



Perfect Is the Enemy of Test Oracle



Ali Reza Ibrahimzada¹, Yigit Varli², Dilara Tekinoglu³, Reyhaneh Jabbarvand¹

¹University of Illinois Urbana-Champaign, ²Middle East Technical University, ³University of Massachusetts Amherst

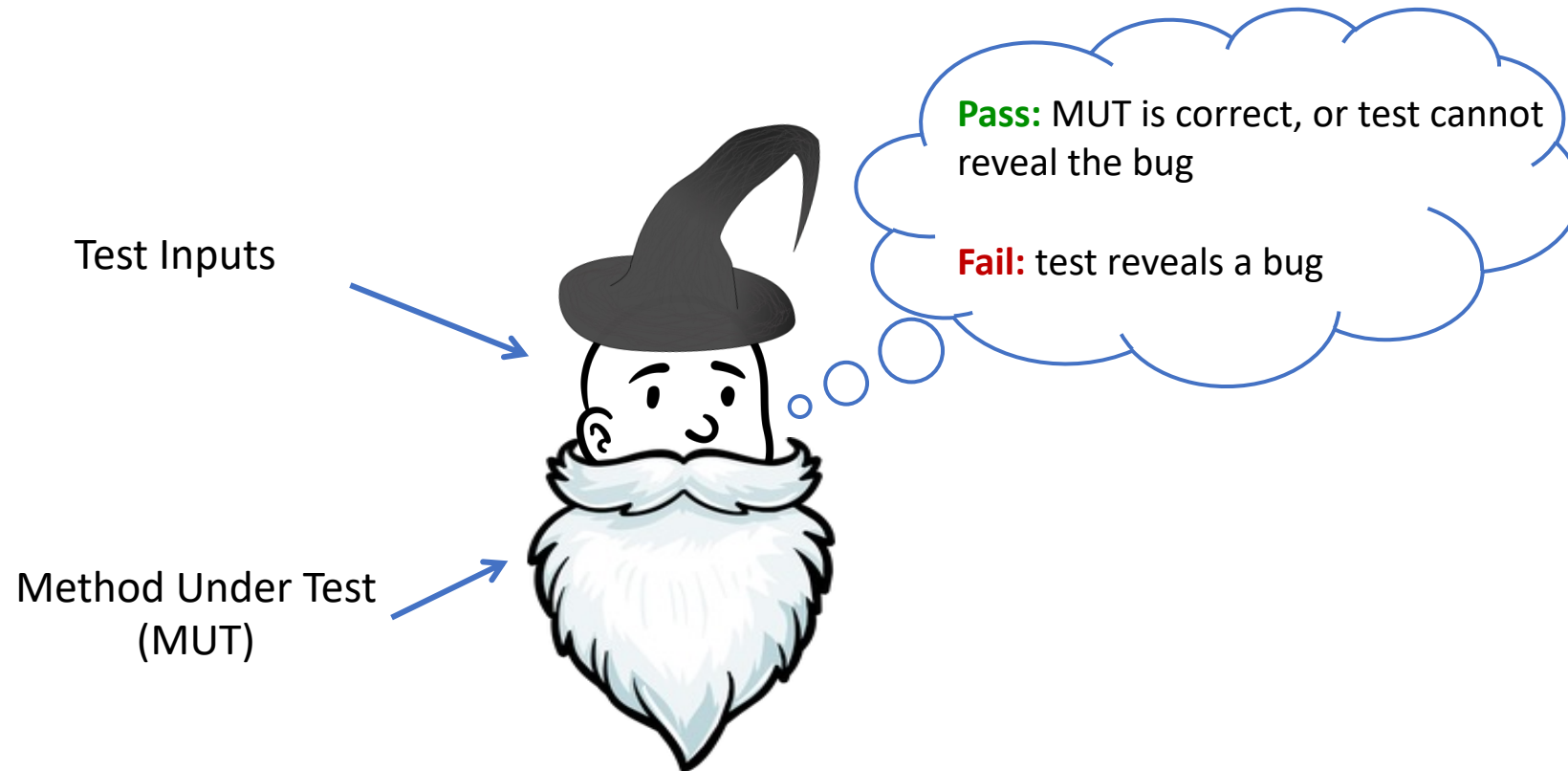


Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)

Singapore, November 2022

What is a Test Oracle?

Does execution of a test reveal a bug in the code?



Automated Test Oracle Construction

❖ Formal Specifications

❖ Assertions

❖ Program Invariants

❖ **Commonality -> determining the EXACT PROGRAM BEHAVIOR**

❖ Metamorphic Relations

Available online at www.sciencedirect.com

Science of Computer Programming
www.elsevier.com/locate/scico

On the Use of Specifications

David Coppit
Department of Computer Science
The College of William and Mary
Williamsburg, VA 23185
david@coppit.org

Abstract

The "oracle problem" is a well-known problem in software testing. Without some means of automatically computing the correct answer for test cases, instead of revealing software faults by analyzing code with complete, specification-based assertion evaluation method is to (1) develop a formal specification, (2) translate this specification into assertions or identify existing faults, and (4) for each version of the software, using a set of test inputs and check for assertion violations. Our goal is to determine whether specification-driven assertions are a viable method of revealing faults in software. Our evaluation is based on two real-world software systems. Our results show that specification-driven assertions can be used to reduce the cost of testing. We also discuss the costs associated with the application of these techniques for reducing these costs.

1. Introduction

Testing is a widely used approach for program development. The primary goal of testing is to identify software during development to help ensure that the software will not fail after deployment. A test case is an input and expected output, and the system is correct for that input if the actual output matches the expected output. Despite the fact that testing proves correctness for only those test cases that pass, it is popular due in part to its low, incremental cost. A key challenge for testing is known as the "oracle problem"—the need for some trustworthy method of determining the correct output for a given test case.

Proceedings of the 2005 29th Annual IEEE/ACM Software Engineering Conference (ICSE 2005), Boston, MA, USA, October 23-29, 2005. Copyright © 2005 IEEE. Authorized licensed use limited to: University of Illinois. All rights reserved.

0-7695-2306-4/05 \$20.00 © 2005 IEEE

416

Randoop: Feedback-Directed Random Testing for Java

Carlos Pacheco, Michael D. Ernst
MIT CSAIL
{cpacheco, mernst}@csail.mit.edu

Abstract

RANDOOP FOR JAVA generates unit tests for Java code using feedback-directed random test generation. Below we describe RANDOOP's input, output, and test generation algorithm. We also give an overview of RANDOOP's annotations-based interface for specifying configuration parameters that affect RANDOOP's behavior and output.

Categories and Subject Descriptors: D.2.5 SOFTWARE ENGINEERING: Testing and Debugging

General Terms: Verification

Keywords: Java, random testing.

1. Introduction

Unit testing is an important and widely-practiced activity in software development. Unfortunately, writing unit tests is often a tedious and error-prone task. In this paper, we describe RANDOOP, a tool that automatically generates unit tests for Java code. RANDOOP is available at <http://people.csail.mit.edu/spacheco/randoop/>.

RANDOOP takes as input a set of classes under test, a time limit, and optionally, a set of "contract checkers" that extend those used by RANDOOP as default. RANDOOP outputs two test suites. One contains *contract-violating tests* that exhibit scenarios where the code under test leads to the violation of an API contract. For example, `test1` in Figure 1 is a contract-violating test that reveals an error in Java's collections framework. It shows a way of creating a set that violates reflexivity of equality in Sun's JDK 1.5 and 1.6. In particular, the call `a.equals(a)` returns `false`, which violates the API for `java.lang.Object`. RANDOOP implements a default set of contracts (including reflexivity of equality) that the user can extend.

The second suite that RANDOOP outputs contains *regression tests*, which do not violate contracts but instead capture an aspect of the current implementation. For example, `test2` in Figure 1 shows a regression test that records an aspect of method `BitSet.size()`'s behavior (specifically, cloning a `BitSet` does not change its size). Regression tests can discover inconsistencies between two different versions of the software; `test2` reveals an inconsistency between Sun's JDK 1.5 (on which the test was generated, and thus succeeds by construction) and Sun's JDK 1.6 Beta 2 (on which the test fails). Contract-violating tests and regression tests both have as their purpose finding errors—the former in the current im-

Input:

1. classes under test
2. time limit
3. properties to check

Output:

1. error-revealing test cases
2. regression test cases

Example error-revealing test

```
// Fails on Sun 1.5, 1.6
public static void test1() {
    LinkedList l1 = new LinkedList();
    Object o1 = new Object();
    l1.addFirst(o1);
    l1.addFirst(o1);
    boolean b1 = new Boolean(l1);
    boolean b2 = new Boolean(l1);
    assertEquals(b1, b2);
}

// Fails on Sun 1.5, 1.6
public static void test2() {
    BitSet b1 = new BitSet(1);
    BitSet b2 = new BitSet(1);
    assertEquals(b1, b2);
    b1.set(0);
    assertEquals(b1, b2);
}
```

Figure 1. RANDOOP's input is a set of classes, a time limit, and optionally, a set of contract checkers. It outputs contract-violating tests and regression tests.

Copyright is held by the author(s).
0032-1792/05/0000-0000-0000. All rights reserved.
ACM 978-1-95959-853-7/05/0000.

815

114

21791

416

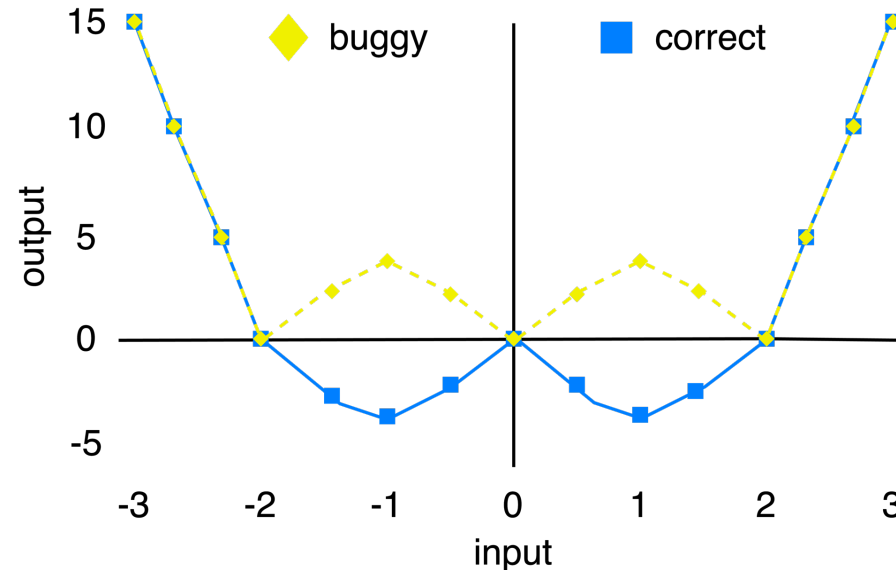
3

Research Gap

```
// Correct implementation  $f(x) = |x| * (x + 2) * (x - 2)$ 
public double example(double x) {
    double output;
    if (x >= 0)
        output = x * (x + 2) * (x - 2);
    else
        output = Math.abs(x) * (x + 2) * (x - 2);
    return output;
}
```

```
// Buggy implementation  $f(x) = |x * (x + 2) * (x - 2)|$ 
public double example_buggy(double x) {
    double output;
    output = Math.abs(x * (x + 2) * (x - 2));
    return output;
}
```

```
@Test
public void test1() {
    double o1 = example_buggy(0.5);
    Assert.assertTrue("msg", o1 >= 0);
}
```



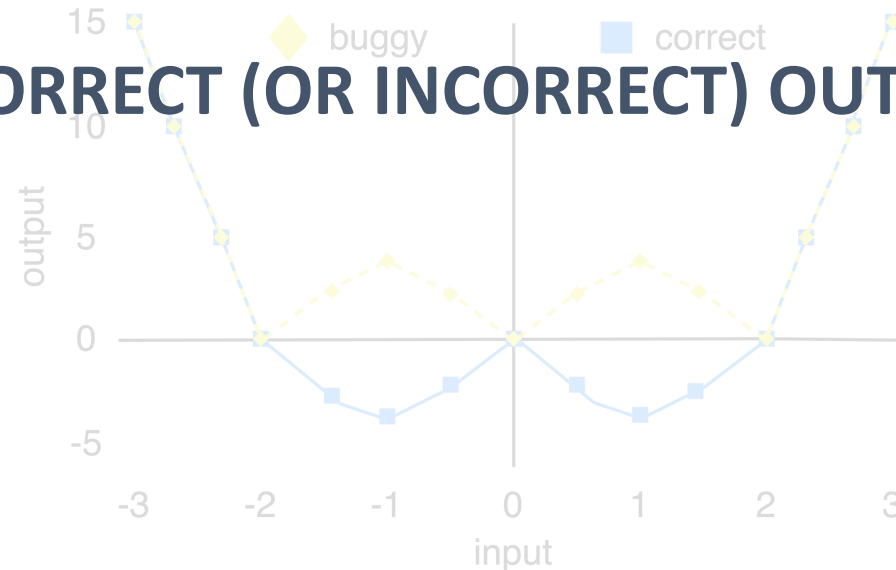
```
@Test
public void test2() {
    double o2 = example_buggy(-1);
    double o3 = example_buggy(1);
    Assert.assertEquals(o2, o3);
}
```

Research Gap

```
// Correct implementation  $f(x) = |x| * (x + 2) * (x - 2)$ 
public double example(double x) {
    double output;
    if (x >= 0)
        output = x * (x + 2) * (x - 2);
    else
        output = Math.abs(x) * (x + 2) * (x - 2);
    return output;
}
```

```
// Buggy implementation  $f(x) = |x * (x + 2) * (x - 2)|$ 
public double example_buggy(double x) {
    double output;
    output = Math.abs(x * (x + 2) * (x - 2));
    return output;
}
```

NEED TO KNOW THE CORRECT (OR INCORRECT) OUTPUT FOR GIVEN INPUTS



```
@Test
public void test1() {
    double o1 = example_buggy(0.5);
    Assert.assertTrue("msg", o1 >= 0);
}
```

```
@Test
public void test2() {
    double o2 = example_buggy(-1);
    double o3 = example_buggy(1);
    Assert.assertEquals(o2, o3);
}
```

Distinguishing Between Correct and Buggy MUTS

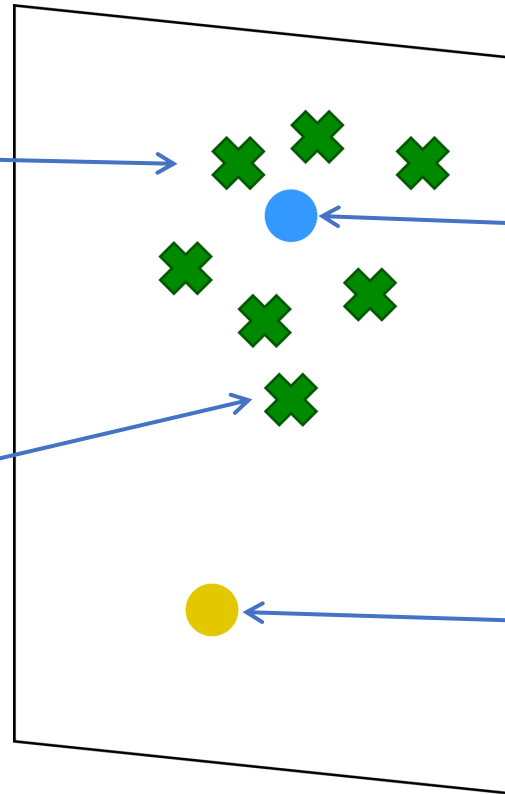
$$f(x) = |x| * (x + 2) * (x - 2)$$

```
@Test Expects o1 = -1.875
public void test1() {
    double o1 = example_buggy(0.5);
    Assert.assertTrue("msg", o1 >= 0);
}
```

Passes on correct code
Fails on buggy code

```
@Test Expects o2 = o3
public void test2() {
    double o2 = example_buggy(-1);
    double o3 = example_buggy(1);
    Assert.assertEquals(o2, o3);
}
```

Passes on correct code
Passes on buggy code



```
// Correct implementation f(x) = |x| * (x + 2) * (x - 2)
public double example(double x) {
    double output;
    if (x >= 0)
        output = x * (x + 2) * (x - 2);
    else
        output = Math.abs(x) * (x + 2) * (x - 2);
    return output;
}
```

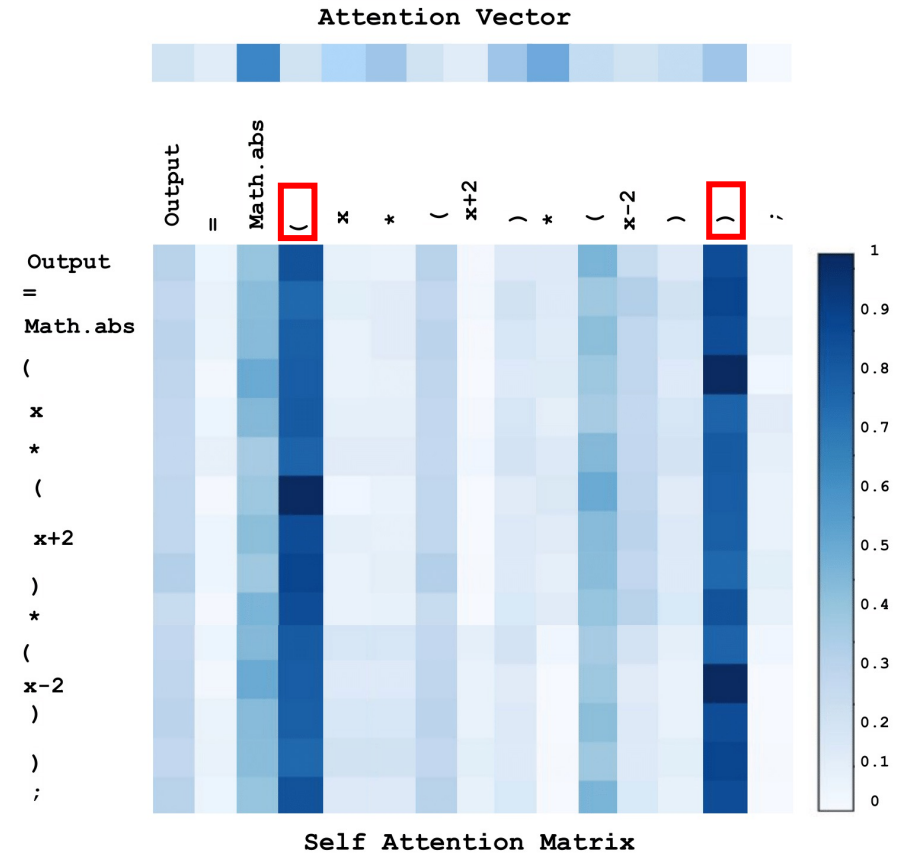
```
// Buggy implementation f(x) = |x * (x + 2) * (x - 2)|
public double example_buggy(double x) {
    double output;
    output = Math.abs(x * (x + 2) * (x - 2));
    return output;
}
```

Model Interpretation

SEER is beyond solving the **ever-challenging** test oracle problem

Model Interpretation (Cont'd)

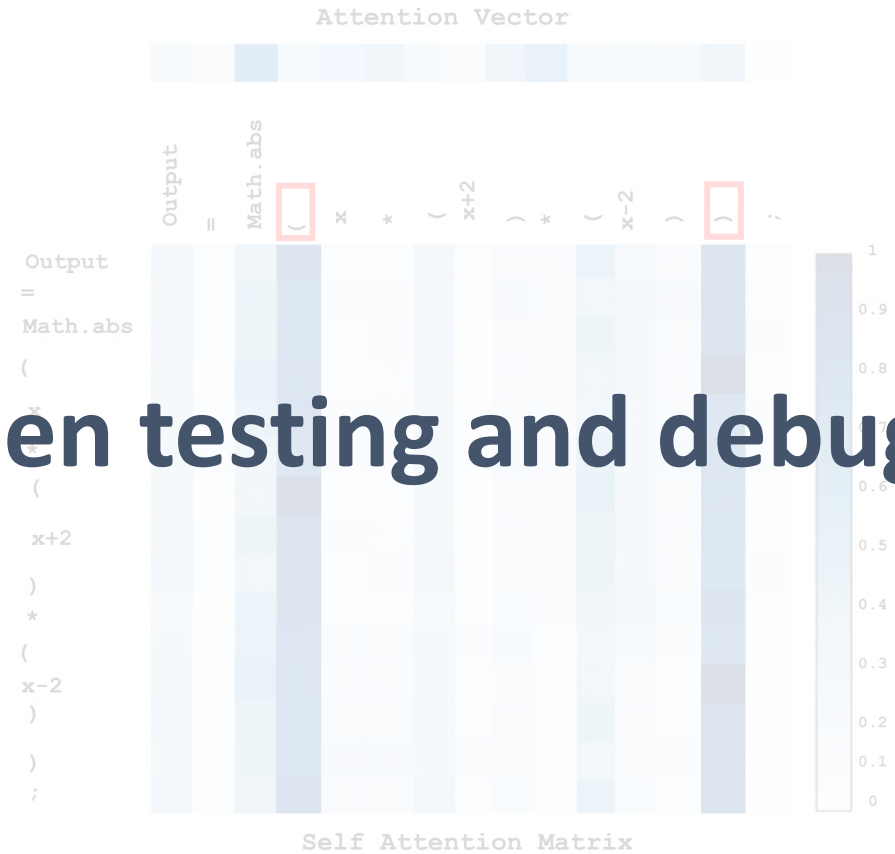
```
// Buggy implementation f(x) = |x * (x + 2) * (x - 2)|  
public double example_buggy(double x) {  
    double output;  
    output = Math.abs(x * (x + 2) * (x - 2));  
    return output;  
}
```



Model Interpretation (Cont'd)

```
// Buggy implementation f(x) = |x * (x + 2) * (x - 2)|  
public double example_buggy(double x) {  
    double output = Math.abs(x * (x + 2) * (x - 2));  
    return output;  
}
```

SEER bridges the gap between testing and debugging



Attention Analysis

Input: MUT's tokens $Tkn = \{c_0, \dots, c_n\}$

Input: MUT's statements $Smt = \{s_0, \dots, s_m\}$

Input: $SA = [[w_{0_0}, \dots, w_{0_n}], \dots, [w_{n_0}, \dots, w_{n_n}]]$

Input: Attention threshold k

Output: Attended tokens $ATkn$, Attended statements $ASmt$

1 $ATkn \leftarrow \emptyset$

2 $ASmt \leftarrow \emptyset$

3 **foreach** $row = [w_{i_0}, \dots, w_{i_n}] \in SA$ **do**

 // $localATkn = \{\langle c_i, ind_i \rangle \mid ind_i \text{ is index of } c_i \text{ in } Tkn\}$

4 $localATkn \leftarrow getMostAttended(row, Tkn, k)$

5 **foreach** $\langle c_i, ind_i \rangle \in localATkn$ **do**

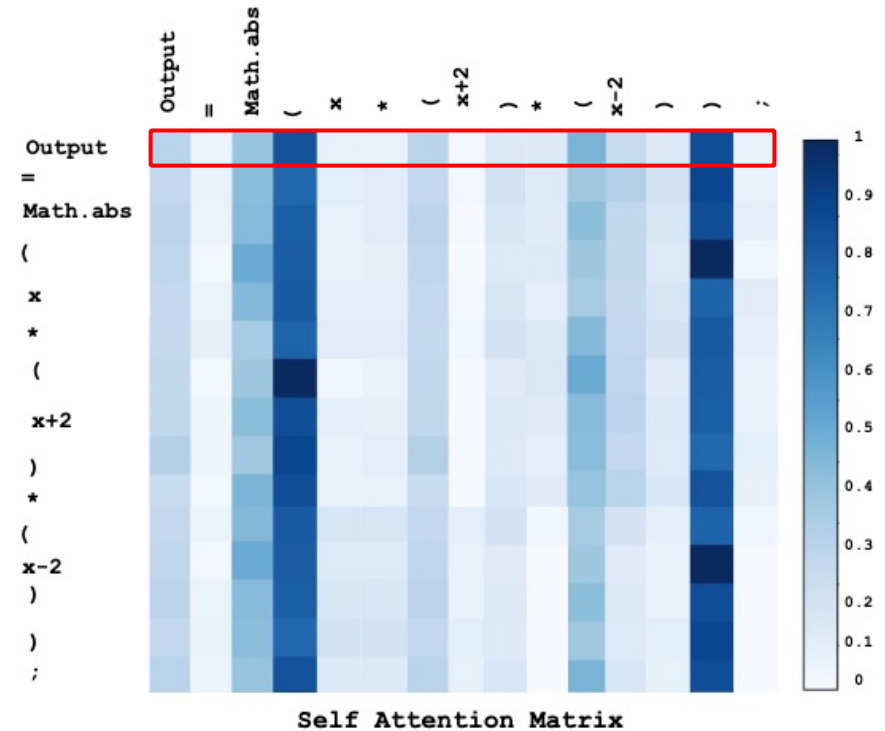
6 **if** $\neg ATkn.contains(\langle c_i, ind_i \rangle)$ **then**

7 $ATkn \leftarrow ATkn \cup c_i$

8 **foreach** $s_i \in Smt$ **do**

9 **if** $|s_i \cap ATkn| > k$ **then**

10 $ASmt \leftarrow ASmt \cup s_i$



```
// Buggy implementation f(x) = |x * (x + 2) * (x - 2)|
public double example_buggy(double x) {
    double output;
    output = Math.abs(x * (x + 2) * (x - 2));
    return output;
}
```

Research Questions

- ❖ Effectiveness in predicting test labels
 - ❖ Generalization to unseen projects
- ❖ Interpretation analysis
 - ❖ Performance

Experimental Setup

SEER's dataset is built on top of Defects4J + Augmentations

Projects	# Bugs	# Mutants	Dataset								
			#Tests #Pass, #Fail (Contribution%)	Defects4J				Higher-Order Mutants			
				Developer Tests		Randoop Tests		Developer Tests		Randoop Tests	
# Pass	# Fail	# Pass	# Fail	# Pass	# Fail	# Pass	# Fail	# Pass	# Fail		
Compress	47	24,322	30,753 (30.26%)	1,214	22	2,965	2,230	385	265	192	23,480
Lang	64	12,824	15,130 (14.89%)	702	59	1,249	296	397	509	1,124	10,794
Chart	26	4,313	14,901 (14.66%)	248	37	5,316	4,987	154	109	229	3,821
Math	106	9,488	13,580 (13.36%)	1,685	91	1,584	732	732	814	546	7,396
Codec	18	8,973	10,210 (10.05%)	271	11	648	307	375	54	93	8,451
Closure	174	1,615	3,170 (3.12%)	1,431	8	113	3	768	76	49	722
JacksonDatabind	112	581	2,818 (2.77%)	2,133	27	77	0	410	94	42	35
Time	26	203	2,415 (2.38%)	1,765	39	255	153	84	70	18	31
Jsoup	93	677	2,134 (2.1%)	925	57	458	17	125	234	277	41
Cli	39	913	1,612 (1.59%)	348	10	292	49	106	75	371	361
Csv	16	1,166	1,582 (1.56%)	327	6	83	0	297	2	23	844
JacksonCore	26	941	1,420 (1.4%)	331	15	105	28	320	163	136	322
Gson	18	531	1,386 (1.36%)	678	15	93	69	295	92	144	0
JXPath	22	143	314 (0.31%)	93	1	74	3	0	1	105	37
Mockito	38	0	100 (0.1%)	56	39	5	0	0	0	0	0
JacksonXml	6	49	81 (0.08%)	19	1	9	3	28	12	2	7
Collections	4	0	7 (0.01%)	4	1	2	0	0	0	0	0
Total	835	66,739	101,613 (100%)	12,230	439	13,328	8,877	4,476	2,570	3,351	56,342

Can SEER Effectively Predict Test Labels?

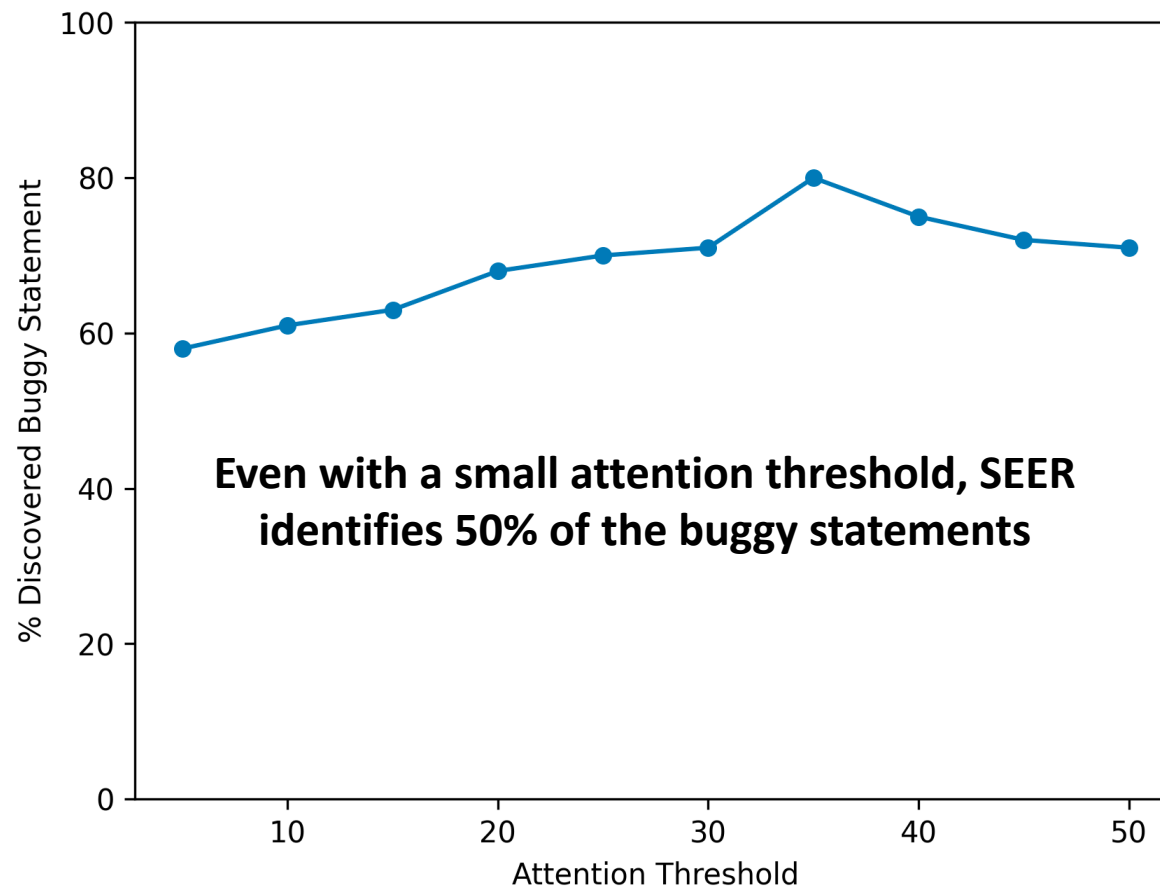
SEER's Evaluation

Accuracy	Precision	Recall	F1 Score
93%	86%	94%	90%

The longer the MUTs and test, the better the performance of SEER

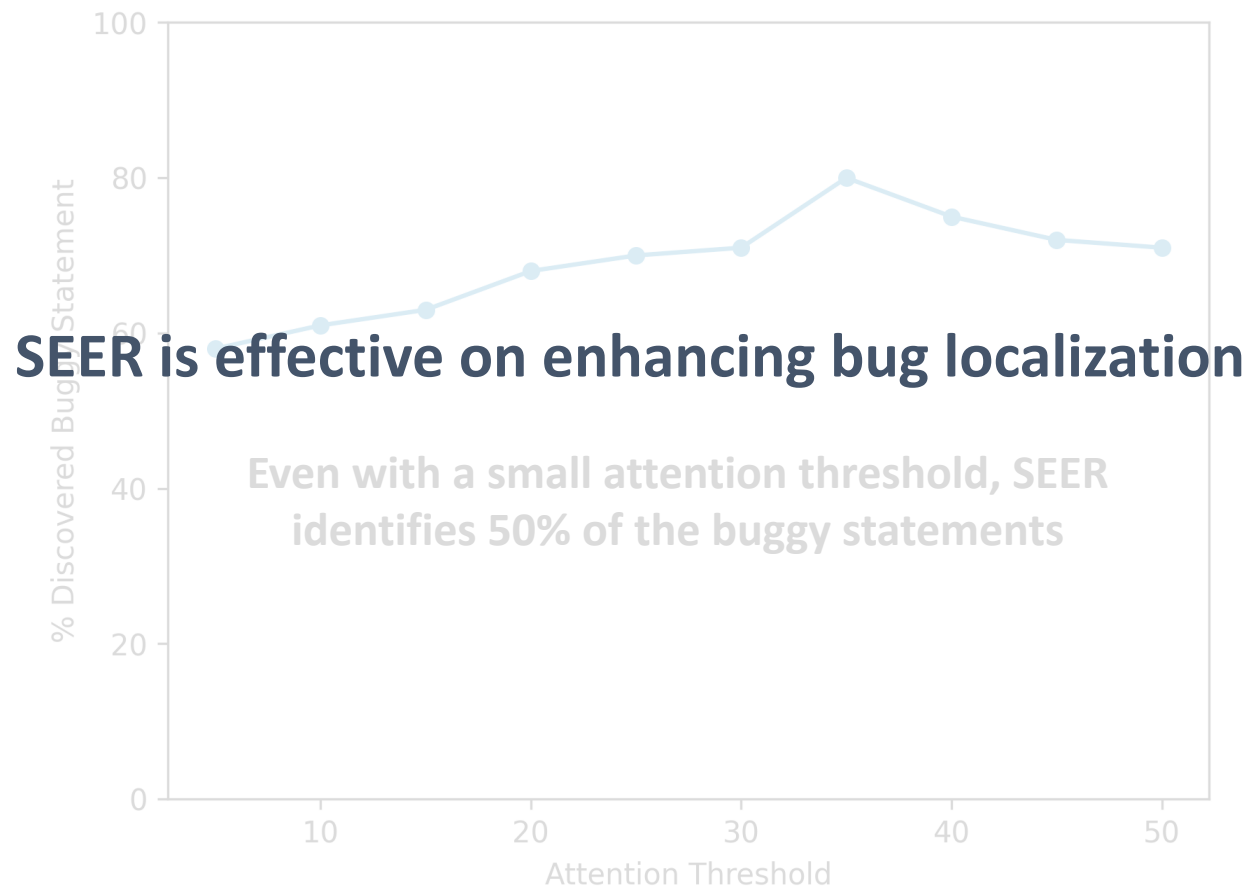
Can SEER find buggy statements?

$$\% \text{ Discovered Buggy Statements} = \frac{\text{Attended Buggy Statements}}{\text{Buggy Statements}} \times 100$$



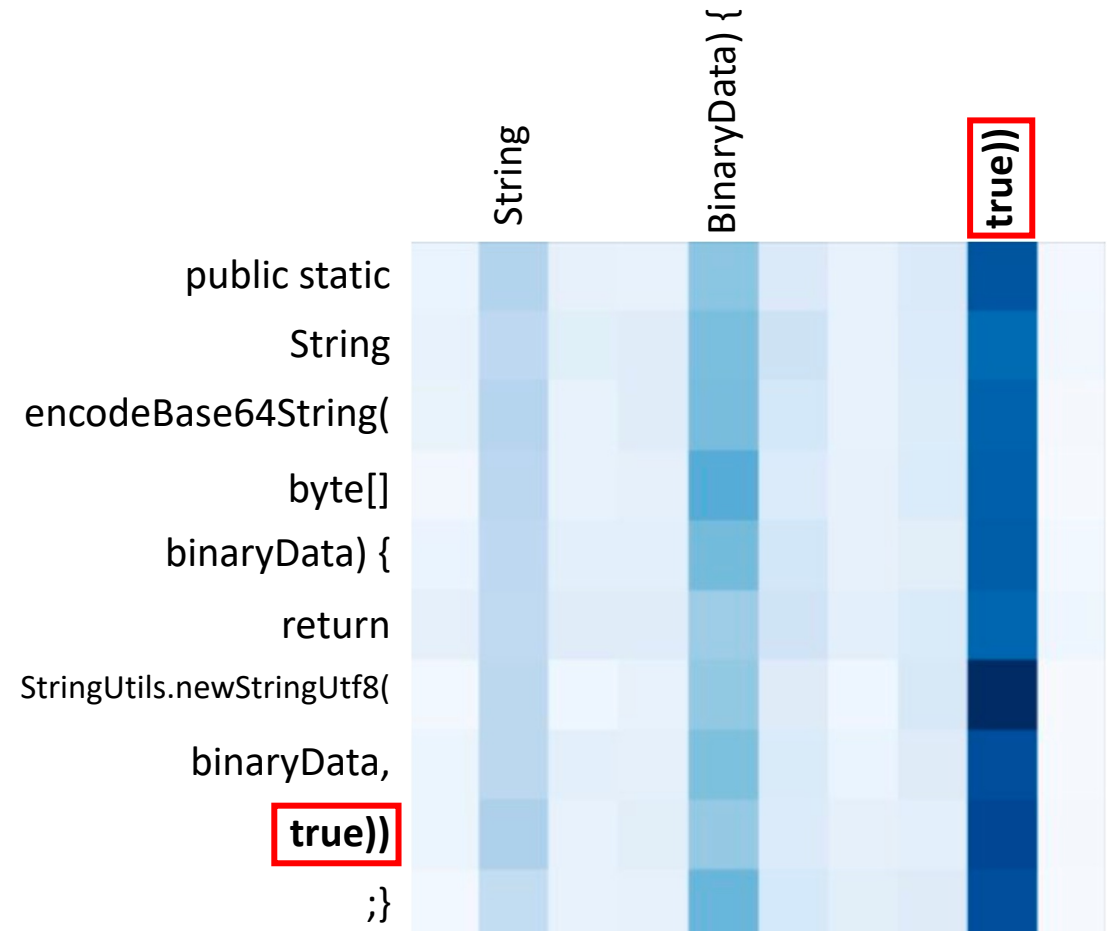
Can SEER find buggy statements?

$$\% \text{ Discovered Buggy Statements} = \frac{\text{Attended Buggy Statements}}{\text{Buggy Statements}} \times 100$$



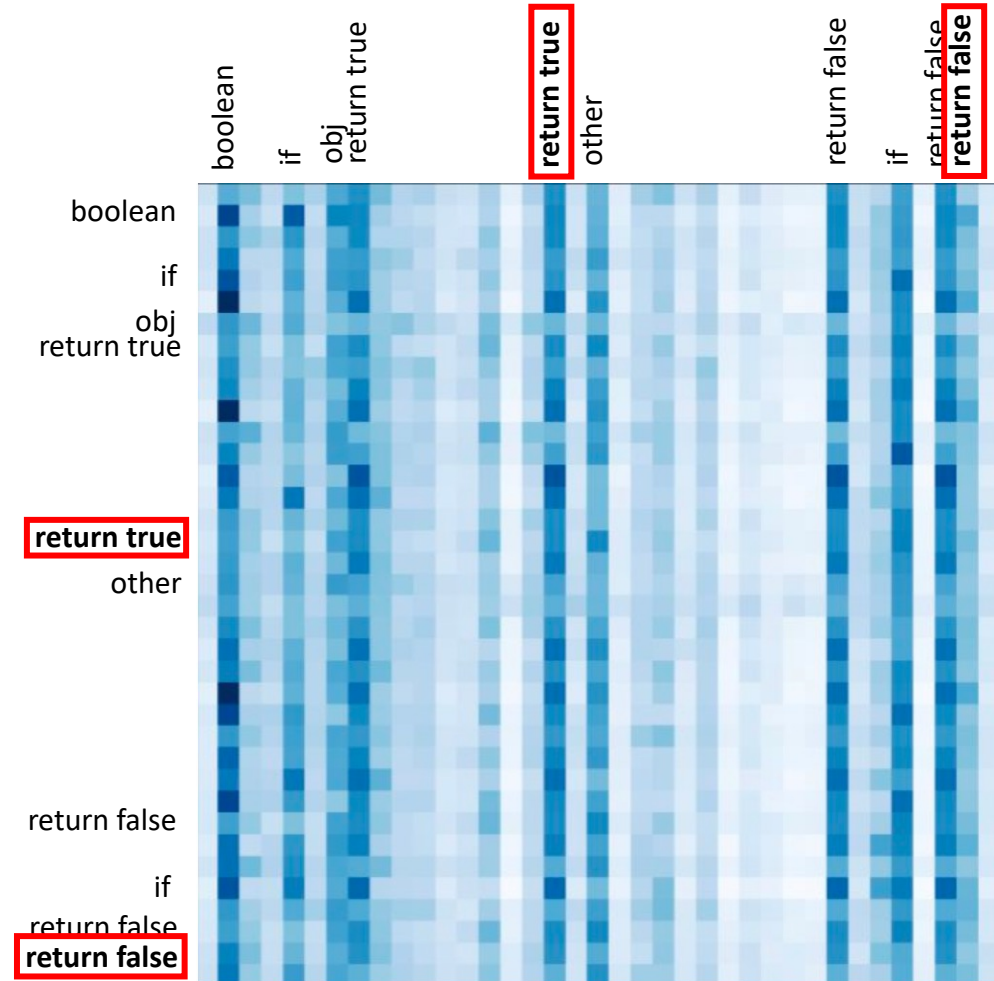
Can SEER find buggy tokens?

```
public static String encodeBase64String(  
    byte[] binaryData) {return StringUtils.newStringUtf8(  
        encodeBase64(binaryData, true));  
}
```



Can SEER find buggy tokens? (Cont'd)

```
public boolean equals(Object obj) {  
    if (this == obj)  
        return true;  
    if (obj == null || getClass() != obj.getClass())  
        return true;  
    ZipArchiveEntry other = (ZipArchiveEntry) obj;  
    if (name == null) {  
        if (other.name != null)  
            return false;  
    } else if (!name.equals(other.name))  
        return false;  
    return false;  
}
```



Conclusions

- ❖ We propose SEER for automated test oracle construction
- ❖ SEER learns the correlation between inputs and outputs
- ❖ SEER model is interpretable and can enhance bug localization
- ❖ SEER is highly accurate and generalizable to unseen projects



<https://github.com/Intelligent-CAT-Lab/SEER>

