ReAssert: Suggesting Repairs for Broken Unit Tests

Brett Daniel
Danny Dig
Vilas Jagannath
Darko Marinov

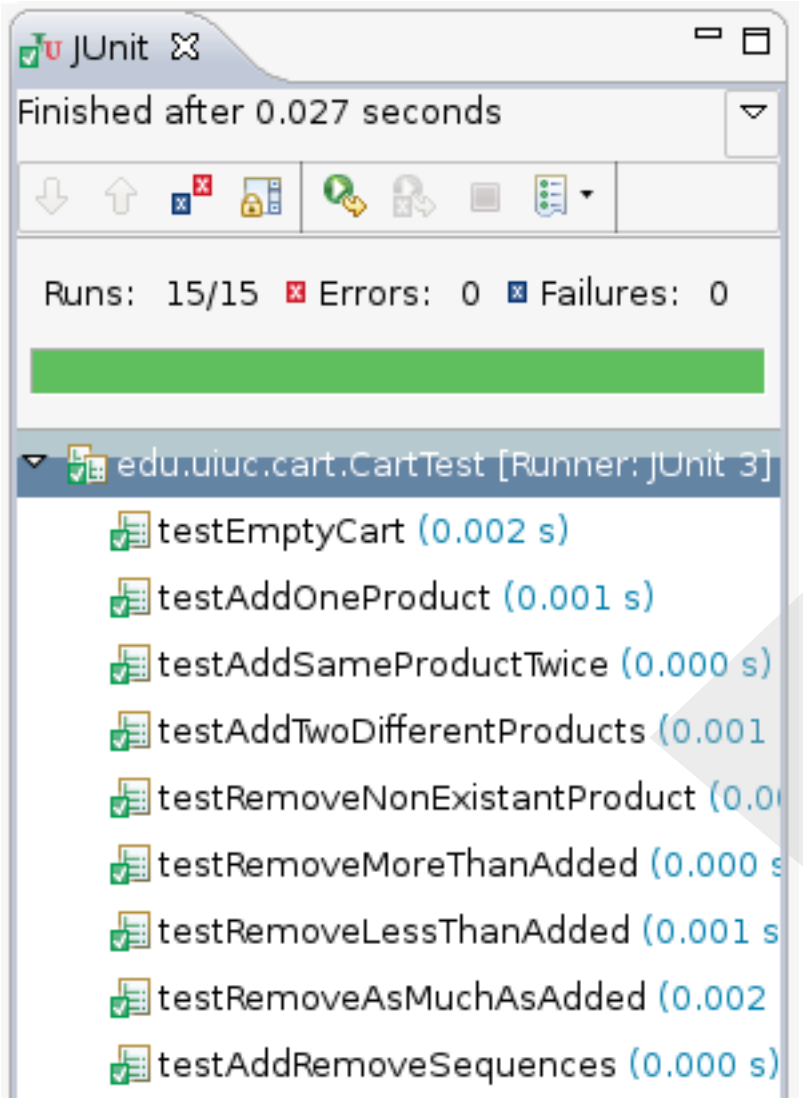
ILLINOIS
UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Tihomir Gvero

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

CS527 Topics in Software Engineering
August 26, 2010

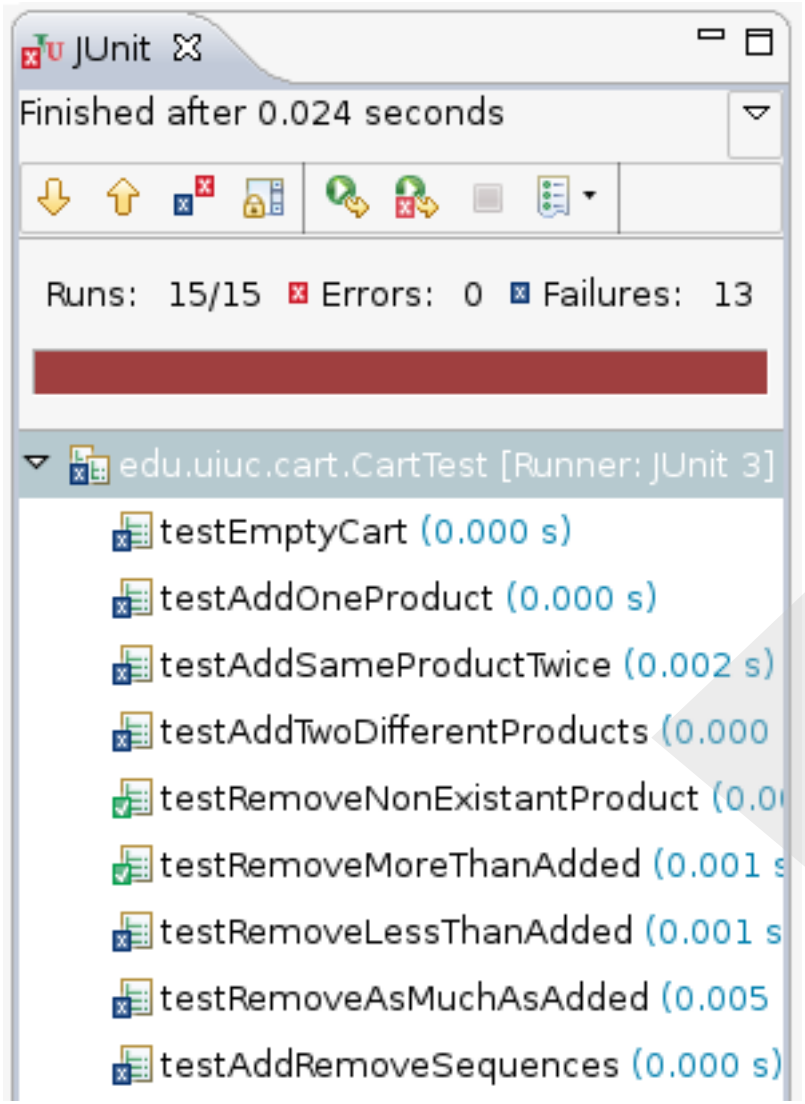
Passing Unit Tests




```
public class Cart {
    ...
    public double getTotalPrice() {...}
    public String getPrintedBill() {...}
    ...
}
```

```
public void testAddTwoDifferentProducts() {
    Cart cart = ...
    assertEquals(3.0, cart.getTotalPrice());
    assertEquals(
        "Discount: -$3.00, Total: $3.00",
        cart.getPrintedBill());
}
```

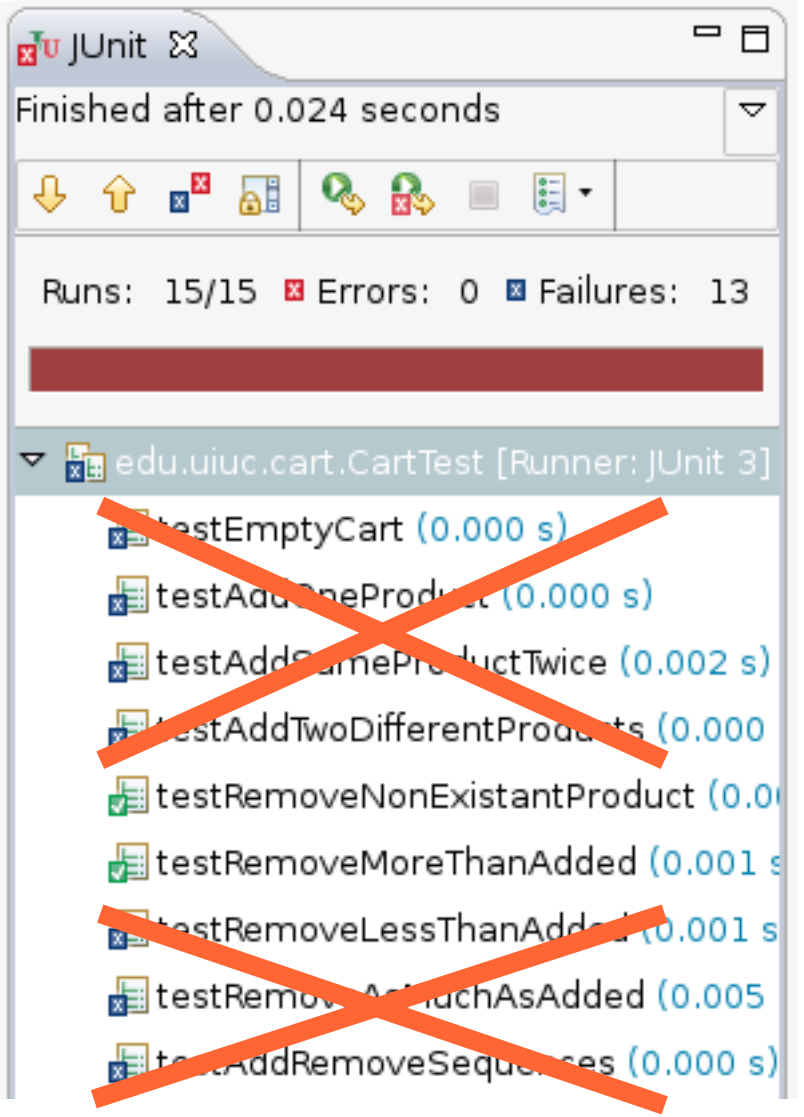
Requirements Change




```
public class Cart {
    ...
    public double getTotalPrice() {...}
    public String getPrintedBill() {...}
    ...
}
```

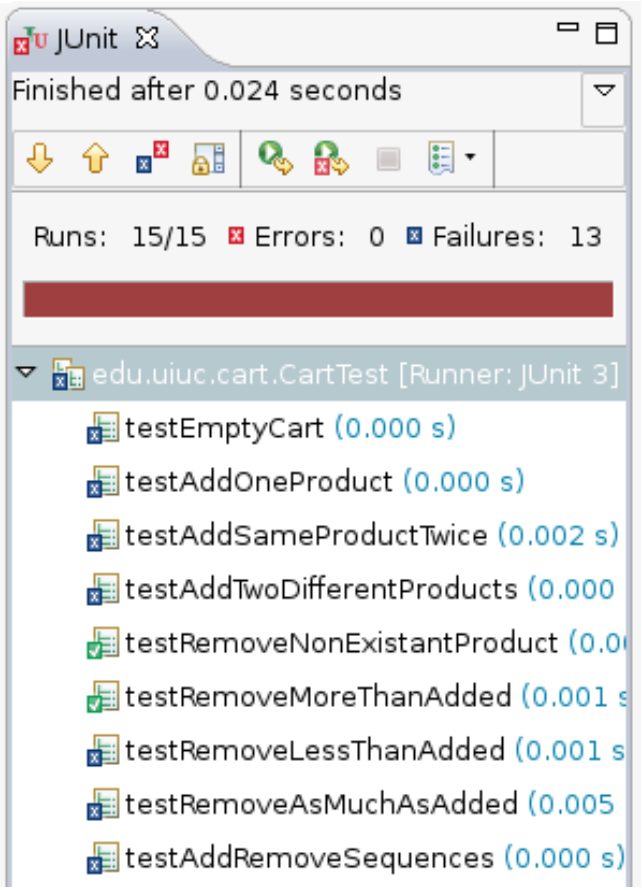
```
public void testAddTwoDifferentProducts() {
    Cart cart = ...
    assertEquals(3.0, cart.getTotalPrice());
    assertEquals(
        "Discount: -$3.00, Total: $3.00",
        cart.getPrintedBill());
}
```

Delete Broken Tests?

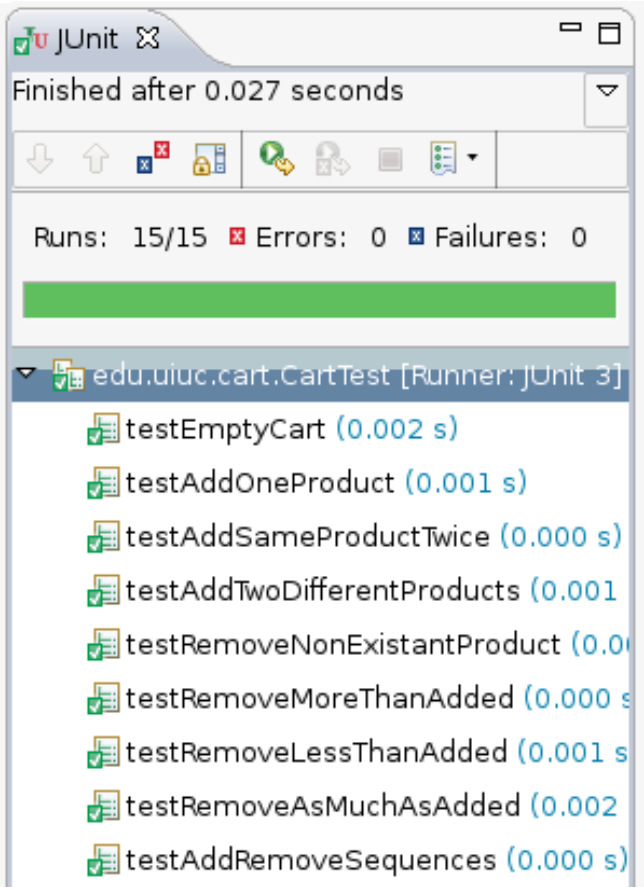


But that reduces the quality of the test suite

Repairing Tests is Preferable

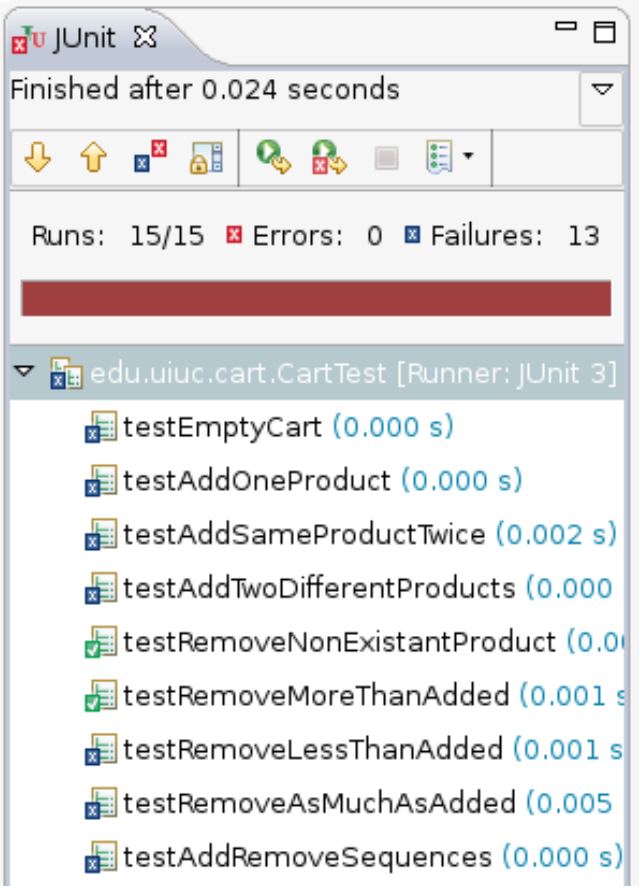


But that requires
a lot of time
and effort

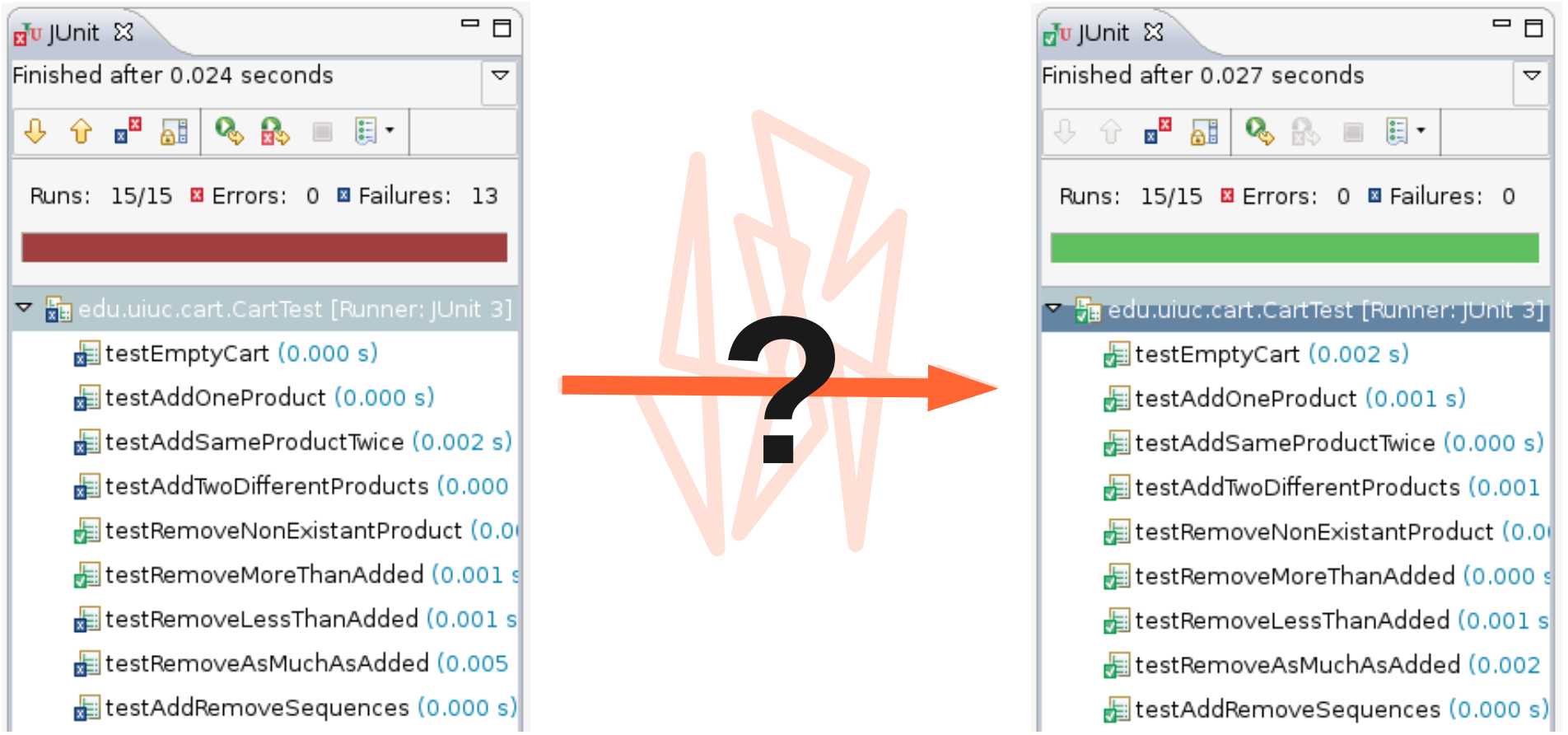


Common Problem

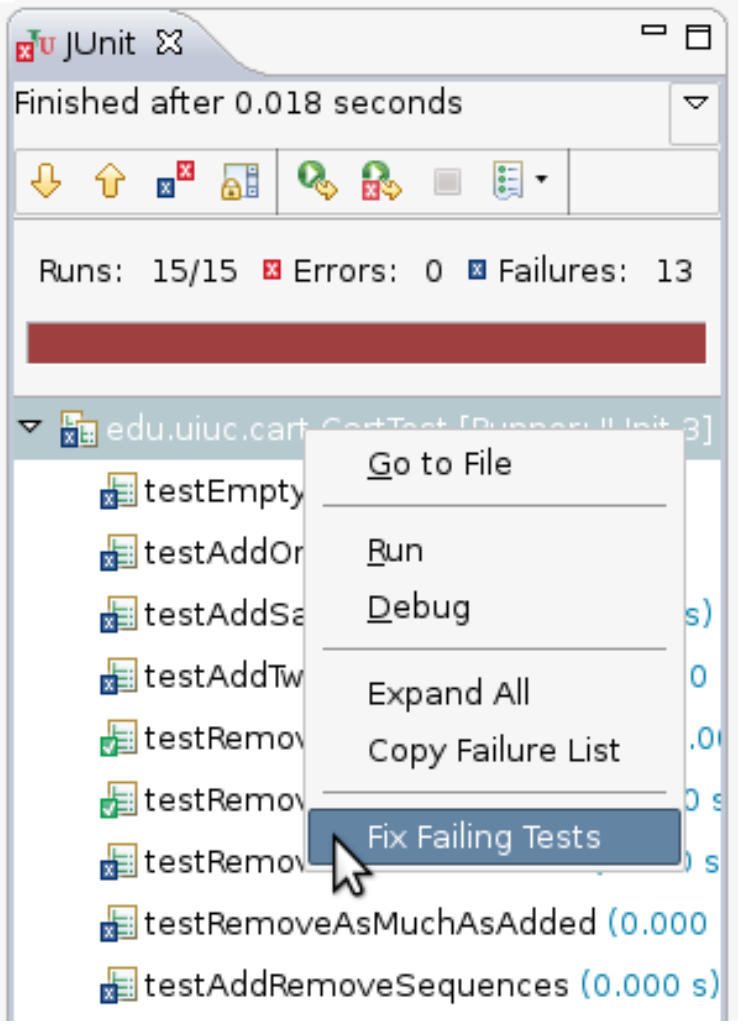
- Changing requirements
- Changing dependencies
- Multiple developers
- Tool-generated tests



Is It A Research Problem?



ReAssert Suggests Repairs

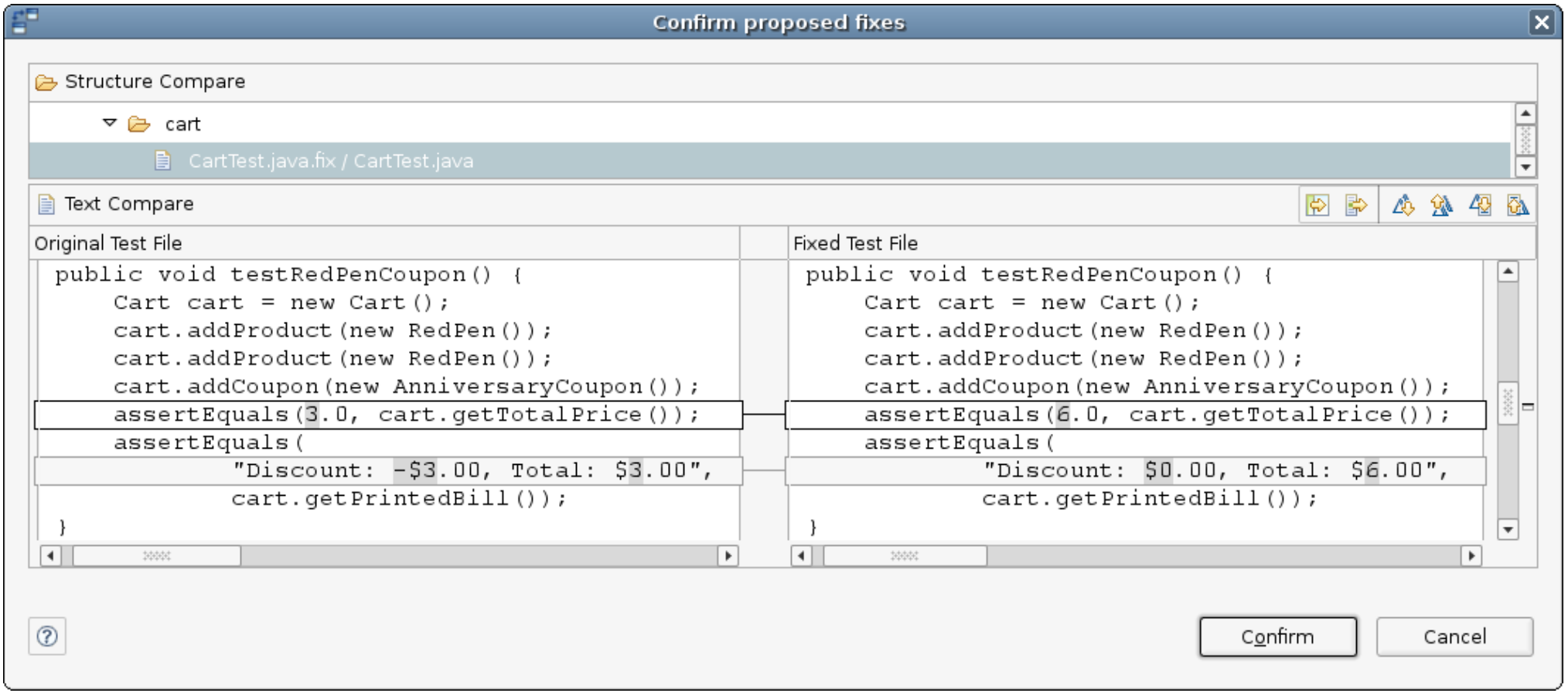


ReAssert: Suggesting Repairs for Broken Unit Tests

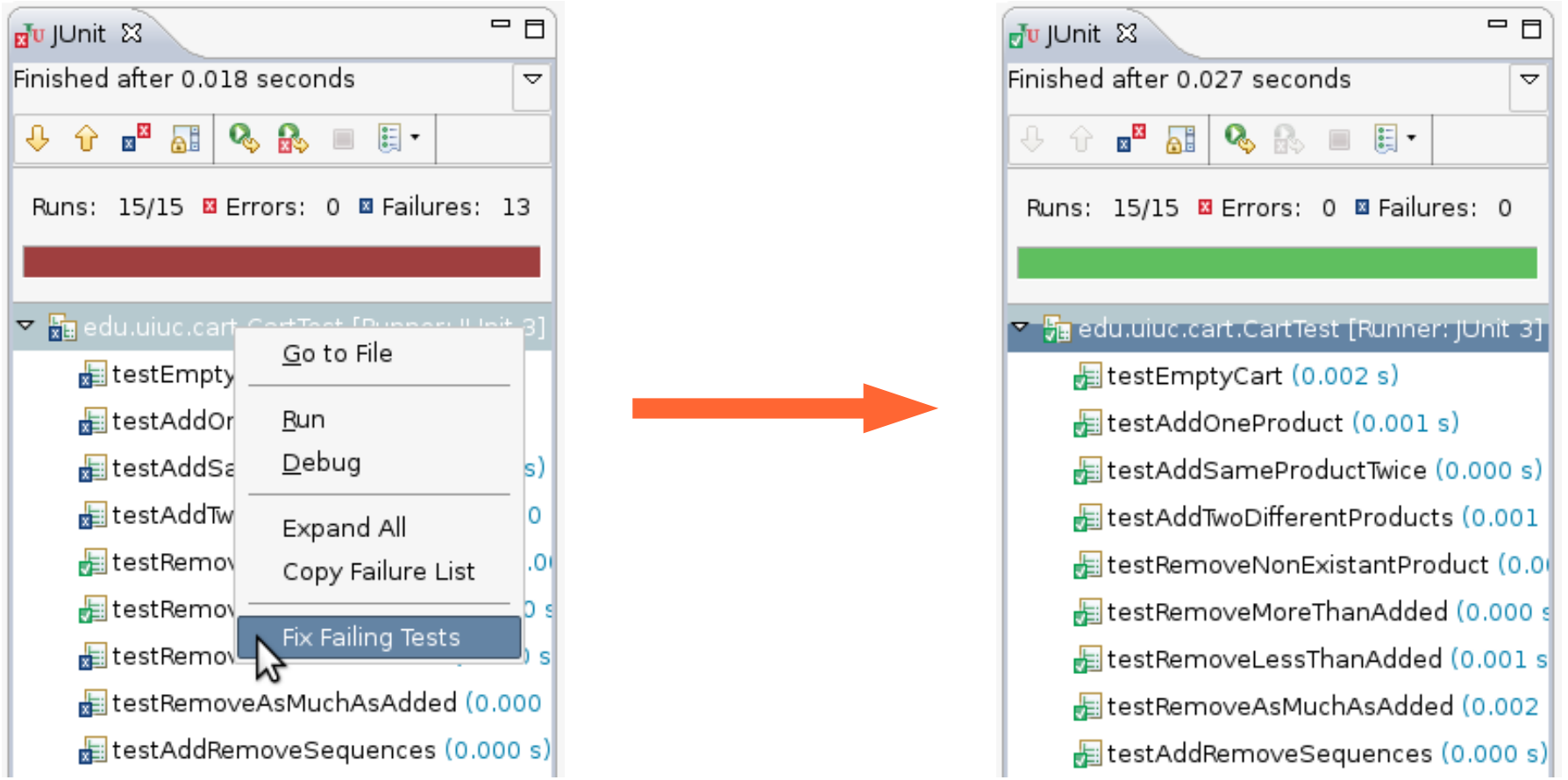
Brett Daniel, Vilas Jagannath, Danny Dig, Darko Marinov

ASE 2009. Auckland, New Zealand

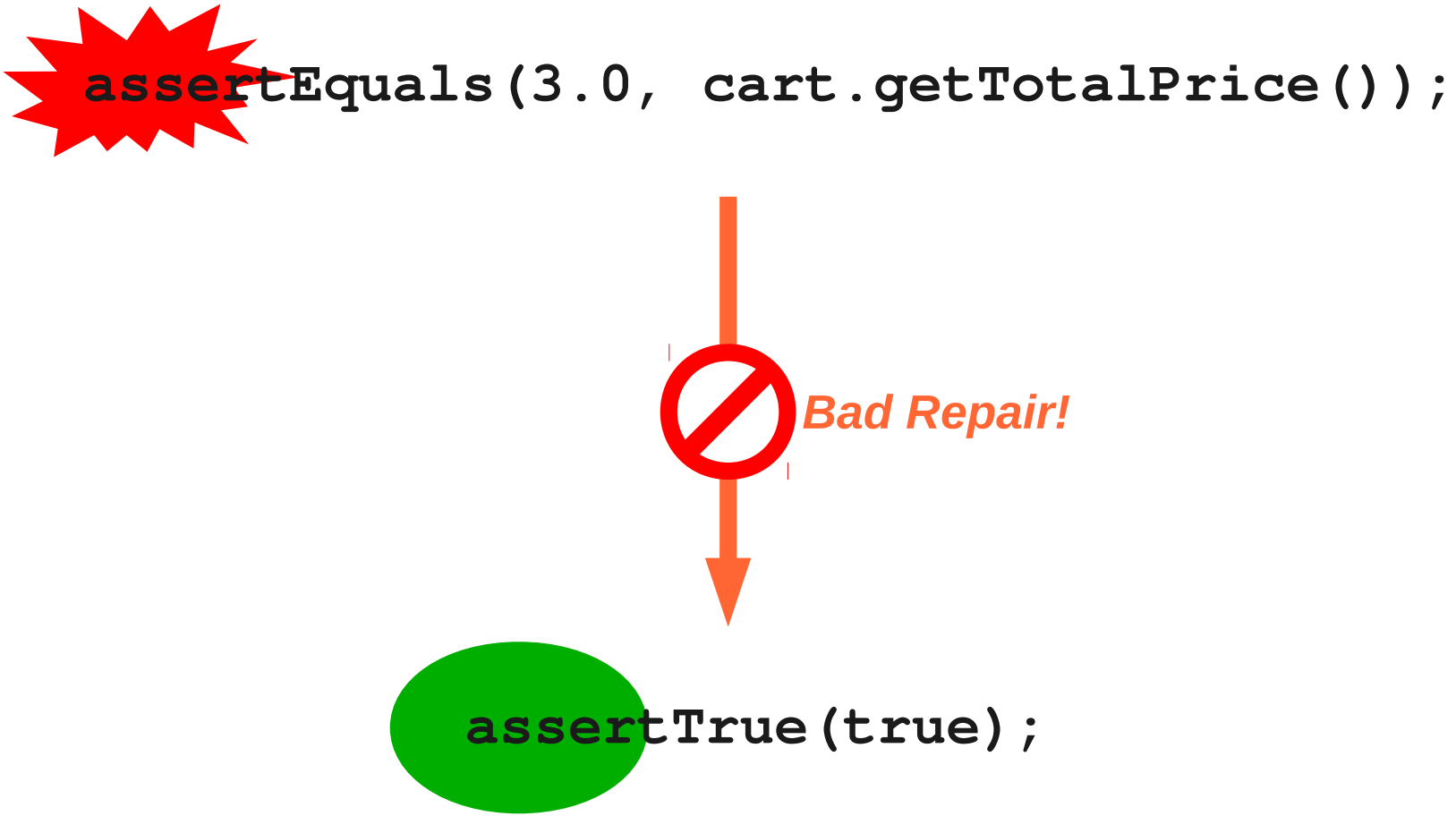
Confirm or Reject Suggestions



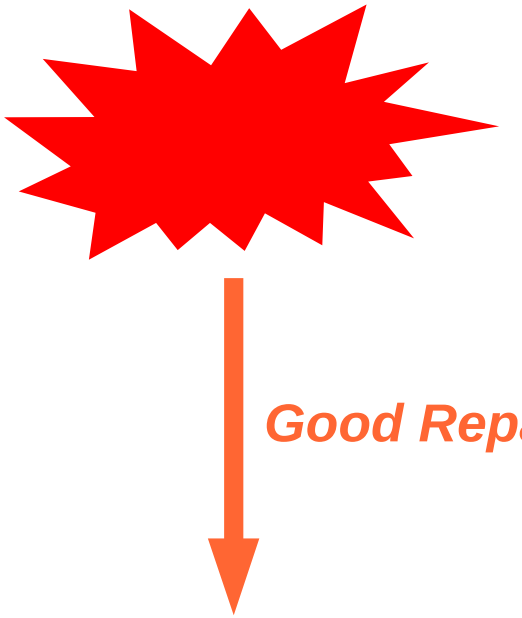
ReAssert Reduces Effort



What is a Good Repair?



Repair Criteria



Make tests **pass**

Make **minimal changes** to test code

Leave SUT **unchanged**

Require **developer approval**

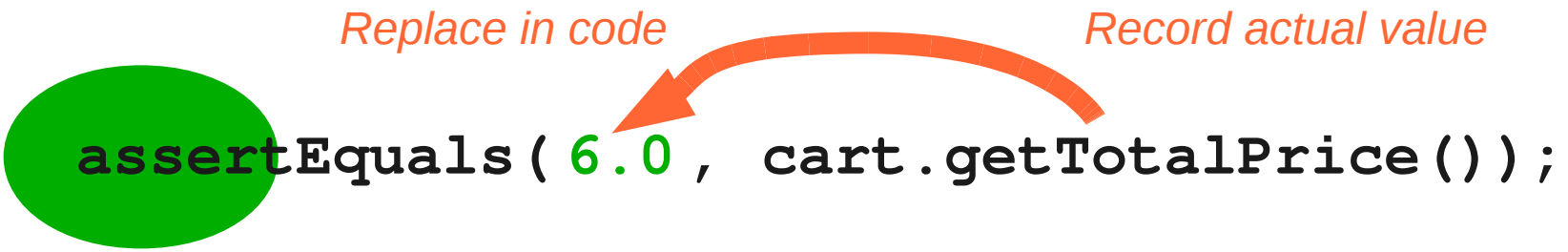
Repair Strategies

- Strategies specific to:
  - Static **structure** of the code
  - The **type** of failure
  - The **runtime values** that caused the failure
- Seven general strategies + custom strategies

Simple Assertion Failure

```
assertEquals(3.0, cart.getTotalPrice());
```

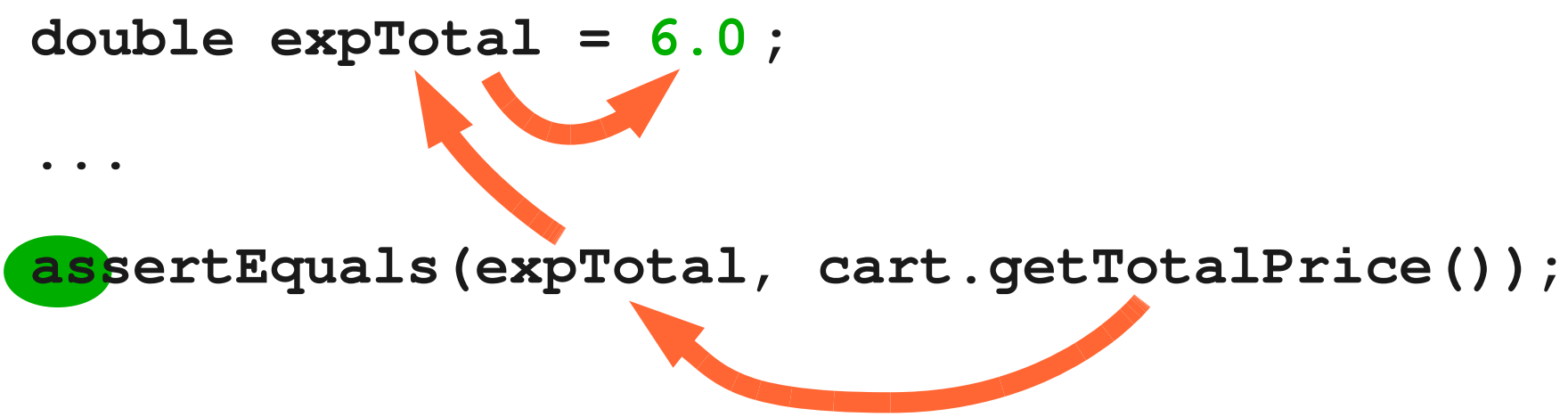
Replace Literal



Temporary Variable

```
double expTotal = 3.0;

...

assertEquals(expTotal, cart.getTotalPrice());
```

Trace Declaration-Use Path

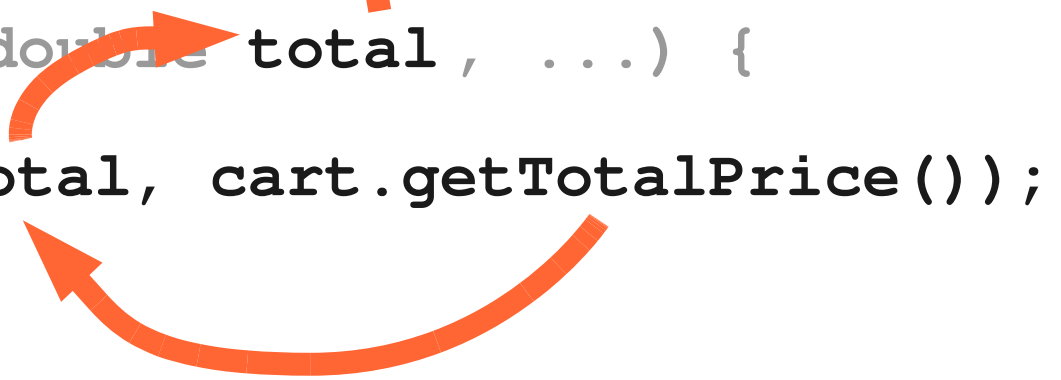


Failure in Helper Method

```
void testAddTwoDifferentProducts() {
  Cart cart = ...
  ...
  checkCart(cart, 3.0, ...);
}

void checkCart(
      Cart cart, double total, ...) {
  ...
  assertEquals(total, cart.getTotalPrice());
  ...
}
```

Trace Declaration-Use Path

```
void testAddTwoDifferentProducts() {
  Cart cart = ...
  ...
  checkCart(cart, 6.0, ...);
}

void checkCart(
      Cart cart, double total, ...) {
  ...
  assertEquals(total, cart.getTotalPrice());
  ...
}
```



Object (In)Equality Failure

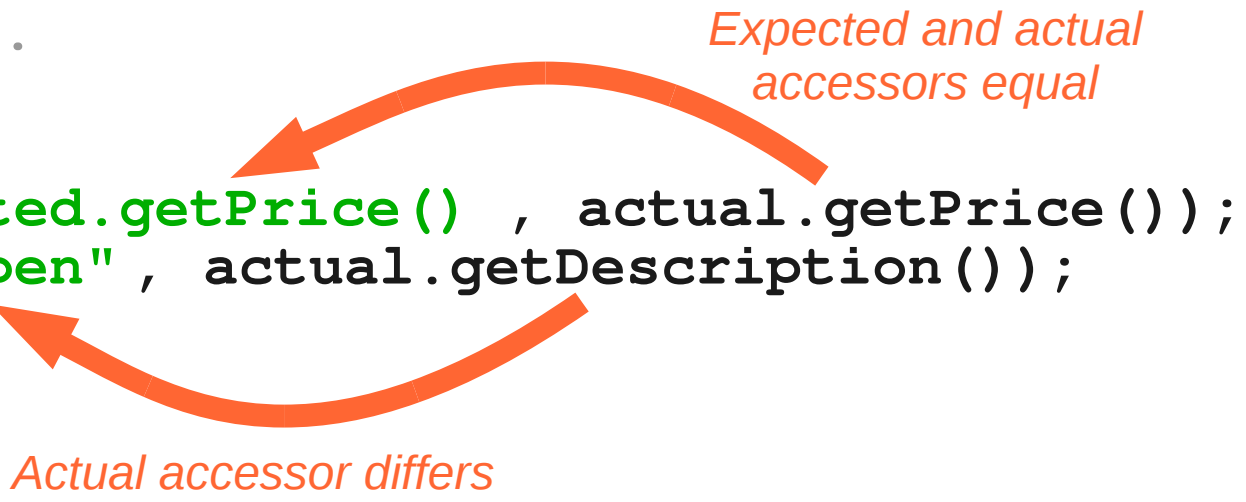
```
Product expected = ...
Product actual = ...
assertEquals(expected, actual);
```

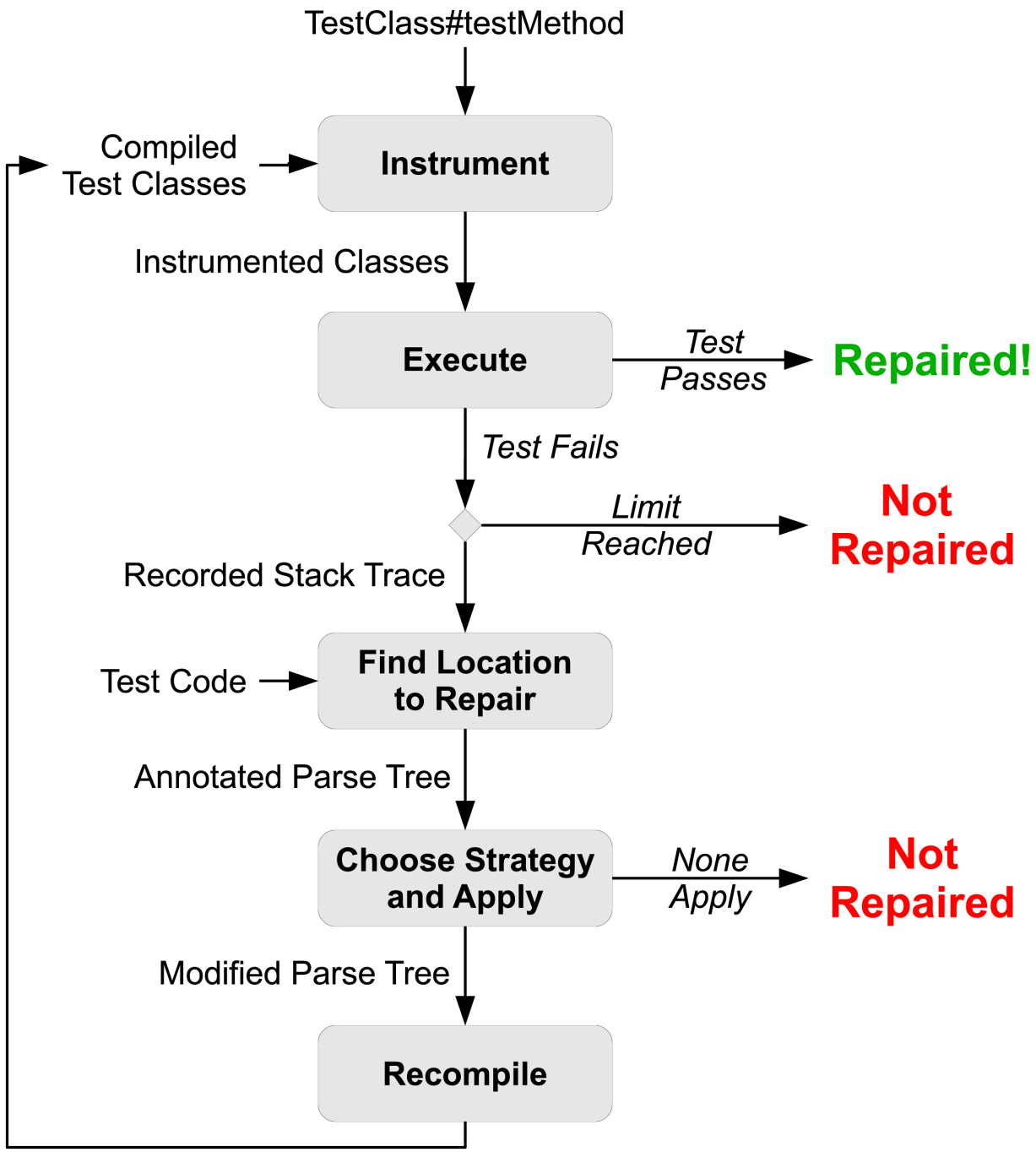
Expand Accessors

```
Product expected = ...
Product actual = ...
{
  assertEquals(                    , actual.getPrice());
  assertEquals(                    , actual.getDescription());
}
```

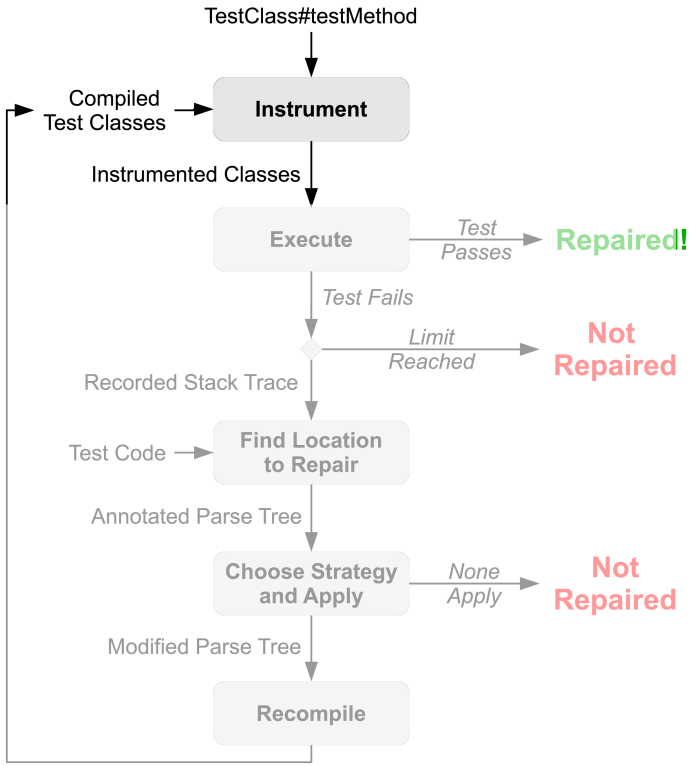
Expand Accessors

```
Product expected = ...
Product actual = ...
{
  assertEquals(expected.getPrice() , actual.getPrice());
  assertEquals("Red pen", actual.getDescription());
}
```




TestClass#testMethod
Compiled
Test Classes
Instrument
Instrumented Classes
Execute
Test
Passes
Repaired!
Test Fails
Limit
Reached
Not
Repaired
Recorded Stack Trace
Test Code
Find Location
to Repair
Annotated Parse Tree
Choose Strategy
and Apply
None
Apply
Not
Repaired
Modified Parse Tree
Recompile

Instrument

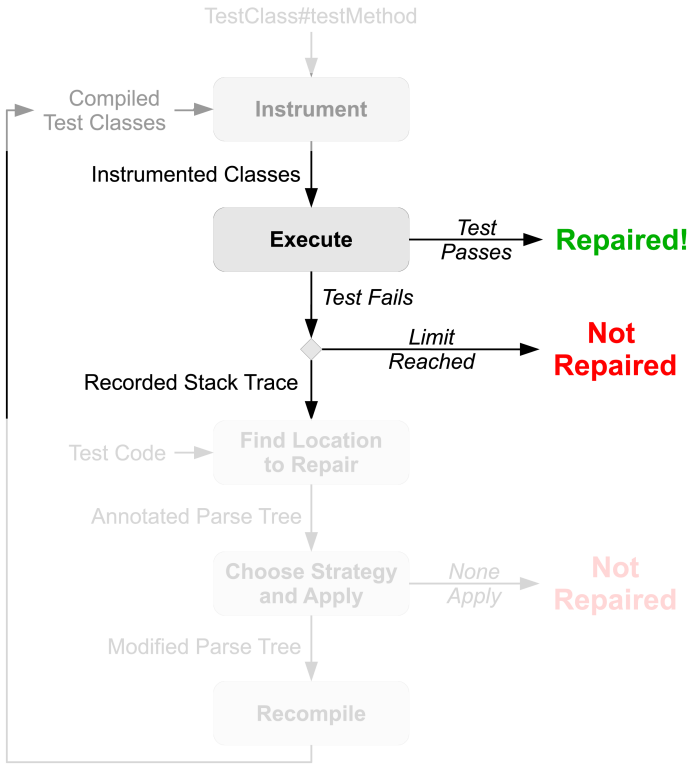



```
public static void assertEquals (
        Object expected,
        Object actual) {
    try {
        // ...assert expected.equals(actual)
    }
    catch (Error e) {
        throw new RecordedAssertFailure(
            e, expected, actual);
    }
}
```

If assertion fails...

...then record values that caused failure

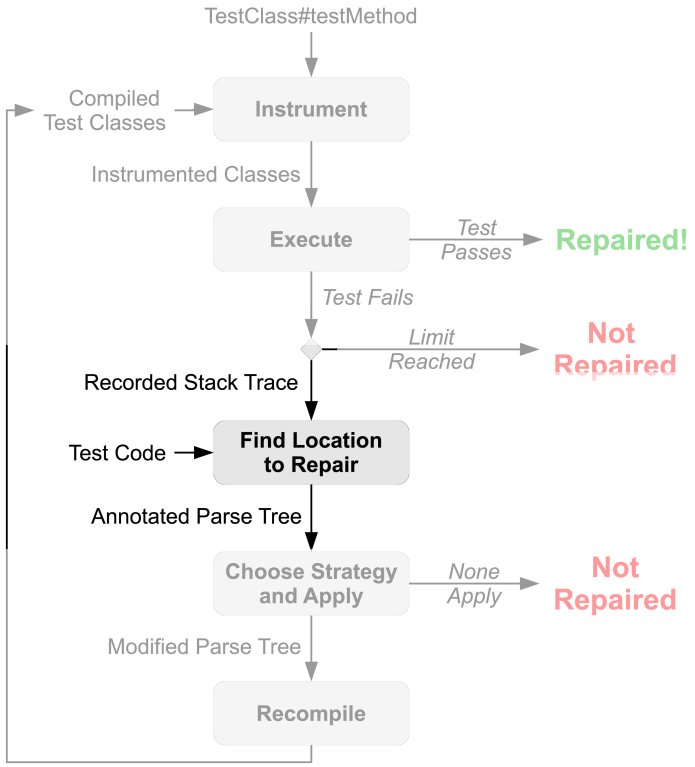
Execute




```
assertEquals(3.0, cart.getTotalPrice());
```

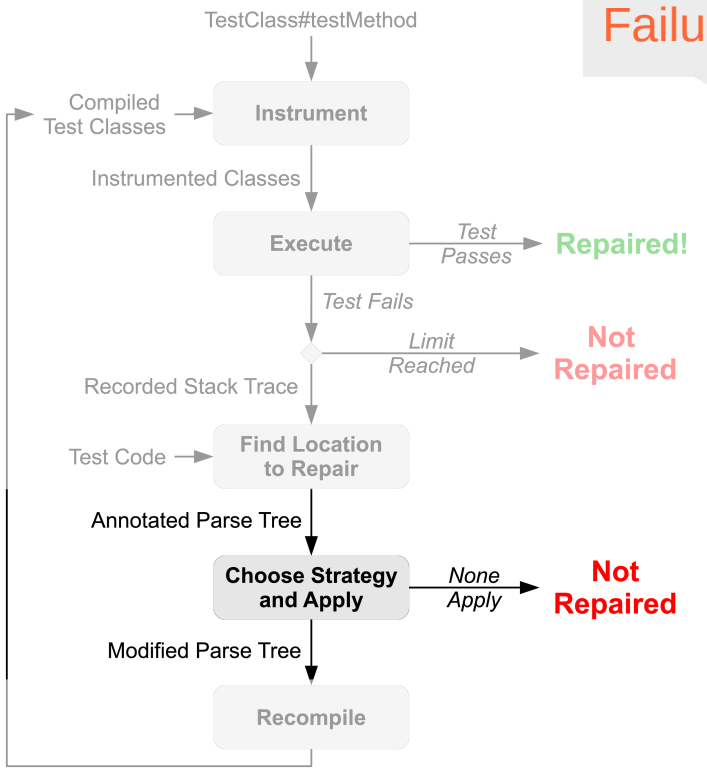
```
throw RecordedAssertFailure(e, 3.0, 6.0);

  edu.illinois.reassert.RecordedAssertFailure:
  org.junit.AssertionFailedError:
  expected:<3.0> but was:<6.0>
    at org.junit.Assert.assertEquals(Assert.java:116)
    at CartTest.testRedPenCoupon(CartTest.java:6)
    ...
```

Find Repair Location




```
edu.illinois.reassert.RecordedAssertFailure:
org.junit.AssertionFailedError:
expected:<3.0> but was:<6.0>
  at org.junit.Assert.assertEquals(Assert.java:116)
  at CartTest.testRedPenCoupon( CartTest.java:6 )
  ...
```

Choose Strategy and Apply



Failure type: assertion failure

Recorded values: literals

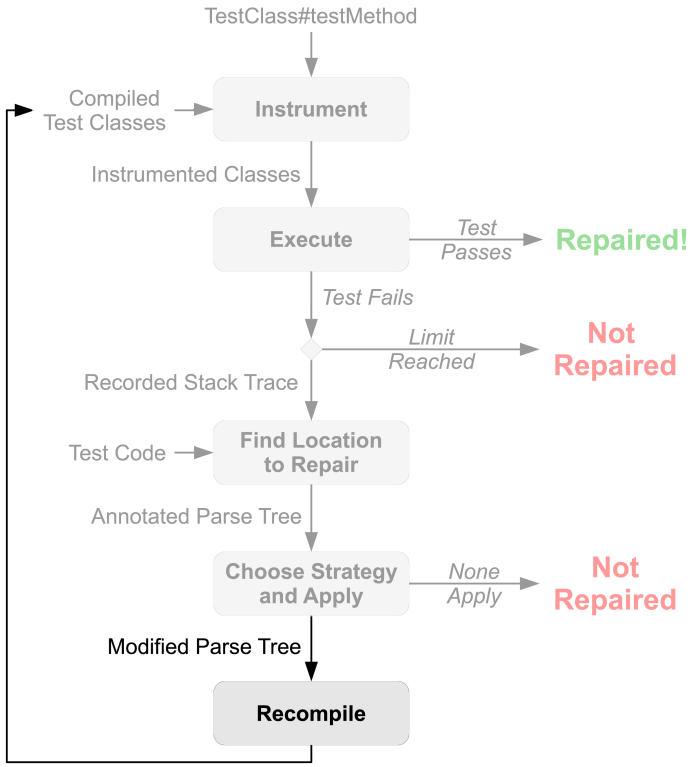
```
assertEquals(3.0, cart.getTotalPrice());
```

Structure: `assertEquals` with literal

∴ ***Replace Literal in Assertion*** **strategy**

```
assertEquals( 6.0 , cart.getTotalPrice());
```

Recompile and Repeat




```
assertEquals( 6.0 , cart.getTotalPrice());
```

```
assertEquals(
    "Discount: -$1.00, Total: $3.00",
    cart.getPrintedBill());
```

Evaluating ReAssert

Q1: How many failures can ReAssert **repair**?

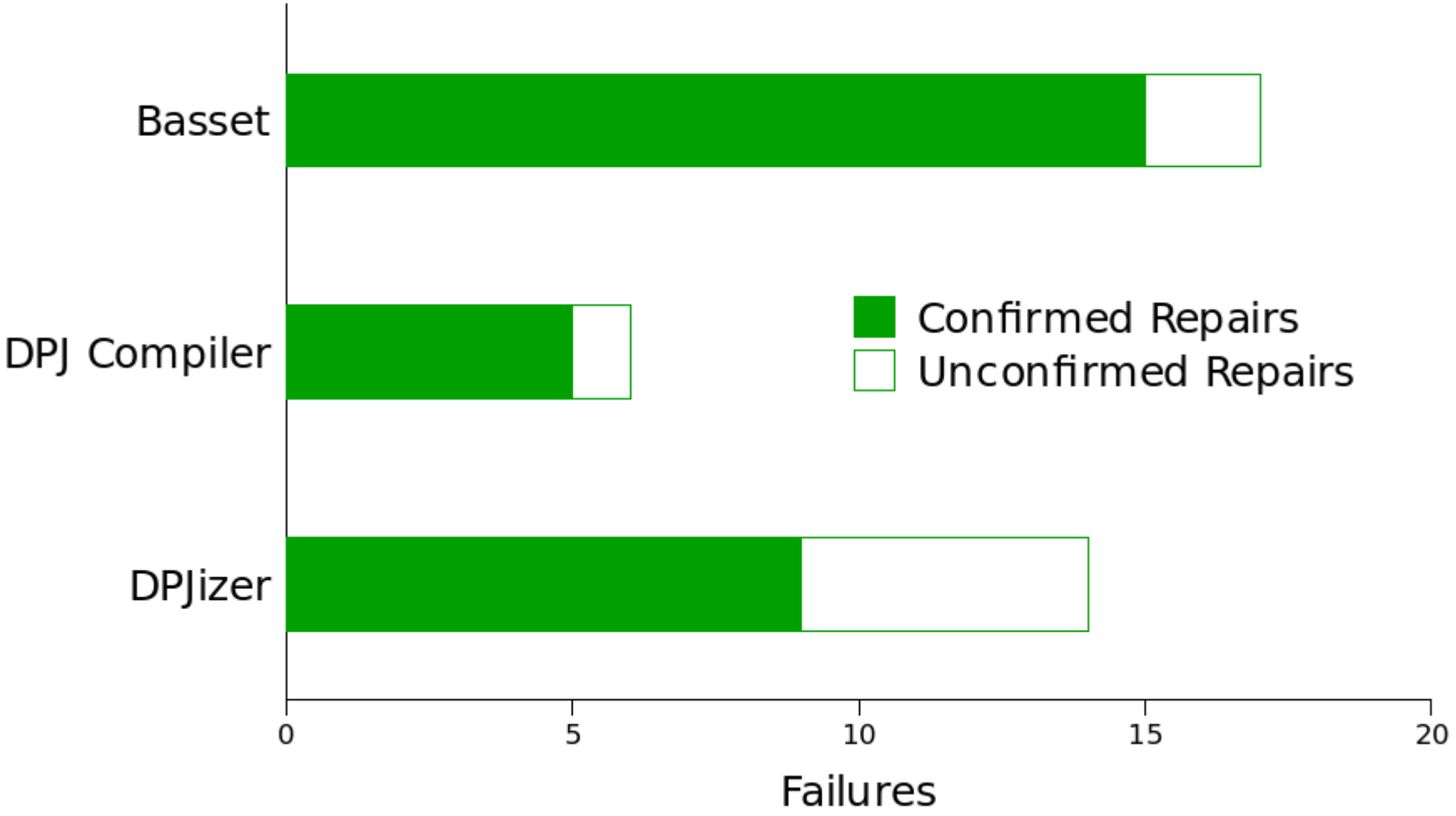
Q2: Are ReAssert's suggested repairs **useful**?

Q3: Does ReAssert **reveal** or **hide** regressions?

Evaluating ReAssert

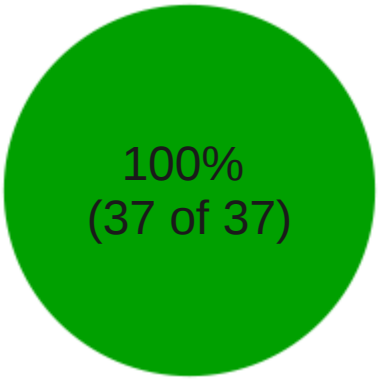
	Repairs?	Useful?	Regressions?
Case Studies	✓	✓	✓
Controlled User Study	✓	✓	✓
Failures in Open-Source Software	✓		

Case Studies

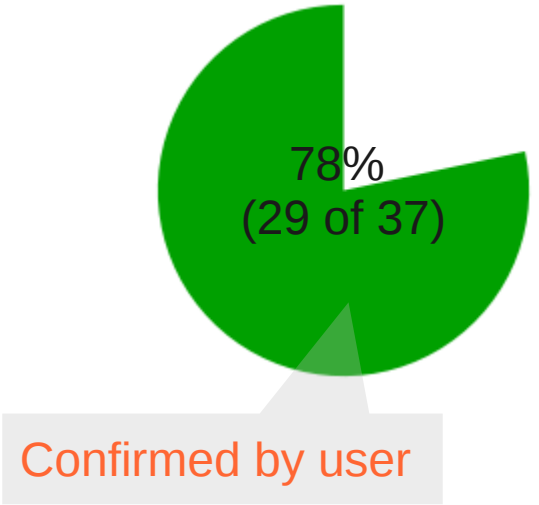


Case Studies

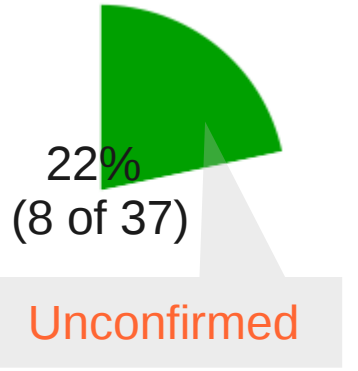
Repairs?



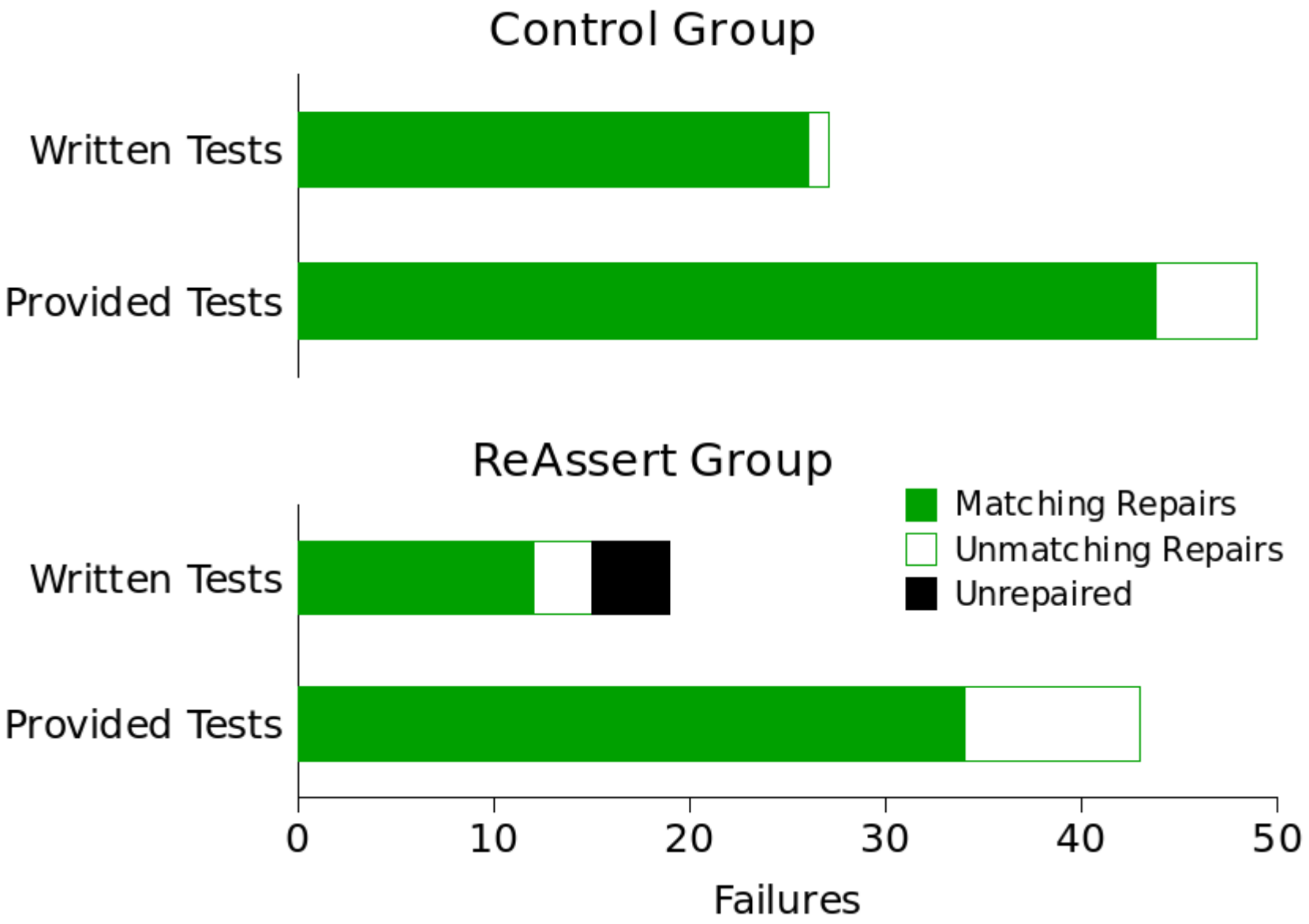
Useful?



Regressions?

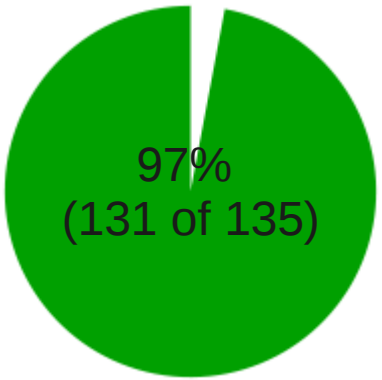


Controlled User Study

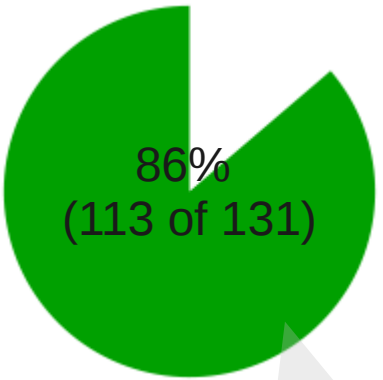


Controlled User Study

Repairs?



Useful?



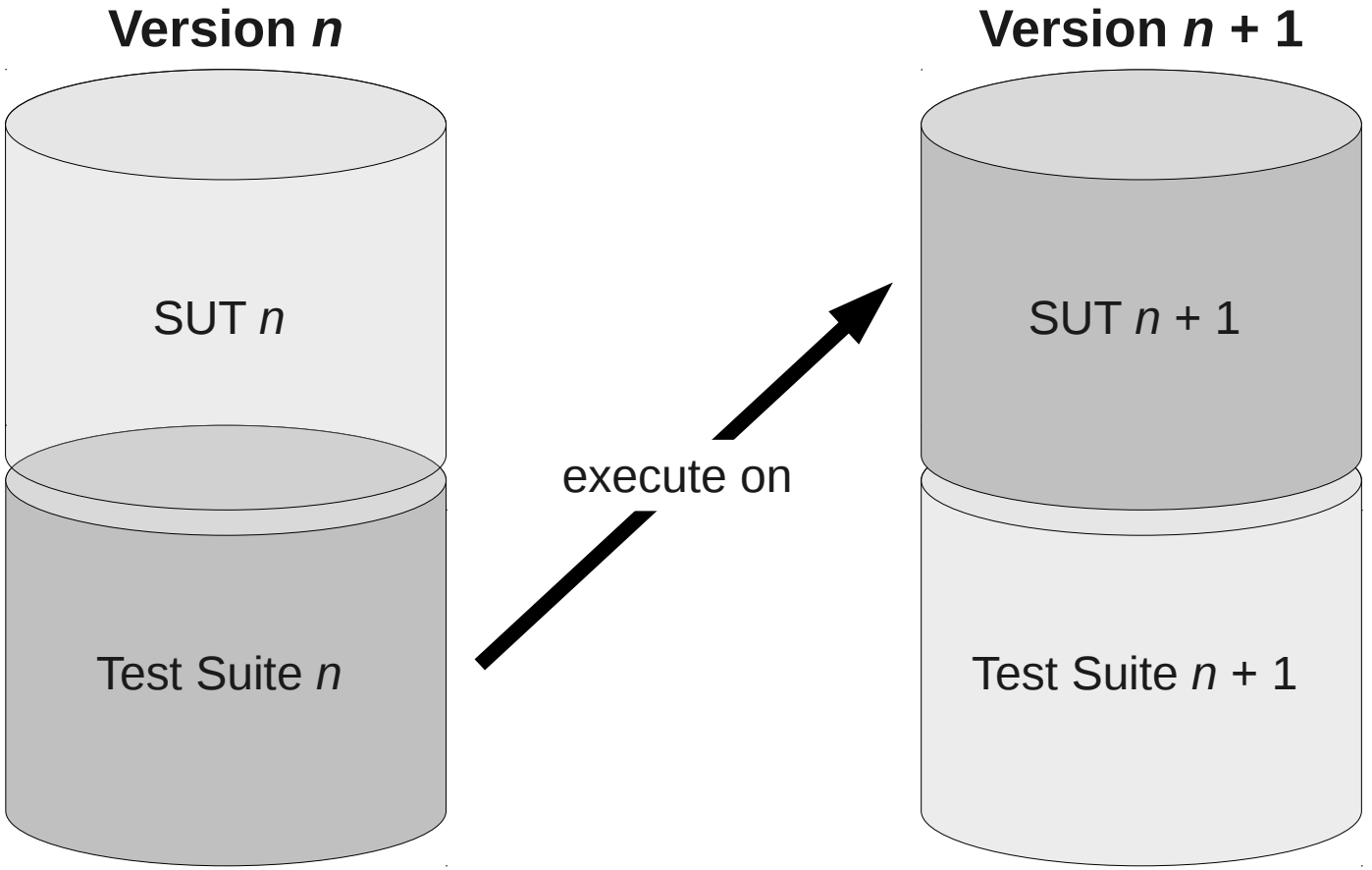
Matching repairs

Regressions?

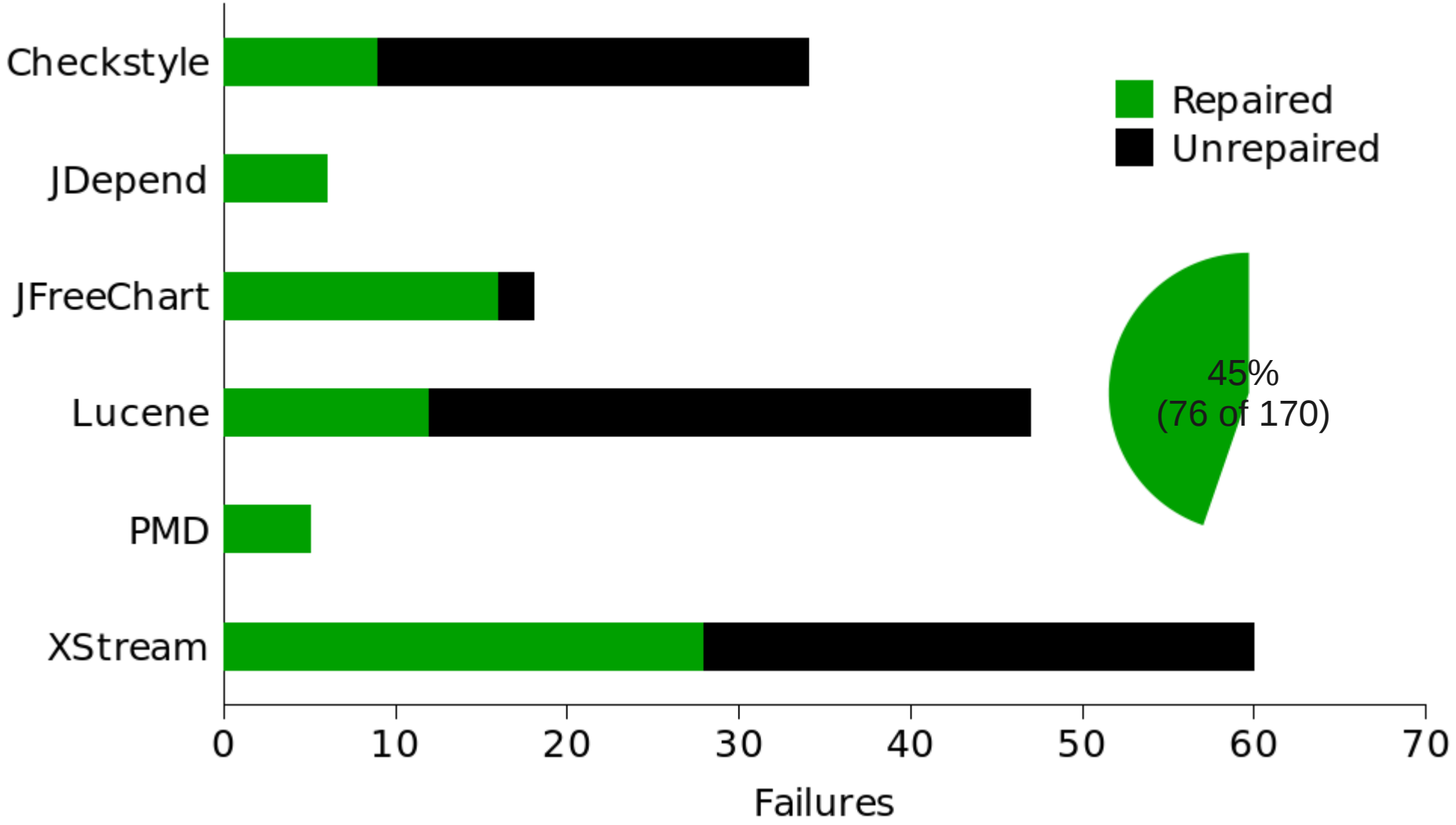
9%
(12 of 131)

vs. 8 introduced by the control group

Failures in Open-Source Software



Failures in Open-Source Software



Evaluating ReAssert

Q1: How many failures can ReAssert **repair**?

45% in open source software

Q2: Are ReAssert's suggested repairs **useful**?

78% to 86% approved by users

Q3: Does ReAssert **reveal** or **hide** regressions?

Both, comparable to manual edits

Unrepairable Failures

- Nondeterminism

```
assertEquals(..., cart.getPurchaseDate());
```

- Multiple contexts

```
for (Product product : cart.getProducts()) {
    assertEquals(3.0, product.getPrice());
}
```

ReAssert's Limitations

- Multiple Expected Values

```
double expTotal;
if (HAS_TAX) {
   expTotal = 3.15;
}
else {
   expTotal = 3.0;
}
assertEquals(expTotal, cart.getTotalPrice());
```

- Computed Expected Value

```
double total = 3.0;
String expBill = "Total: $" + total;
assertEquals(expBill, cart.getPrintedBill());
```

- Expected Object Comparison

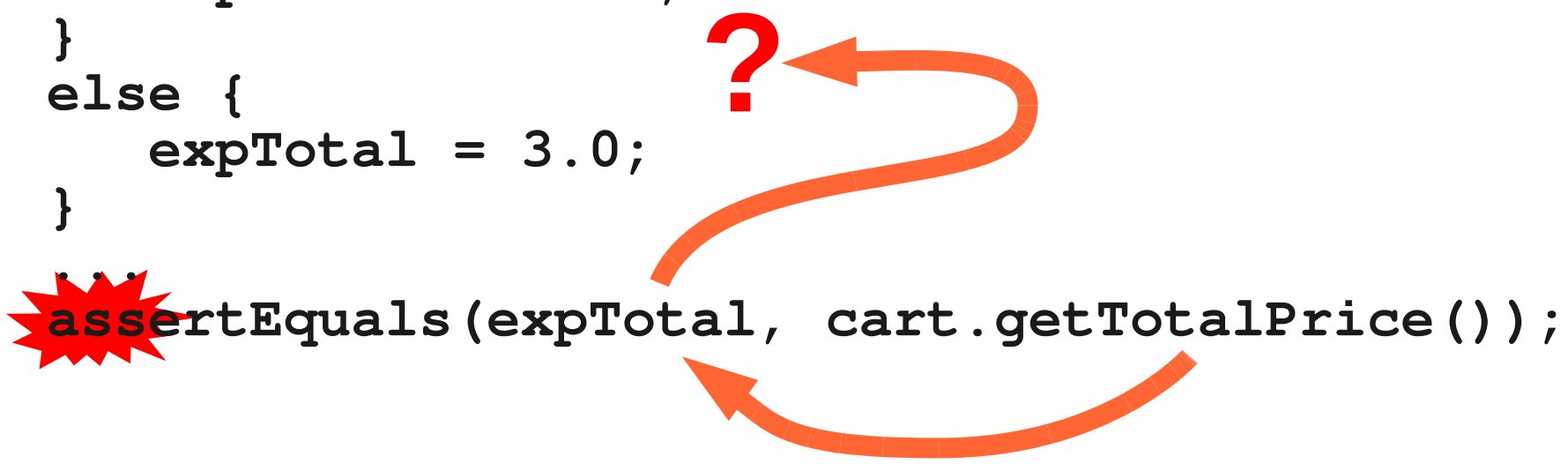
```
Product expProduct = new Product("Red pen", 3.0);
assertEquals(expProduct, cart.getItem(0));
```

Multiple Expected Values

```
double expTotal;
if (HAS_TAX) {
    expTotal = 3.15;
}
else {
    expTotal = 3.0;
}
...
assertEquals(expTotal, cart.getTotalPrice());
```

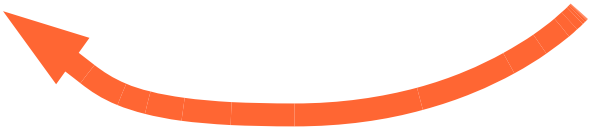
Multiple Expected Values

```
double expTotal;
if (HAS_TAX) {
    expTotal = 3.15;
}
else {
    expTotal = 3.0;
}
...
assertEquals(expTotal, cart.getTotalPrice());
```



ReAssert's Naïve Repair

```
double expTotal;
if (HAS_TAX) {
    expTotal = 3.15;
}
else {
    expTotal = 3.0;
}
...
assertEquals(6.0, cart.getTotalPrice());
```



Insight Many failures can be repaired by **changing literal values** in test code

Problem ReAssert **could not determine** which literals needed to change and how

Hypothesis **Symbolic execution** can discover literals that cause a test to pass

[**On Test Repair Using Symbolic Execution**
Brett Daniel, Tihomir Gvero, Darko Marinov
ISSTA 2010. Trento, Italy]

Symbolic Execution

Dynamic symbolic execution

Nondeterministic choice generator produces concrete values

```
int input = PexChoose.Value<int>("i");
if (input < 5) {
  throw new Exception();
}
```

Branches introduce *path constraints*

Solve constraints to execute alternate paths

http://research.microsoft.com/en-us/projects/pex/
http://research.microsoft.com/en-us/um/redmond/projects/z3/

Symbolic Execution in Testing

Test Generation

Find values that make a **program fail**
(or achieve coverage)

Test Repair

Find values that make a **test pass**

Symbolic Test Repair

1) Find location of failure
2) Determine “expected” computation
3) Make “expected-side” literals symbolic
4) Execute and accumulate constraints
5) Solve constraints and replace in code

```
double expTotal;
if (HAS_TAX) {
   expTotal = 3.15;
}
else {
   expTotal = 3.0;
}
assertEquals(
  expTotal,
  cart.getTotalPrice());
```

Symbolic Test Repair

1) Find location of failure
2) Determine “expected” computation
3) Make “expected-side” literals symbolic
4) Execute and accumulate constraints
5) Solve constraints and replace in code

```
double expTotal;
if (HAS_TAX) {
   expTotal = 3.15;
}
else {
   expTotal = 3.0;
}
assertEquals(
  expTotal,
  cart.getTotalPrice());
```

Symbolic Test Repair

1) Find location of failure
2) Determine “expected” computation
3) Make “expected-side” literals symbolic
4) Execute and accumulate constraints
5) Solve constraints and replace in code

```
double expTotal;
if (HAS_TAX) {
   expTotal = 3.15;
}
else {
   expTotal = 3.0;
}
assertEquals(
  expTotal,
  cart.getTotalPrice());
```

Symbolic Test Repair

1) Find location of failure
2) Determine “expected” computation
3) Make “expected-side” literals symbolic
4) Execute and accumulate constraints
5) Solve constraints and replace in code

```
double expTotal;
if (HAS_TAX) {
   expTotal = PexChoose.
      Value<double>(“e1”);
}
else {
   expTotal = PexChoose.
      Value<double>(“e2”);
}
assertEquals(
  expTotal,
  cart.getTotalPrice());
```

Symbolic Test Repair

1) Find location of failure
2) Determine “expected” computation
3) Make “expected-side” literals symbolic
4) Execute and accumulate constraints
5) Solve constraints and replace in code

```
double expTotal;
if (HAS_TAX) {
   expTotal = PexChoose.
      Value<double>(“e1”) ;
}
else {
   expTotal = PexChoose.
      Value<double>(“e2”) ;
}
assertEquals(
  expTotal,
  cart.GetTotalPrice());
```

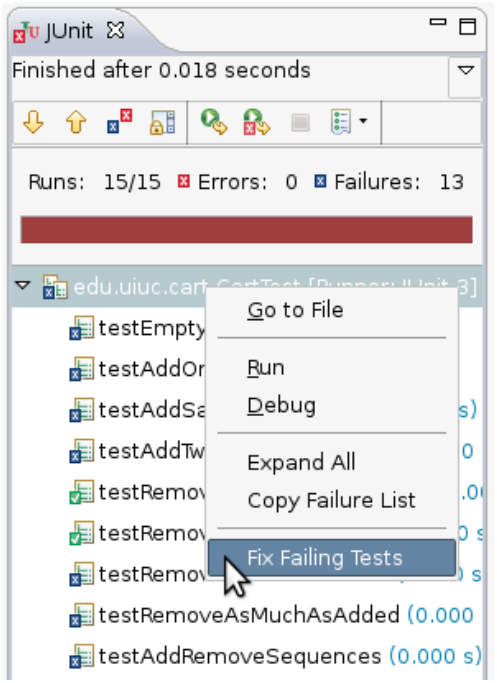
e2 == 6.0

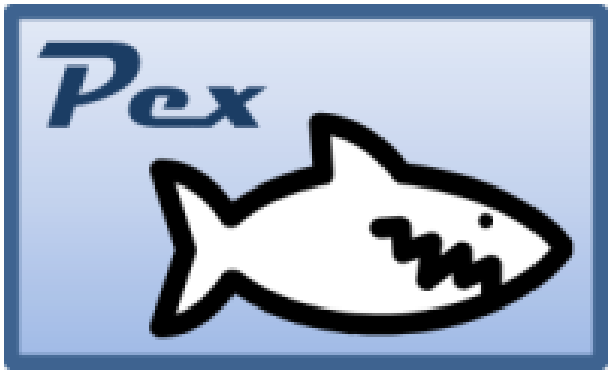
Symbolic Test Repair

1) Find location of failure
2) Determine “expected” computation
3) Make “expected-side” literals symbolic
4) Execute and accumulate constraints
5) Solve constraints and replace in code

```
double expTotal;
if (HAS_TAX) {
   expTotal = 3.15;
}
else {
   expTotal = 6.0;
}
assertEquals(
  expTotal,
  cart.getTotalPrice());
```

Implementation Mismatch





- Java
- Eclipse

- .NET
- Visual Studio

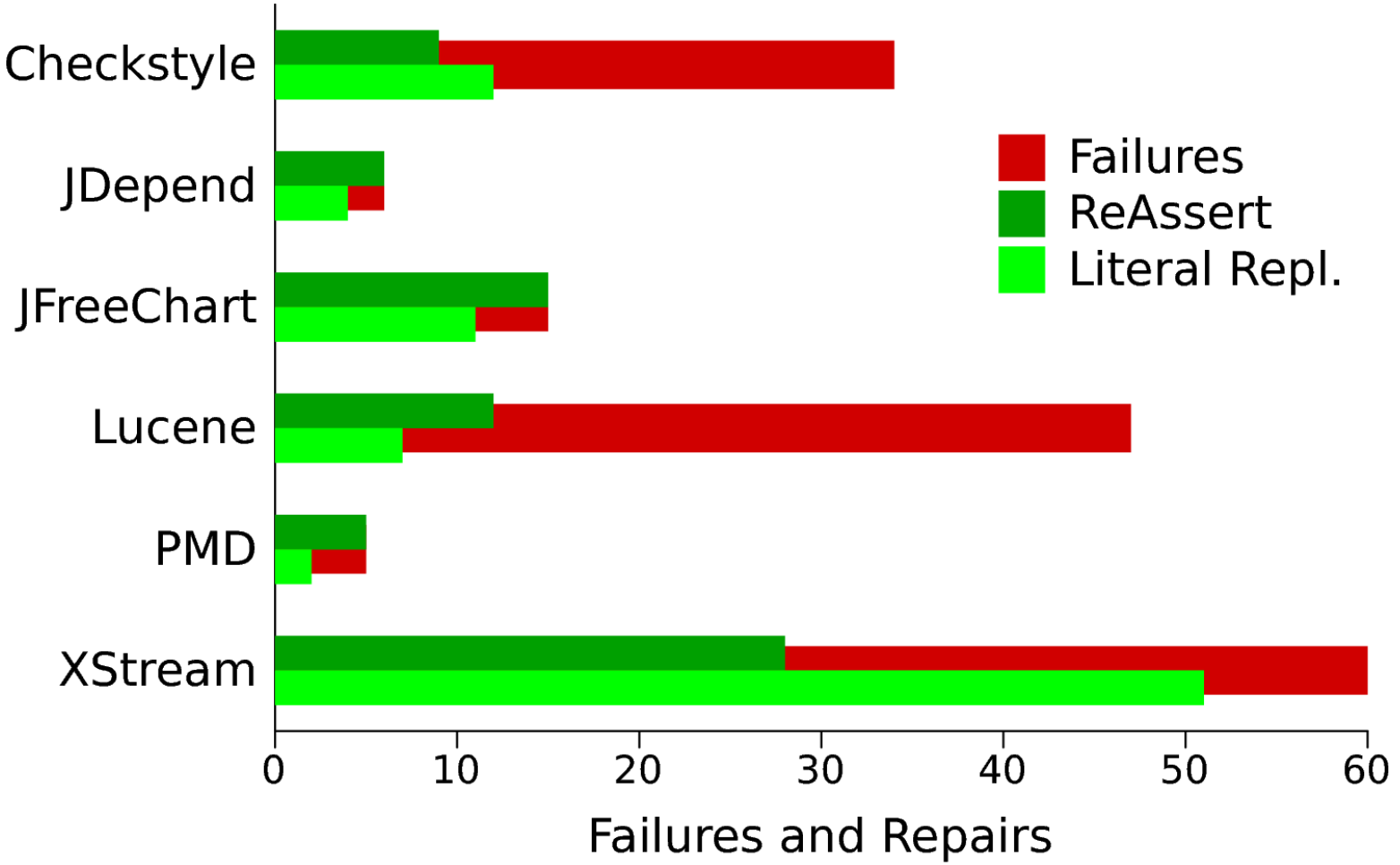
Evaluating Symbolic Test Repair

Q4: How many failures can ideal **literal replacement** repair?

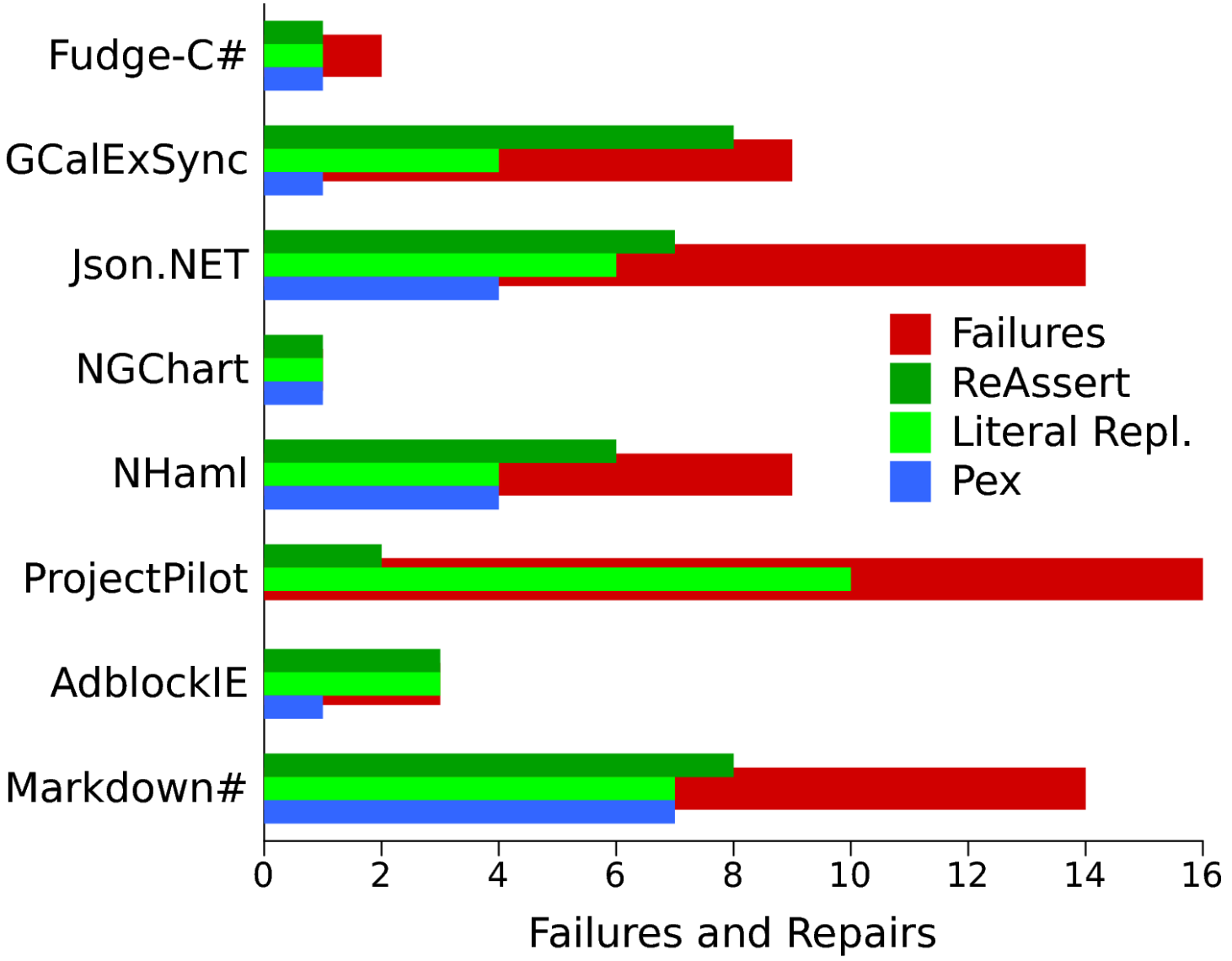
Q5: How do ReAssert and literal replacement **compare**?

Q6: Can **symbolic execution** discover literals?

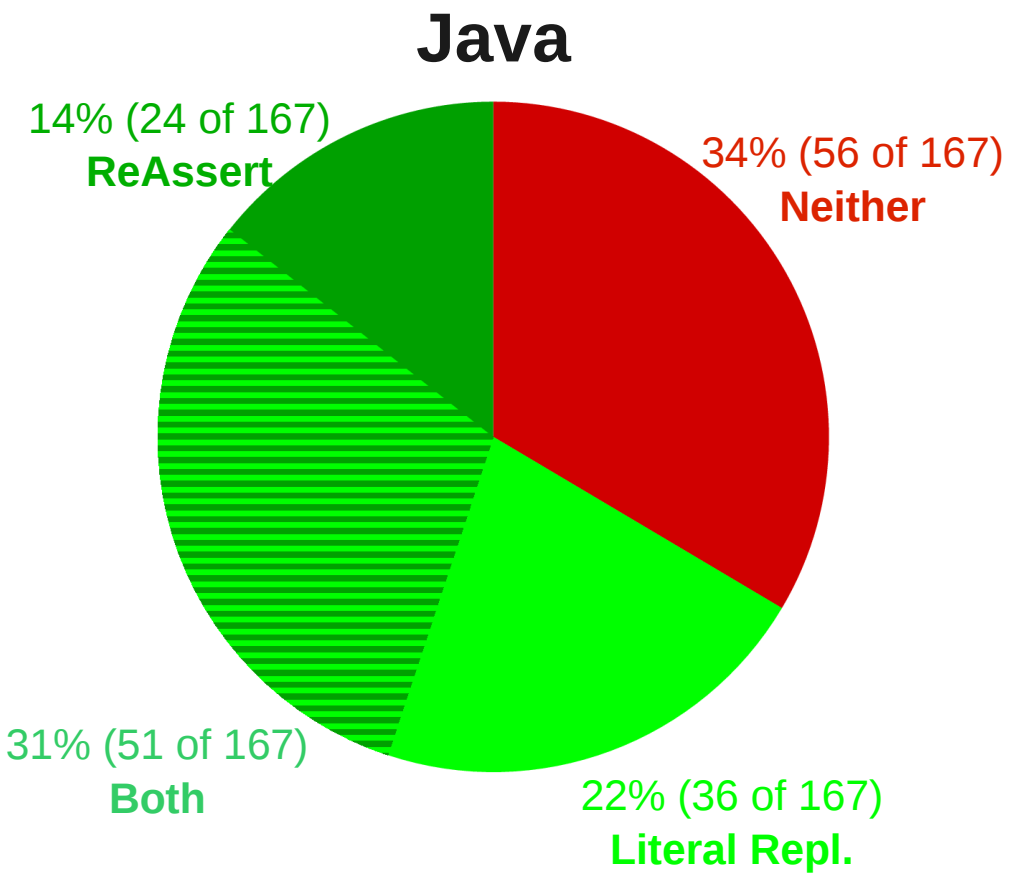
Open Source Software - Java

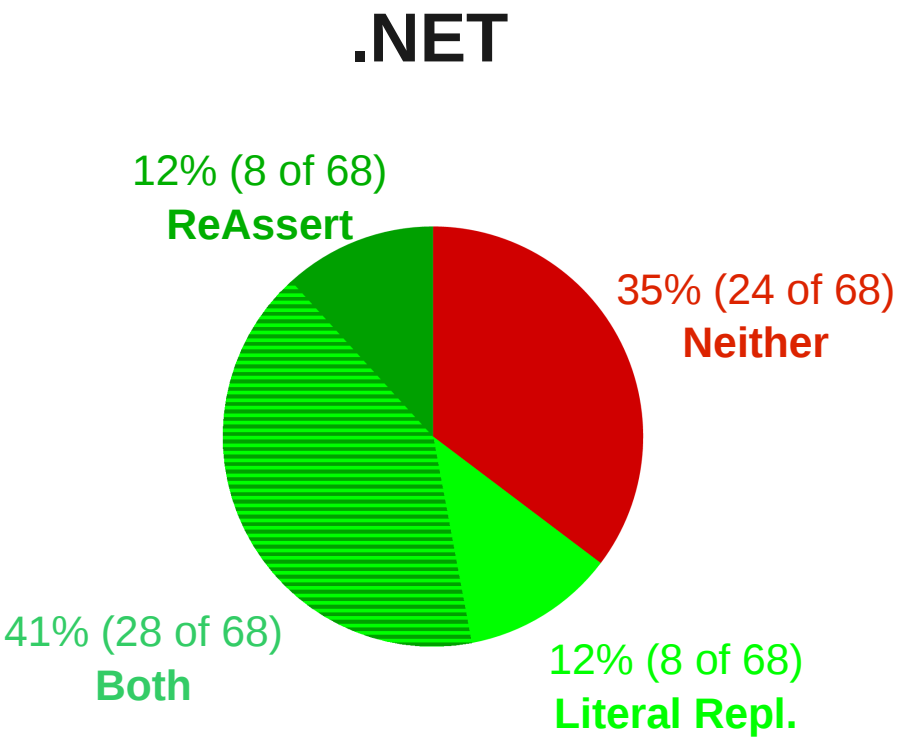


Open Source Software - .NET

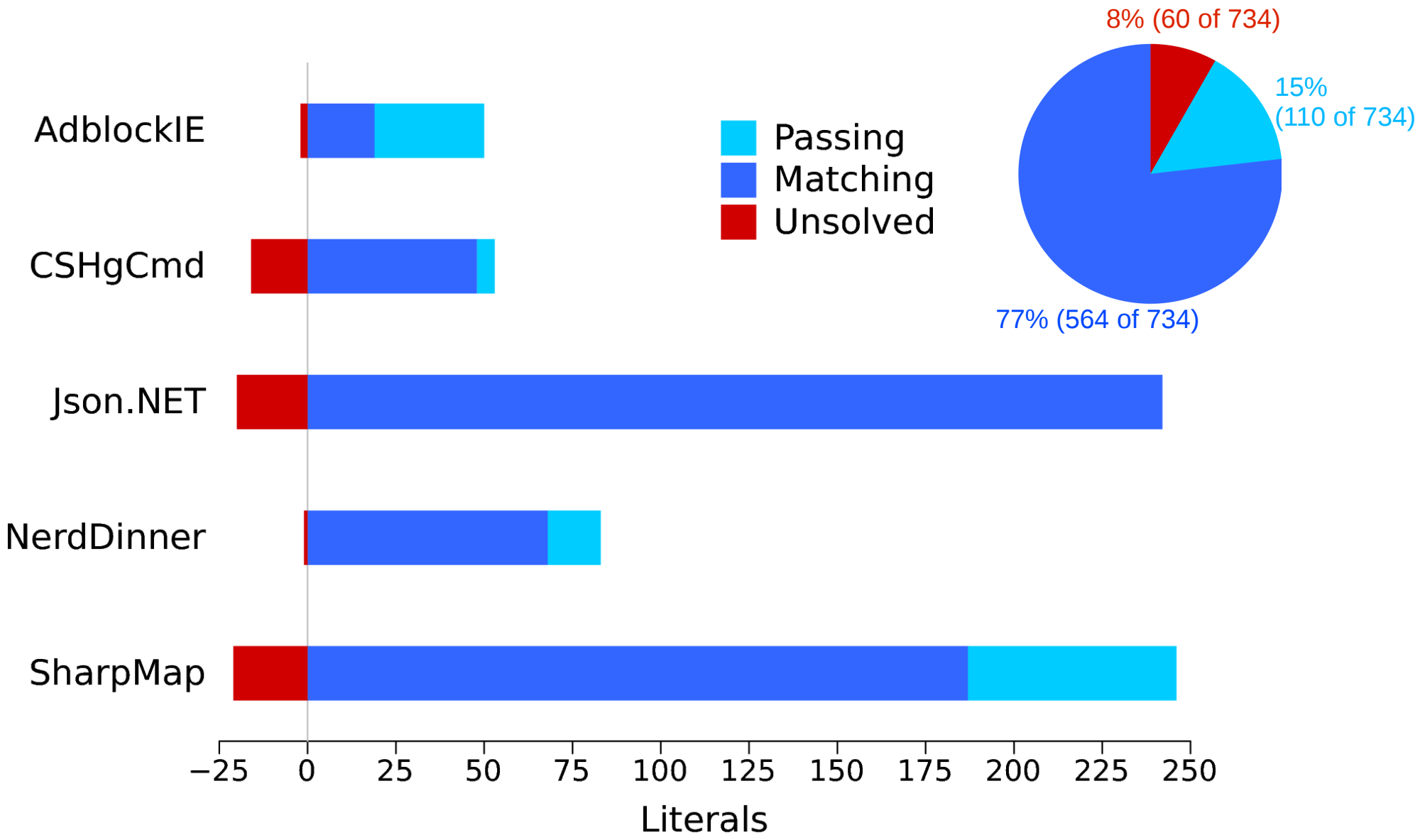


ReAssert vs. Literal Replacement





Recreate Literals



Evaluating Symbolic Test Repair

Q4: How many failures can ideal **literal replacement** repair?

About half

Q5: How do ReAssert and literal replacement **compare**?

12% to 22% improvement when combined

Q6: Can **symbolic execution** discover literals?

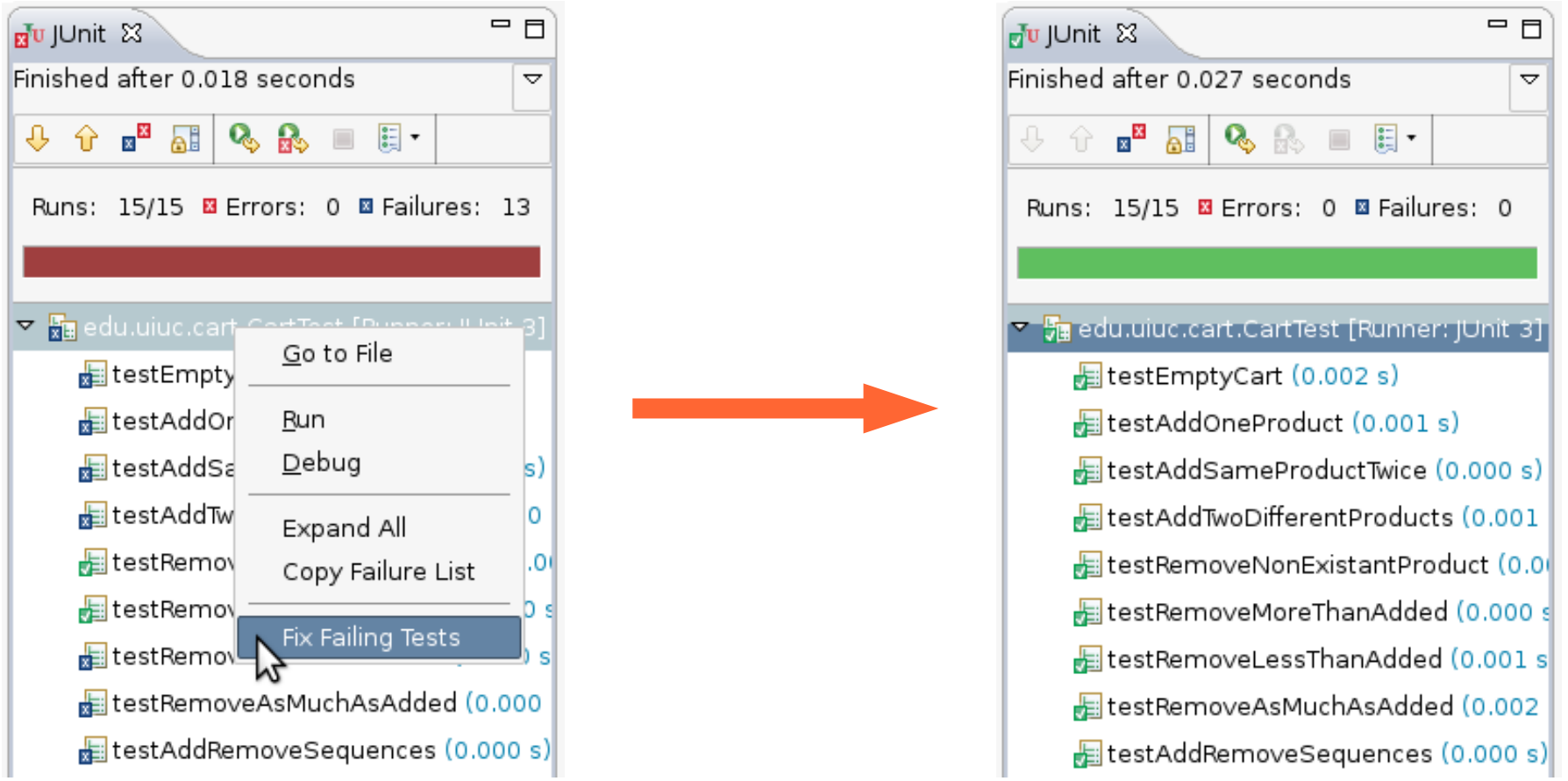
Yes: 52% to 92% of literals

Thanks

- Rob Bocchino, Bobak Hadidi, Steve Lauterburg, and Mohsen Vakilian for their case studies
- Nikolai Tillmann for help with Pex
- Milos Gligoric, Munawar Hafiz, Viktor Kuncak, Yun Young Lee, and Samira Tasharofi for valuable comments
- Anonymous colleagues for potential evolutions
- Anonymous user study participants

- Supported in part by the US National Science Foundation under Grant No. CCF-0746856

ReAssert



http://mir.cs.illinois.edu/reassert