Causal Testing:
Understanding defects root causes

INSPIRED Lab

(INterdisciplinary Software Practice Improvement REsearch and Development)

Dr. Brittany Johnson-Matthews

Assistant Professor in Computer Science

George Mason University

CAHMP Core Faculty

https://brittjay.me

https://cahmp.gmu.edu/

Today's Talk

How did we get here (and by we, I mean me)?

Improving what you already do with what already exists

Where else can this apply?

Is it really useful?

What's the back story on the research in today's presentation?

It all started with a study...

Study of why developer do and don't use tools

Findings suggest the major issues include:

tool output
result understandability
tool design
workflow integration

So I went on a mission to provide <u>useful</u>, <u>usable</u>, and <u>validated</u> interventions for improving software practice.

Fast forward to post-PhD...

Opportunity to work in the testing space – cool!

We already know **testing is powerful** (and commonly used).

But as I made progress, I realized:

1. There are **a lot** of techniques available for use.
2. Traditional testing alone does not answer the question *"why is this happening?"*

Can we take what developers already do,
and the work that's already been done,
to provide insights that existing tools don't?

Causal Testing

Automated **causal experiments**

Start with **existing test cases**

Uses **existing debugging techniques**

Goal = **minimally different executions**

```
Failing: New York, NY, USA to
         900 René Lévesque Blvd. W Montreal, QC, Canada
Failing: Boston, MA, USA to
         900 René Lévesque Blvd. W Montreal, QC, Canada
Failing: New York, NY, USA to
         1 Harbour Square, Toronto, ON, Canada
Passing: New York, NY, USA to
         39 Dalton St, Boston, MA, USA
Passing: Toronto, ON, Canada to
         900 René Lévesque Blvd. W Montreal, QC, Canada
Passing: Vancouver, BC, Canada to
         900 René Lévesque Blvd. W Montreal, QC, Canada
```

```
Minimally different execution traces:

Failing:                             Passing:
[...]                                [...]
findSubEndPoints(sor6, tar7);        findSubEndPoints(sor6, tar7);
findSubEndPoints(sor7, tar8);        findSubEndPoints(sor7, tar8);
metricConvert(pathSoFar);
findSubEndPoints(sor8, tar9);        findSubEndPoints(sor8, tar9);
[...]                                [...]
```

Causal Testing

Bug report

Directions from
`New York, NY, USA`
to
`900 René Lévesque Blvd. W. Montreal, QC, Canada`
are wrong.

1 **Failed test:**

```
✗ assertTrue(pathFromTo("New York, NY, USA", "900 René Lévesque Blvd. W. Montreal, QC, Canada"));
```

2 **Perturb input(s) & execute tests:**

	Pass	Fail
`pathFromTo("New York, NY, USA", "W. Montreal, QC, Canada")`		✗
`pathFromTo("New York, NY, USA", "René Lévesque Blvd. W. Montreal, QC, Canada")`		✗
`pathFromTo("New York, NY, USA", "1 Harbor Point Blvd. Boston, MA, USA")`	✓	
`pathFromTo("Boston, MA, USA", "900 René Lévesque Blvd. W. Montreal, QC, Canada")`		✗
`pathFromTo("Vancouver, BC, Canada", "900 René Lévesque Blvd. W. Montreal, QC, Canada")`	✓	

```
...
```

3 **Find most similar** **input + execution**

Similar passing:

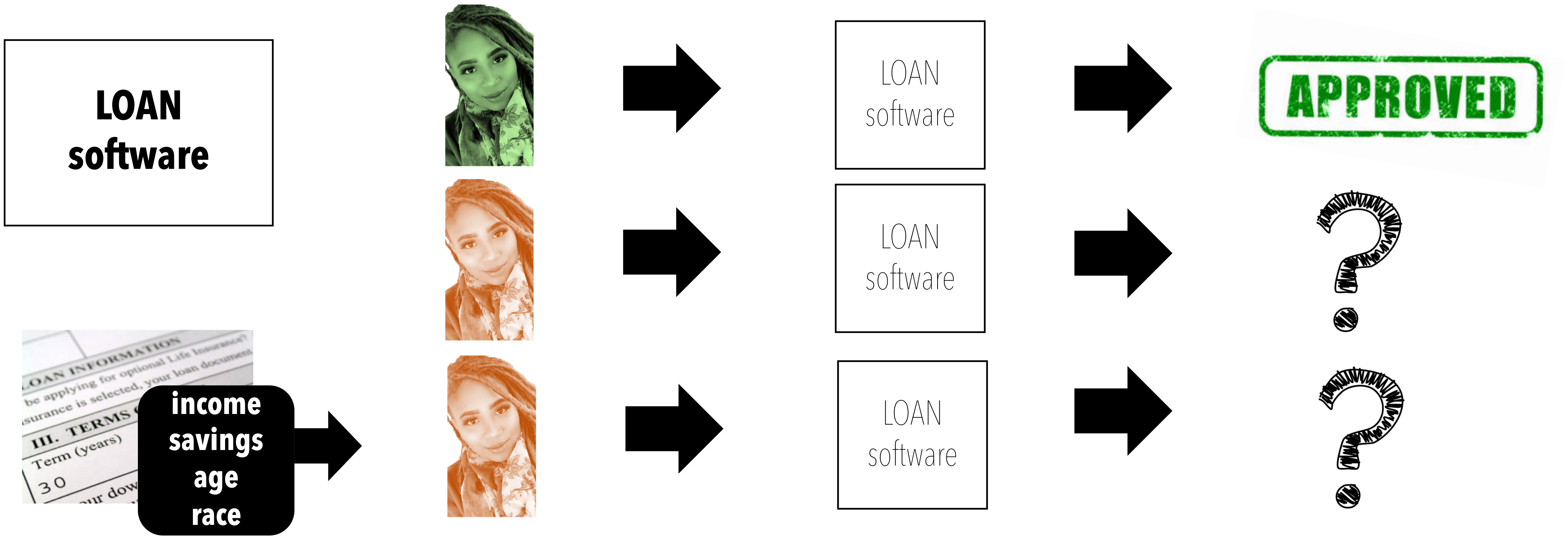
```
pathFromTo("New York, NY, USA", "1 Harbor Point Blvd. Boston, MA, USA")
pathFromTo("Vancouver, BC, Canada", "900 René Lévesque Blvd. W. Montreal, QC, Canada")
```

Similar failing:

```
pathFromTo("New York, NY, USA", "René Lévesque Blvd. W. Montreal, QC, Canada")
pathFromTo("Boston, MA, USA", "900 René Lévesque Blvd. W. Montreal, QC, Canada")
```

What else can we do with causal testing?

Causal Fairness Testing

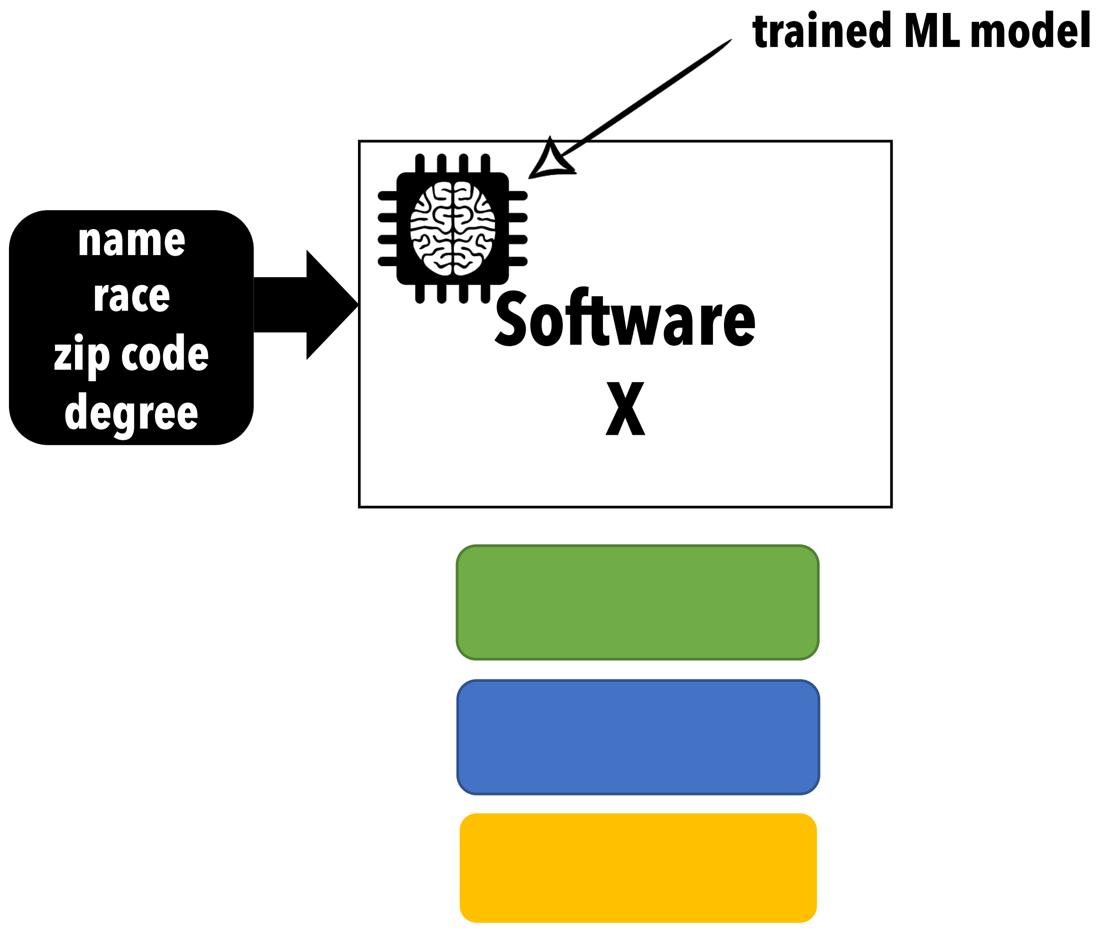


…

How often is the outcome different because of race alone?

http://fairness.cs.umass.edu

ML-Based Causal Testing



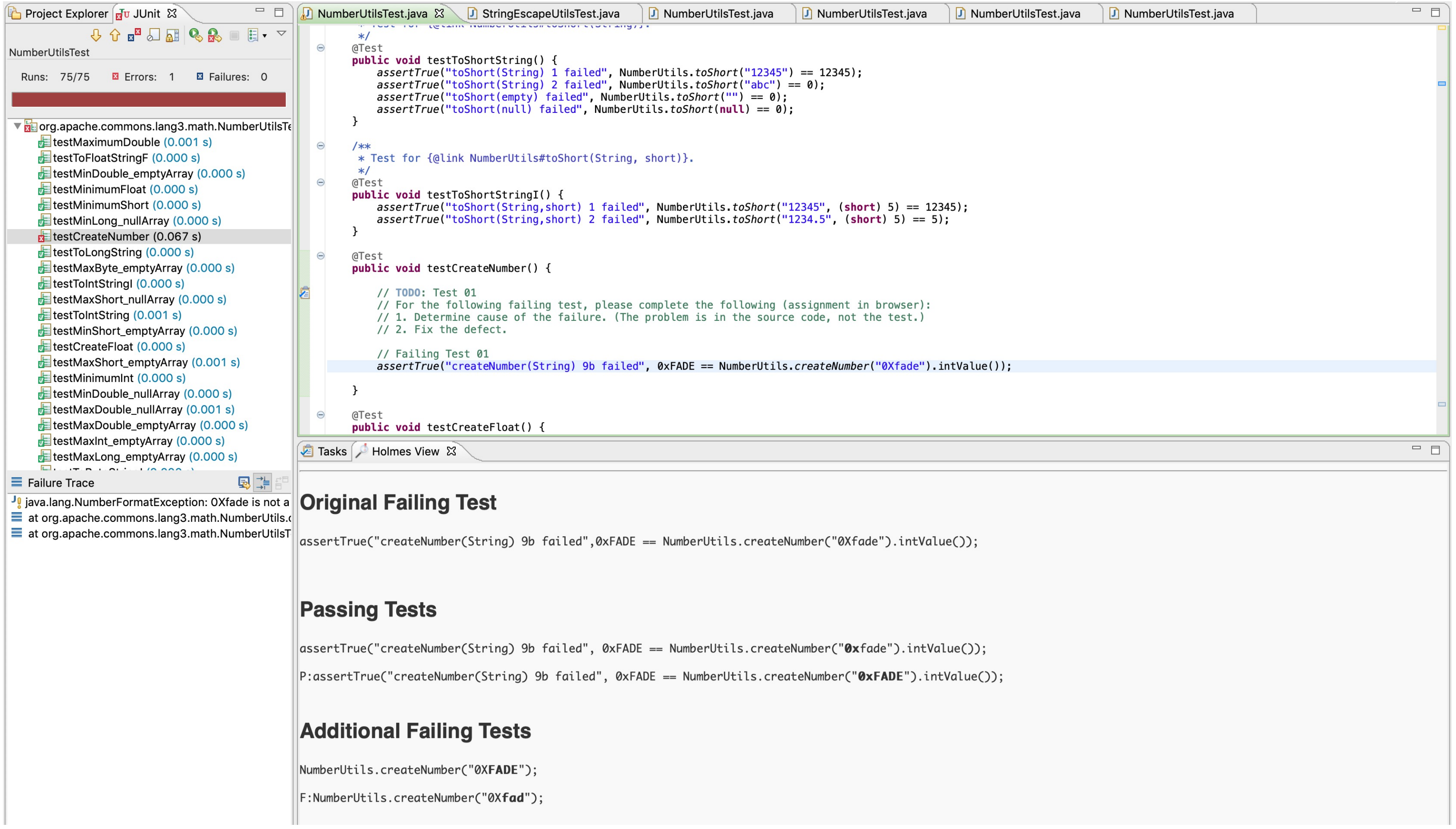
`assertEqual (` April red 12345 PhD , John green 01020 PhD `)`

`assertTrue (` April red 12345 PhD `,` 🟩 `)`

Is it useful ?

HOLMES

proof-of-concept Causal Testing implementation



Project Explorer | JUnit

NumberUtilsTest

Runs: 75/75 Errors: 1 Failures: 0

```
org.apache.commons.lang3.math.NumberUtilsTe
  testMaximumDouble (0.001 s)
  testToFloatStringF (0.000 s)
  testMinDouble_emptyArray (0.000 s)
  testMinimumFloat (0.000 s)
  testMinimumShort (0.000 s)
  testMinLong_nullArray (0.000 s)
  testCreateNumber (0.067 s)
  testToLongString (0.000 s)
  testMaxByte_emptyArray (0.000 s)
  testToIntStringI (0.000 s)
  testMaxShort_nullArray (0.000 s)
  testToIntString (0.001 s)
  testMinShort_emptyArray (0.000 s)
  testCreateFloat (0.000 s)
  testMaxShort_emptyArray (0.001 s)
  testMinimumInt (0.000 s)
  testMinDouble_nullArray (0.000 s)
  testMaxDouble_nullArray (0.001 s)
  testMaxDouble_emptyArray (0.000 s)
  testMaxInt_emptyArray (0.000 s)
  testMaxLong_emptyArray (0.000 s)
```

Failure Trace

```
java.lang.NumberFormatException: 0Xfade is not a
at org.apache.commons.lang3.math.NumberUtils.
at org.apache.commons.lang3.math.NumberUtilsT
```

NumberUtilsTest.java | StringEscapeUtilsTest.java | NumberUtilsTest.java | NumberUtilsTest.java | NumberUtilsTest.java | NumberUtilsTest.java

```java
     */
    @Test
    public void testToShortString() {
        assertTrue("toShort(String) 1 failed", NumberUtils.toShort("12345") == 12345);
        assertTrue("toShort(String) 2 failed", NumberUtils.toShort("abc") == 0);
        assertTrue("toShort(empty) failed", NumberUtils.toShort("") == 0);
        assertTrue("toShort(null) failed", NumberUtils.toShort(null) == 0);
    }

    /**
     * Test for {@link NumberUtils#toShort(String, short)}.
     */
    @Test
    public void testToShortStringI() {
        assertTrue("toShort(String,short) 1 failed", NumberUtils.toShort("12345", (short) 5) == 12345);
        assertTrue("toShort(String,short) 2 failed", NumberUtils.toShort("1234.5", (short) 5) == 5);
    }

    @Test
    public void testCreateNumber() {

        // TODO: Test 01
        // For the following failing test, please complete the following (assignment in browser):
        // 1. Determine cause of the failure. (The problem is in the source code, not the test.)
        // 2. Fix the defect.

        // Failing Test 01
        assertTrue("createNumber(String) 9b failed", 0xFADE == NumberUtils.createNumber("0Xfade").intValue());

    }

    @Test
    public void testCreateFloat() {
```

Tasks | Holmes View

Original Failing Test

```
assertTrue("createNumber(String) 9b failed",0xFADE == NumberUtils.createNumber("0Xfade").intValue());
```

Passing Tests

```
assertTrue("createNumber(String) 9b failed", 0xFADE == NumberUtils.createNumber("0xfade").intValue());

P:assertTrue("createNumber(String) 9b failed", 0xFADE == NumberUtils.createNumber("0xFADE").intValue());
```

Additional Failing Tests

```
NumberUtils.createNumber("0XFADE");

F:NumberUtils.createNumber("0Xfad");
```

http://holmes.cs.umass.edu/

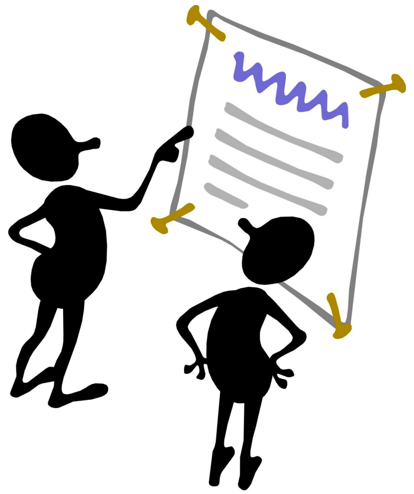
Can causal testing help developers debug?

Does Causal Testing improve ability to identify defect root cause?

YES!

Does Causal Testing improve ability to repair defects?

YES!



Do developers find Causal Testing useful? What's most useful?

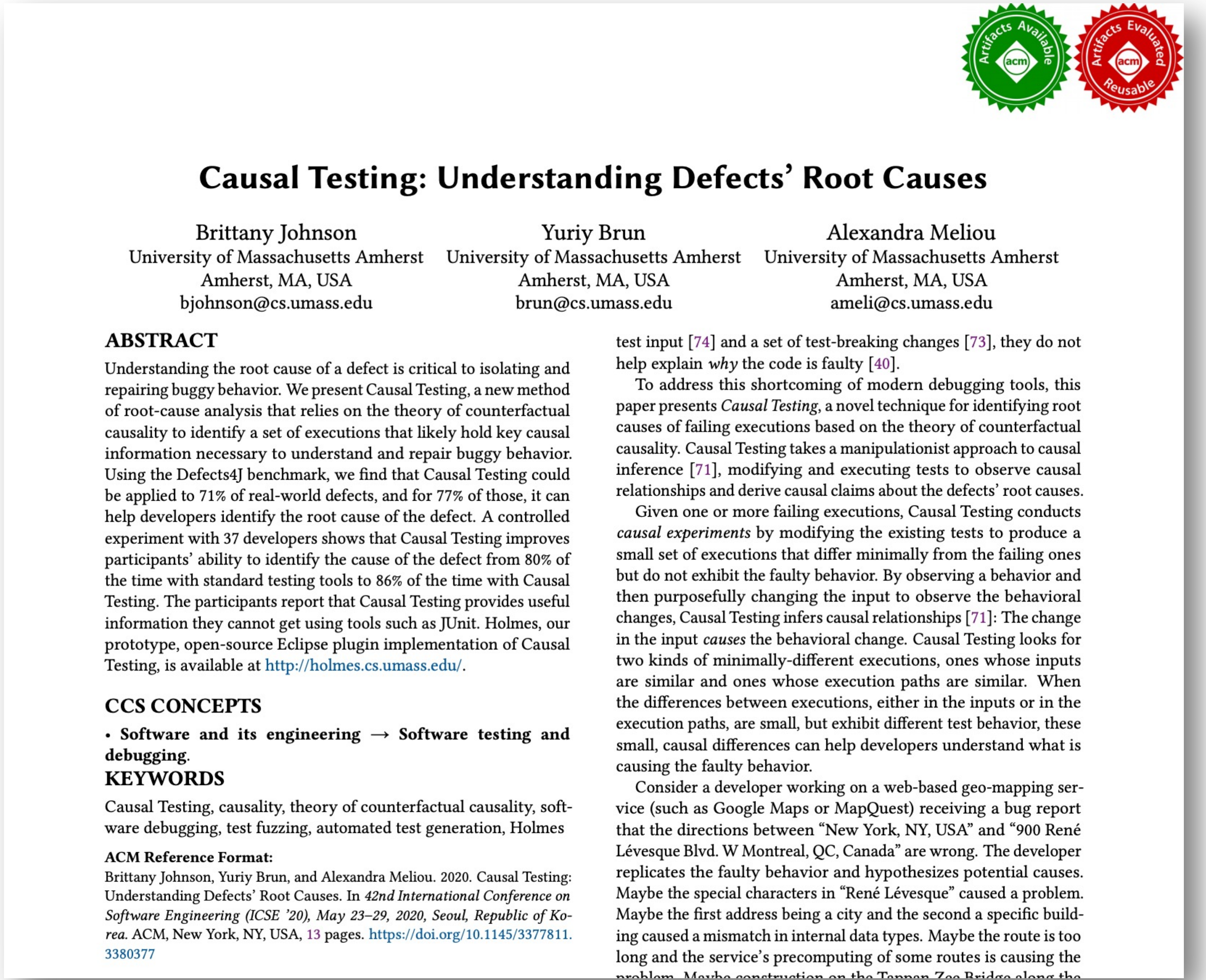
YES!

similar passing tests → cause identification

Causal Testing is a **useful** technique that provides **more insight** into faulty executions with **code you've already written**.

Thanks!

Brittany Johnson
johnsonb@gmu.edu



Causal Testing: Understanding Defects' Root Causes

Brittany Johnson
University of Massachusetts Amherst
Amherst, MA, USA
bjohnson@cs.umass.edu

Yuriy Brun
University of Massachusetts Amherst
Amherst, MA, USA
brun@cs.umass.edu

Alexandra Meliou
University of Massachusetts Amherst
Amherst, MA, USA
ameli@cs.umass.edu

ABSTRACT

Understanding the root cause of a defect is critical to isolating and repairing buggy behavior. We present Causal Testing, a new method of root-cause analysis that relies on the theory of counterfactual causality to identify a set of executions that likely hold key causal information necessary to understand and repair buggy behavior. Using the Defects4J benchmark, we find that Causal Testing could be applied to 71% of real-world defects, and for 77% of those, it can help developers identify the root cause of the defect. A controlled experiment with 37 developers shows that Causal Testing improves participants' ability to identify the cause of the defect from 80% of the time with standard testing tools to 86% of the time with Causal Testing. The participants report that Causal Testing provides useful information they cannot get using tools such as JUnit. Holmes, our prototype, open-source Eclipse plugin implementation of Causal Testing, is available at http://holmes.cs.umass.edu/.

CCS CONCEPTS

• **Software and its engineering → Software testing and debugging.**

KEYWORDS

Causal Testing, causality, theory of counterfactual causality, software debugging, test fuzzing, automated test generation, Holmes

ACM Reference Format:
Brittany Johnson, Yuriy Brun, and Alexandra Meliou. 2020. Causal Testing: Understanding Defects' Root Causes. In *42nd International Conference on Software Engineering (ICSE '20), May 23–29, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3377811.3380377

test input [74] and a set of test-breaking changes [73], they do not help explain *why* the code is faulty [40].

To address this shortcoming of modern debugging tools, this paper presents *Causal Testing*, a novel technique for identifying root causes of failing executions based on the theory of counterfactual causality. Causal Testing takes a manipulationist approach to causal inference [71], modifying and executing tests to observe causal relationships and derive causal claims about the defects' root causes.

Given one or more failing executions, Causal Testing conducts *causal experiments* by modifying the existing tests to produce a small set of executions that differ minimally from the failing ones but do not exhibit the faulty behavior. By observing a behavior and then purposefully changing the input to observe the behavioral changes, Causal Testing infers causal relationships [71]: The change in the input *causes* the behavioral change. Causal Testing looks for two kinds of minimally-different executions, ones whose inputs are similar and ones whose execution paths are similar. When the differences between executions, either in the inputs or in the execution paths, are small, but exhibit different test behavior, these small, causal differences can help developers understand what is causing the faulty behavior.

Consider a developer working on a web-based geo-mapping service (such as Google Maps or MapQuest) receiving a bug report that the directions between "New York, NY, USA" and "900 René Lévesque Blvd. W Montreal, QC, Canada" are wrong. The developer replicates the faulty behavior and hypothesizes potential causes. Maybe the special characters in "René Lévesque" caused a problem. Maybe the first address being a city and the second a specific building caused a mismatch in internal data types. Maybe the route is too long and the service's precomputing of some routes is causing the problem. Maybe construction on the Tappan Zee Bridge along the