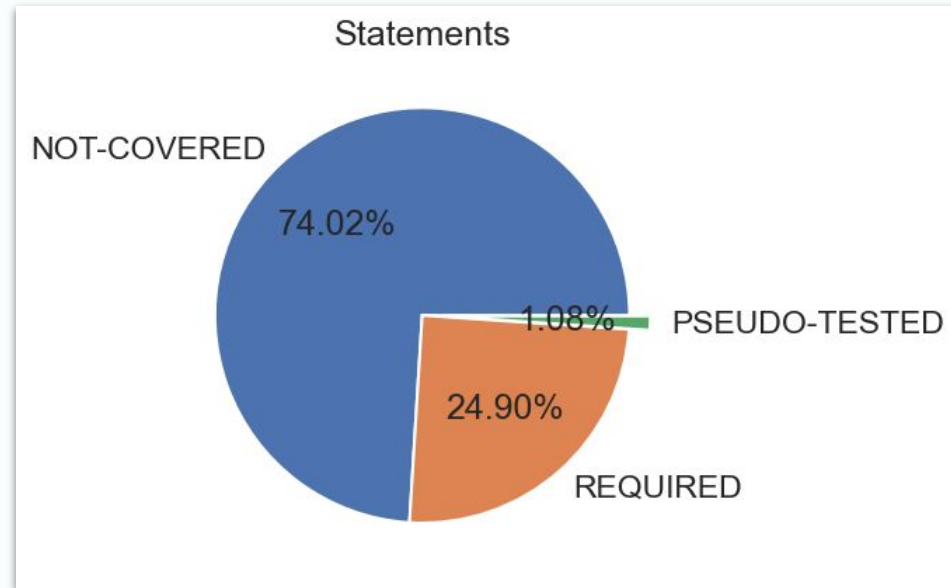
21% of covered (required + pseudo-tested) methods were actually pseudo-tested

RQ1: How **frequent** are pseudo-tested elements?

1.08% of statements
pseudo-tested

4% of covered statements were
pseudo-tested

Pseudo-tested statements **in
required** methods made up
48% of pseudo-tested
statements



RQ2 and 3: Background

Mutation Testing seeds synthetic faults into code to evaluate a test suites fault detection ability.

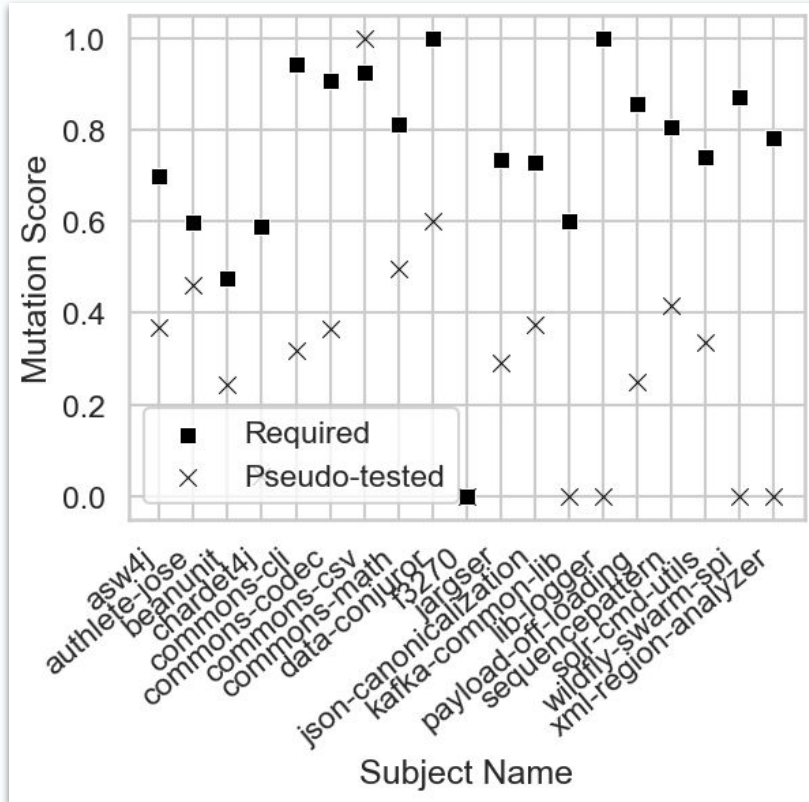
$a > b$

mutates to

$a < b$

If your test suite can detect the synthetic faults, it will likely detect real faults.

RQ2: Do pseudo-tested elements have **low mutation scores**?



Overall method mutation scores

■ 0.82 (Required)

× 0.40 (Pseudo-tested)

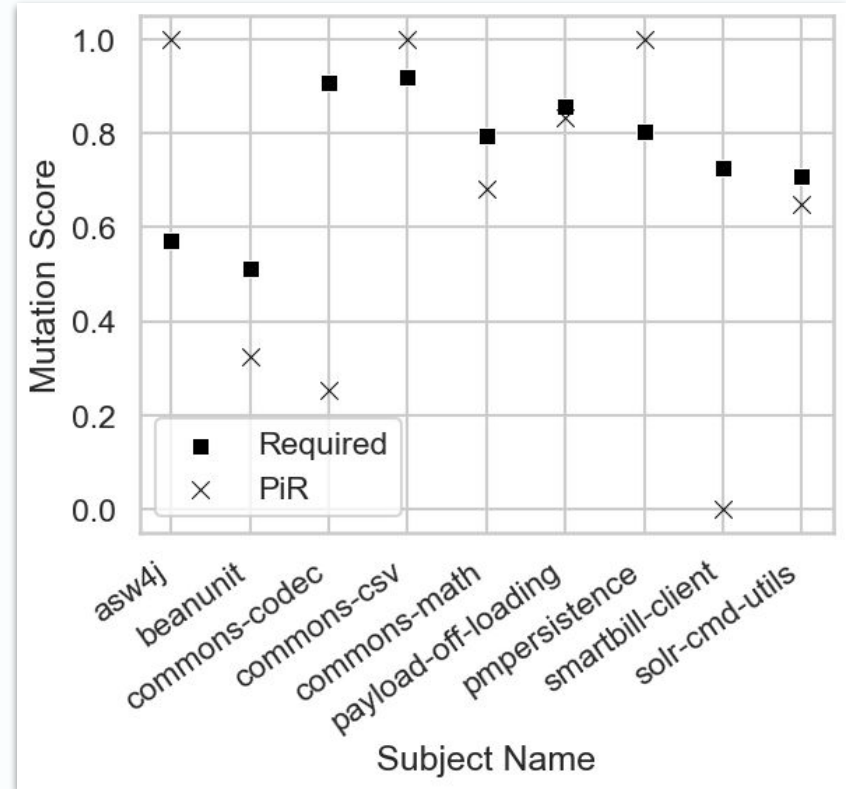
RQ2: Do pseudo-tested elements have **low mutation scores**?

Overall statement mutation scores

■ 0.80 (Required)

× 0.57 (PiR)

PiR - Pseudo-tested statements within required methods



RQ3: Does **PIT**'s default set of operators effectively highlight deficient testing with respect to pseudo-tested statements?



State-of-the-art Mutation
Testing Tool for Java projects

RQ3: Does PIT's **default set of operators** effectively highlight deficient testing with respect to pseudo-tested statements?

Default Operator Set

Mutators	Void Method Calls
Conditionals Boundary	Empty returns
Increments	False Returns
Invert Negatives	True returns
Math	Null returns
Negate Conditionals	Primitive returns



RQ3: Does PIT's **default set of operators** effectively highlight deficient testing with respect to pseudo-tested statements?

Elements	Mutants per Element
Required Methods	6.97
Pseudo-tested Methods	3.67
Required Statements	2.31
PiR Statements	0.92

PIT places more mutants in required methods than pseudo-tested methods

PIT places less than one mutant per PiR statement

Testing deficiencies in pseudo-tested statements may have a lower chance of being identified using PIT's default operator set alone.

RQ4: What are the **causes** of pseudo-tested statements?

⊙ No targeting assertion (**70**)

• Partial assertion (**7**)

⊘ No targeting test (**9**)

↪ Unintended exception handling (**6**)

RQ4: © No targeting assertion (70)

Production Code

```
public String formatTag(String tag, String content) {  
    String html = "<" + tag + ">";  
    html += content;  
    html += "</" + tag + ">";  
    return html;  
}
```

Test Suite



```
@Test  
public void testFormatTag() {  
    String str = formatTag("p", "hello world!");  
    assertThat(str, startsWith("<p>"));  
    assertThat(str, endsWith("</p>"));  
}
```

Solution: add an assertion to check expected content is between the tags

RQ4: Partial assertion (7)

```
1  @Override
2  public String toString() {
3      final StringBuilder buf = new StringBuilder();
4      buf.append("[ Options: [ short ");
5      buf.append(shortOpts.toString());
6      buf.append(" ] [ long ");
7      buf.append(longOpts);
8      buf.append(" ]");
9      return buf.toString();
10 }
```

Line 5 and 7 are pseudo-tested due to partial assertion.

And due to both shortOpts and longOpts containing "foo" and "bar"...

```
@Test
public void testToString() {
    final Options options = new Options();
    options.addOption("f", "foo");
    options.addOption("b", "bar");

    final String s = options.toString();
    assertNotNull("null string returned", s);
    assertTrue("foo option missing", s.toLowerCase().contains("foo"));
    assertTrue("bar option missing", s.toLowerCase().contains("bar"));
}
```

Test only checks the final string *contains* "foo" and "bar".

RQ4: No targeting test (9)

```
1 public Boolean isValidInput(String input){
2
3     if(input.equals(/* Valid Input */)) {
4         return true;
5     }
6
7     throw InvalidInputException();
8 }
```

Making the IF
statement
pseudo-tested

There is a test
checking the
exception is thrown

But no test to execute
the IF Statement to
True

```
1 @Test(expected = InvalidInputException.class)
2 public void testInvalidInput(){
3     isValidInput(/* Invalid Input */);
4 }
```

RQ4: Unintended exception handling (6)

```
1 public Boolean isValidInput(String input){
2
3     if(input.equals(/* Valid Input */)) {
4         return true;
5     }
6
7     throw InvalidInputException();
8 }
```

Checks if input string is a valid input

And throws an exception if it is not valid

```
1 @Test(expected = InvalidInputException.class)
2 public void testValidInputs() {
3     assertTrue(isValidInput(/* Valid Input */));
4
5     assertTrue(isValidInput(/* Invalid Input */));
6 }
```

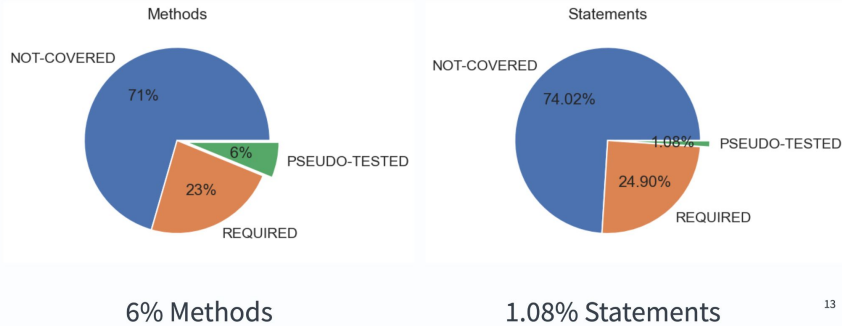
Replication Package

The screenshot shows the GitHub interface for the repository 'icsme-2024-replication-package'. At the top, the repository name is displayed with a 'Public' badge, along with 'Edit Pins' and 'Watch 0' options. Below this, the current branch is 'main', with '1 Branch' and '0 Tags' indicated. A search bar 'Go to file' and buttons for 'Add file' and 'Code' are visible. The commit history shows a recent commit by 'MgnMtn' titled 'add replication package' with hash '98270a0' and '2 Commits'.

File	Commit Message	Time
extract-data	add replication package	1 minute ago
output-data	add replication package	1 minute ago
.gitignore	add replication package	1 minute ago
LICENSE	Initial commit	2 months ago
README.md	add replication package	1 minute ago
RQ1.ipynb	add replication package	1 minute ago
RQ2and3.ipynb	add replication package	1 minute ago
project-characterisation.ipynb	add replication package	1 minute ago
projects.md	add replication package	1 minute ago

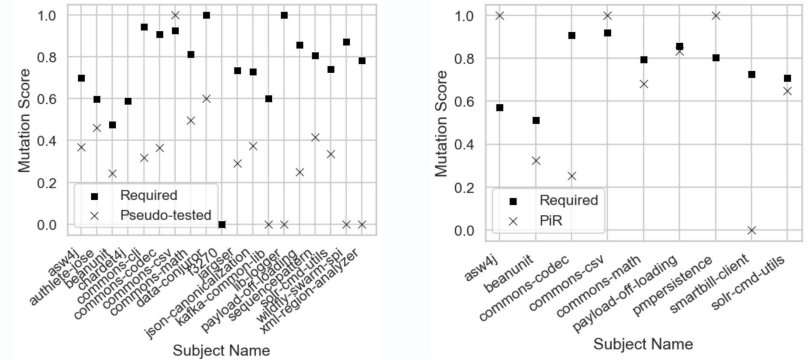
<https://github.com/PseudoTested/icsme-2024-replication-package>

RQ1: How frequent are pseudo-tested elements?



13

RQ2: Do pseudo-tested elements have low mutation scores?



14

RQ3: Does PIT's default set of operators effectively highlight deficient testing with respect to pseudo-tested statements?

Elements	Mutants per Element
Required Methods	6.97
Pseudo-tested Methods	3.67
Required Statements	2.31
PIR Statements	0.92

PIT places more mutants in required methods than pseudo-tested methods

PIT places less than one mutant per PIR statement

Testing deficiencies in pseudo-tested statements may have a lower chance of being identified using PIT's default operator set alone.

24

RQ4: What are the causes of pseudo-tested statements?

- ⊙ No targeting assertion (70)
- ◐ Partial assertion (7)
- ⊘ No targeting test (9)
- ⤵ Unintended exception handling (6)

25

31