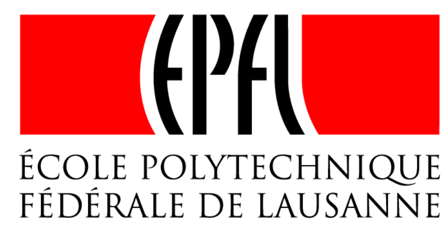
On Test Repair Using Symbolic Execution

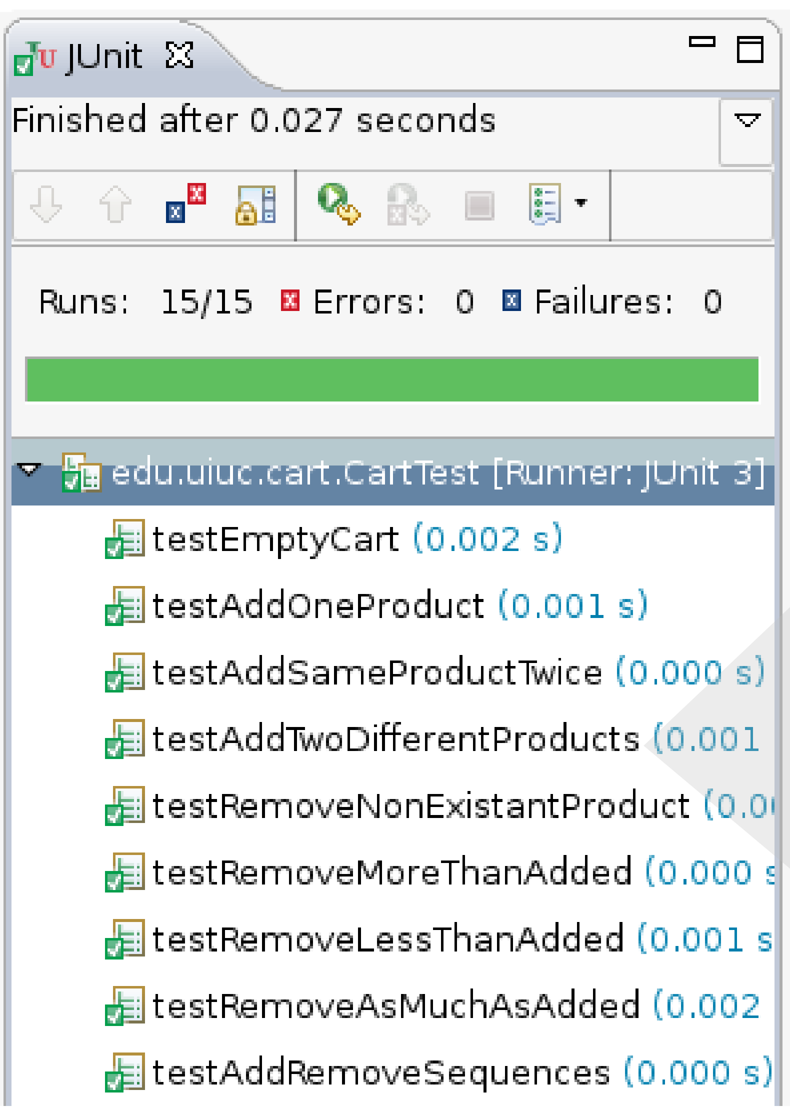
Brett Daniel
Darko Marinov

Tihomir Gvero



ISSTA 2010: International Symposium on Software Testing and Analysis
July 15, 2010
Trento, Italy

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

JUnit

Finished after 0.024 seconds

Runs: 15/15 Errors: 0 Failures: 13

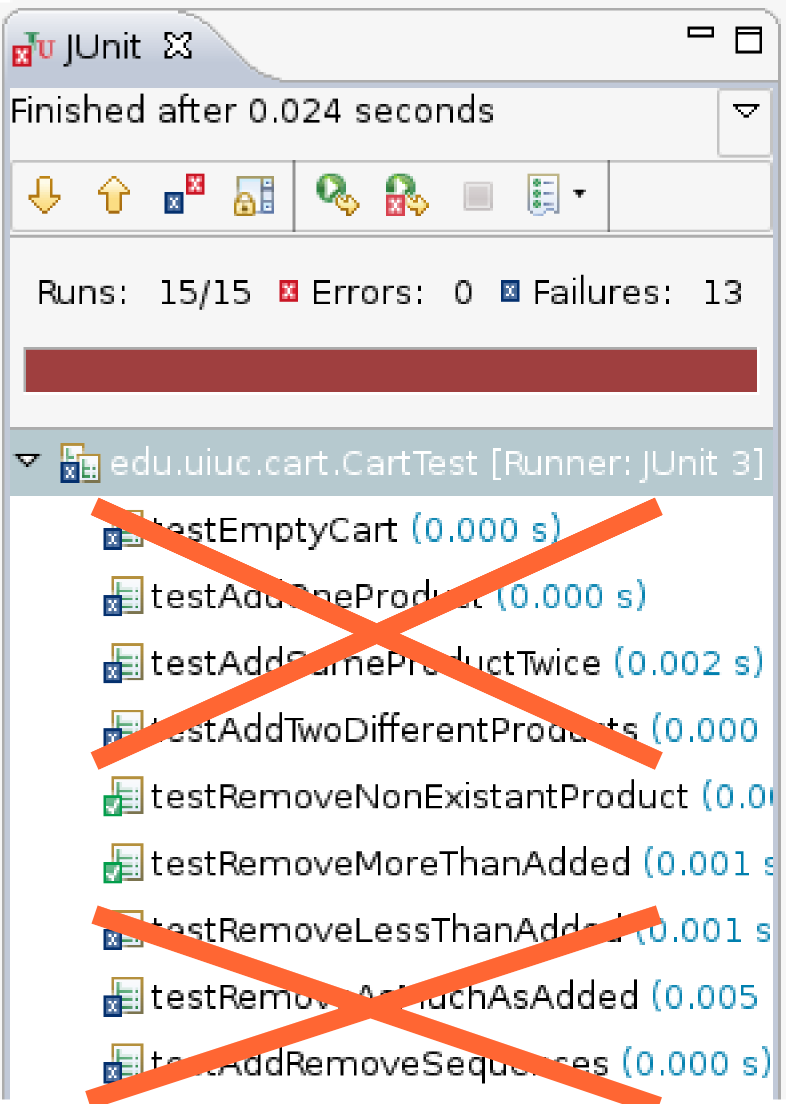
edu.uiuc.cart.CartTest [Runner: JUnit 3]

- testEmptyCart (0.000 s)
- testAddOneProduct (0.000 s)
- testAddSameProductTwice (0.002 s)
- testAddTwoDifferentProducts (0.000
- testRemoveNonExistantProduct (0.0
- testRemoveMoreThanAdded (0.001 s
- testRemoveLessThanAdded (0.001 s
- testRemoveAsMuchAsAdded (0.005
- testAddRemoveSequences (0.000 s)

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

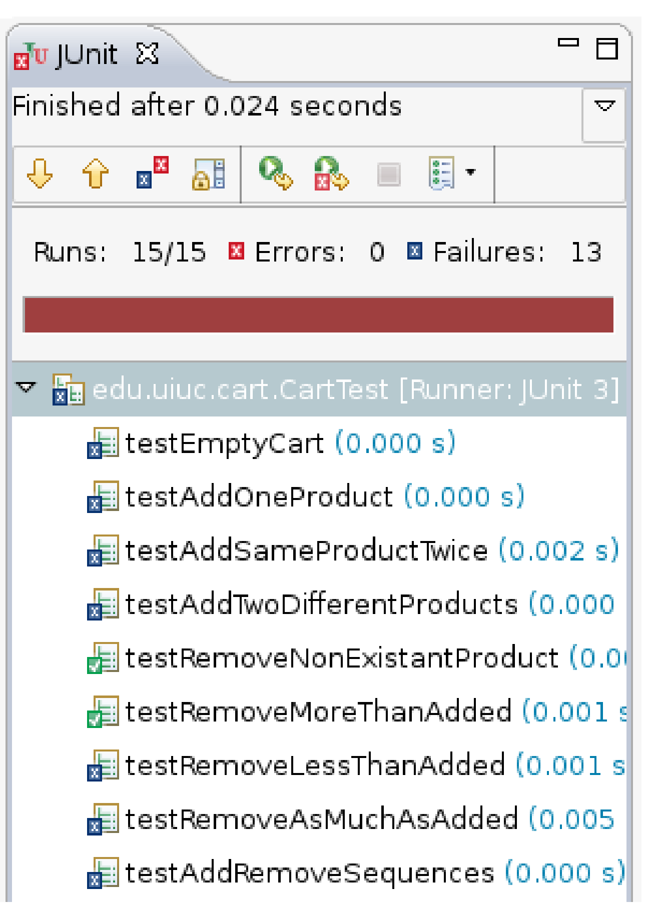
```
public void testAddTwoDifferentProducts() {
    Cart cart = ...
    assertEquals(3.0, cart.getTotalPrice());
    assertEquals(
        "Discount: -$3.00, Total: $3.00",
        cart.getPrintedBill());
}
```

Delete Broken Tests?

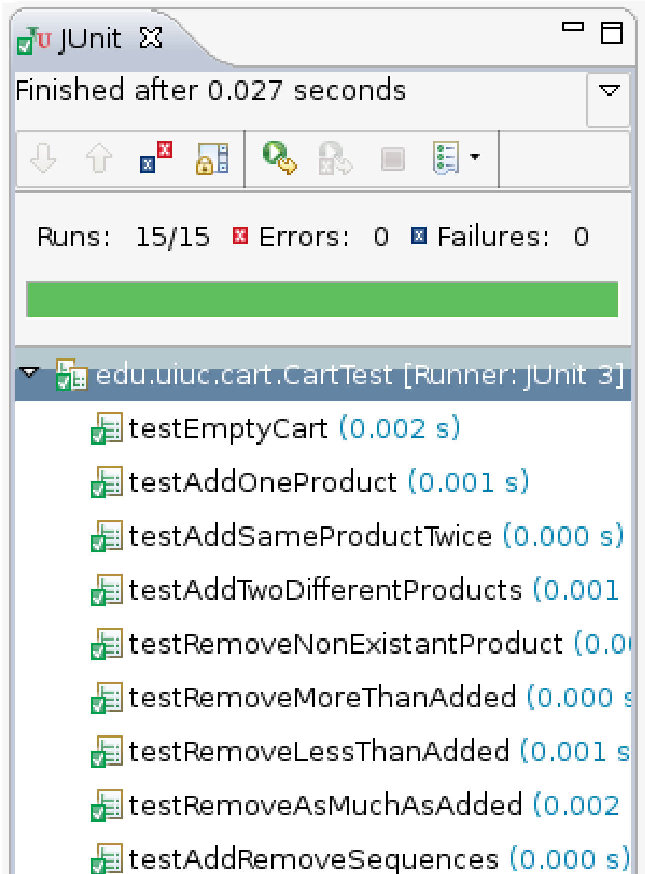


But that reduces the quality of the test suite

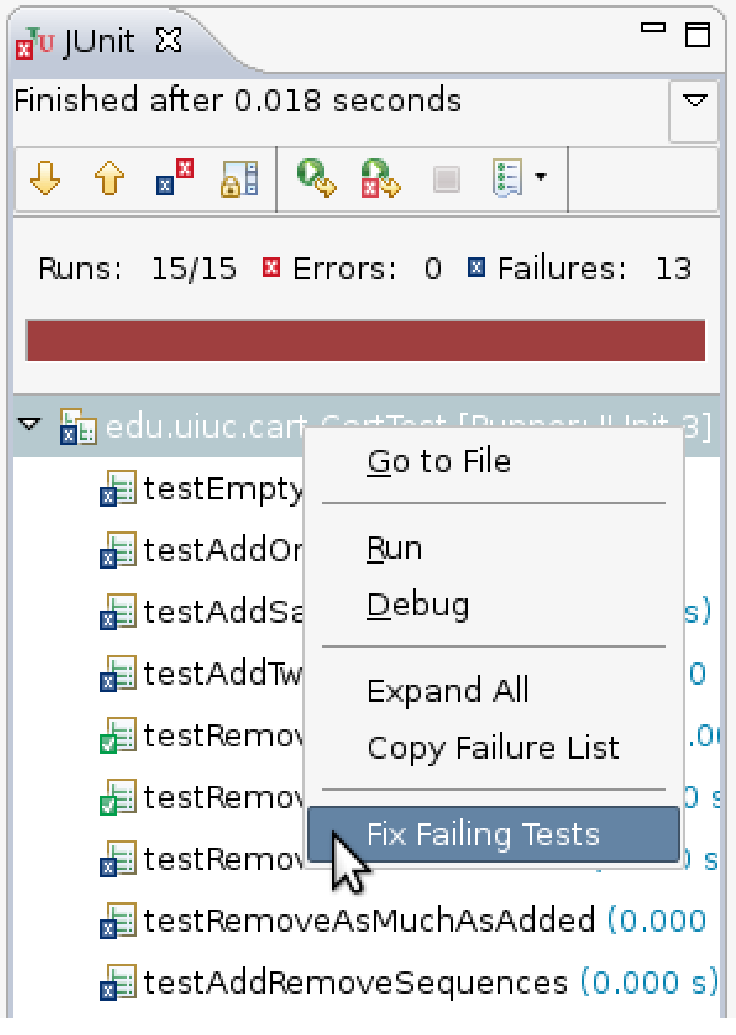
Repairing Tests is Preferable



But that requires a lot of time and effort

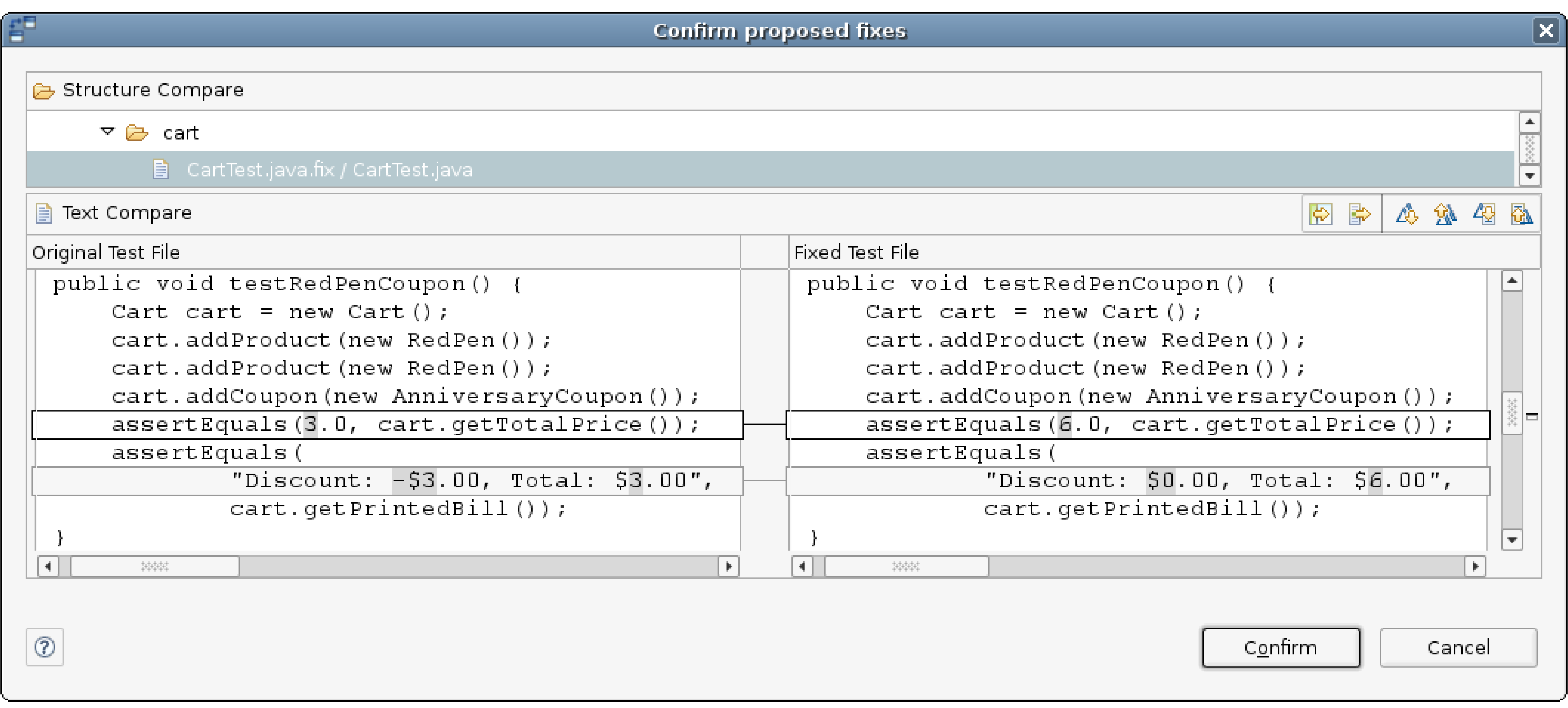


ReAssert Suggests Repairs

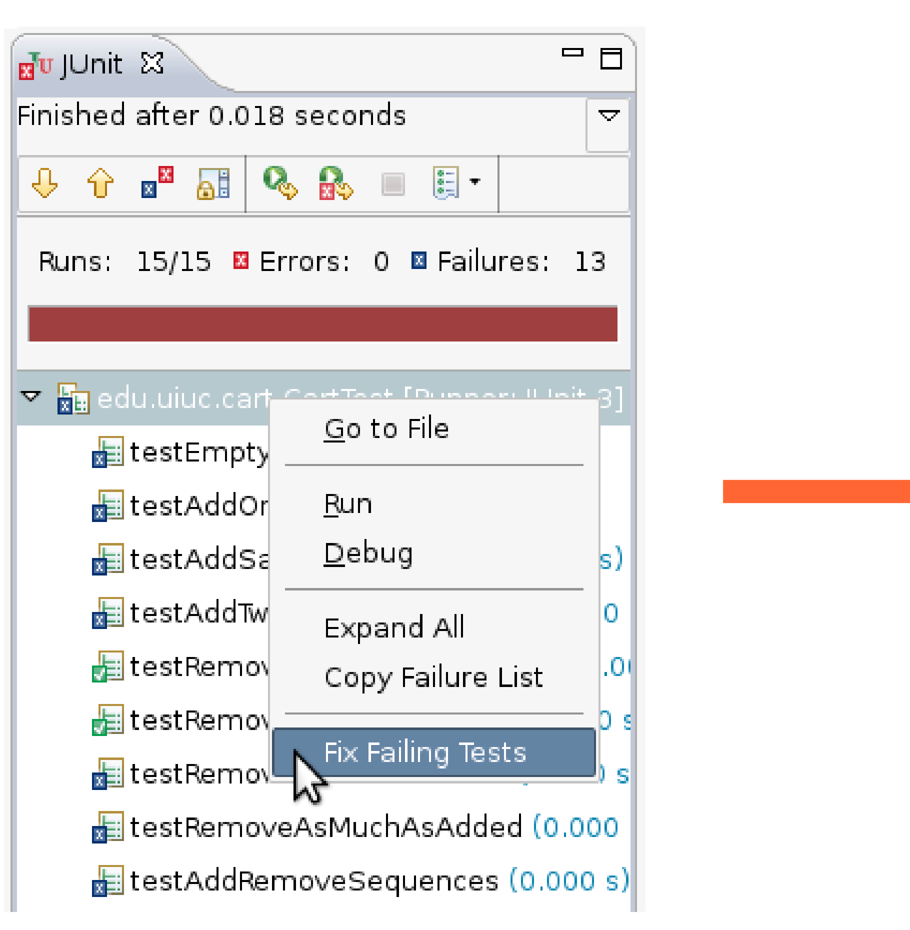


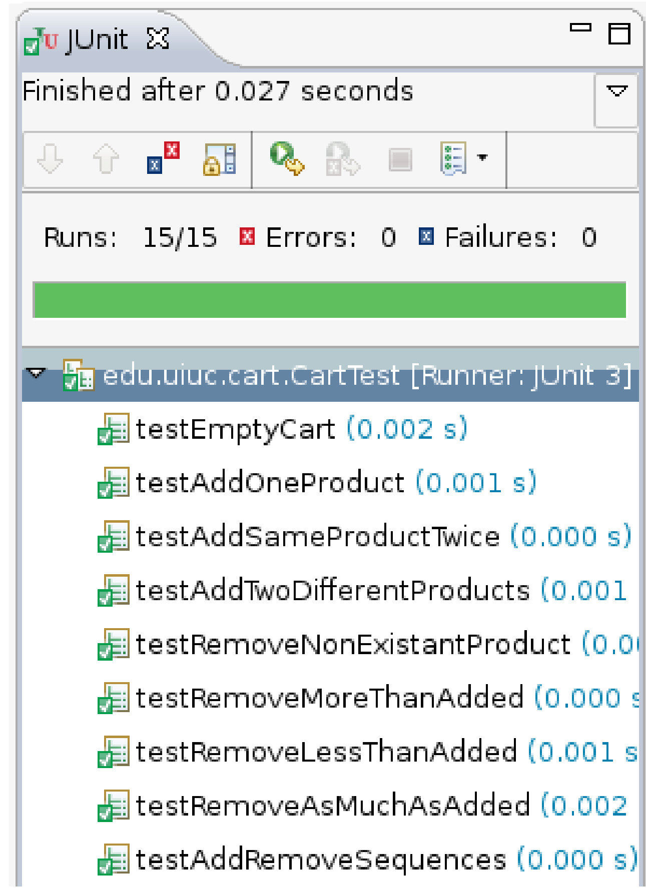
[**ReAssert: Suggesting Repairs for Broken Unit Tests**
Brett Daniel, Vilas Jagannath, Danny Dig, Darko Marinov
ASE 2009, Auckland, New Zealand]

Confirm or Reject Suggestions



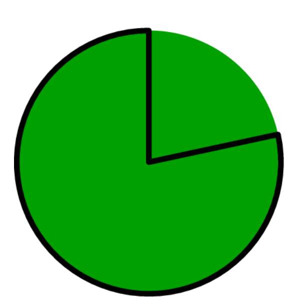
ReAssert Reduces Effort





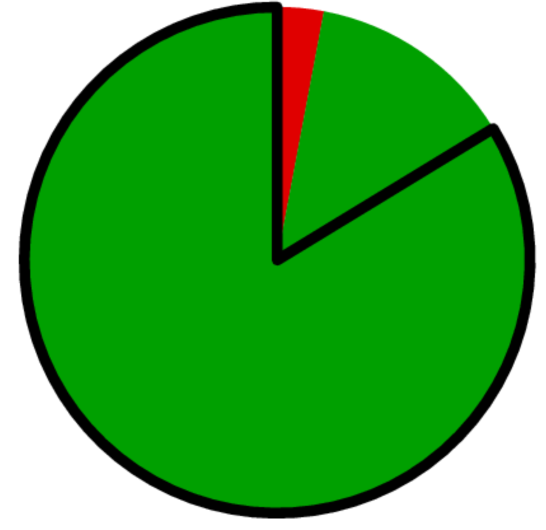
Does ReAssert Work?

Case Studies



100% (37 of 37) **Repaired**
78% (29 of 37) **Useful**

User Study



97% (131 of 135) **Repaired**
86% (113 of 131) **Useful**
3% (4 of 135) **Not Repaired**

Open Source

45% (75 of 167) **Repaired**
55% (92 of 167) **Not Repaired**

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

Simple Assertion Failure

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

Replace Literal

Replace in code

Record actual value

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

Temporary Variable

```
double expTotal = 3.0;

...

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

Trace Declaration-Use Path

```
double expTotal = 6.0;

...

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

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());
```

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

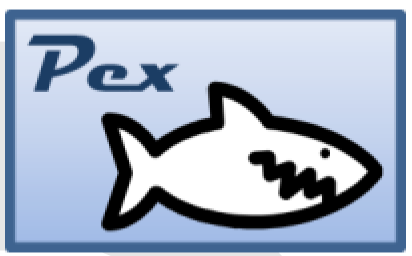
```
double total = 3.0;
String expBill = "Total: $" + total;
assertEquals(expBill, cart.getPrintedBill());
```

- Expected Object Comparison

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

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());
```

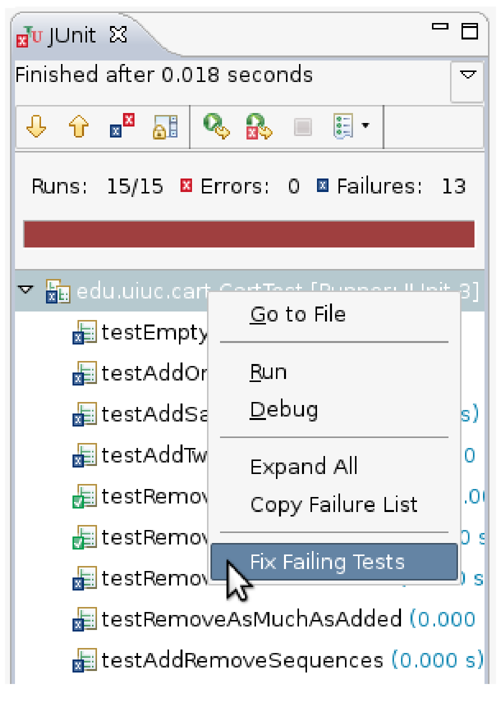
e2 == 6.0

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

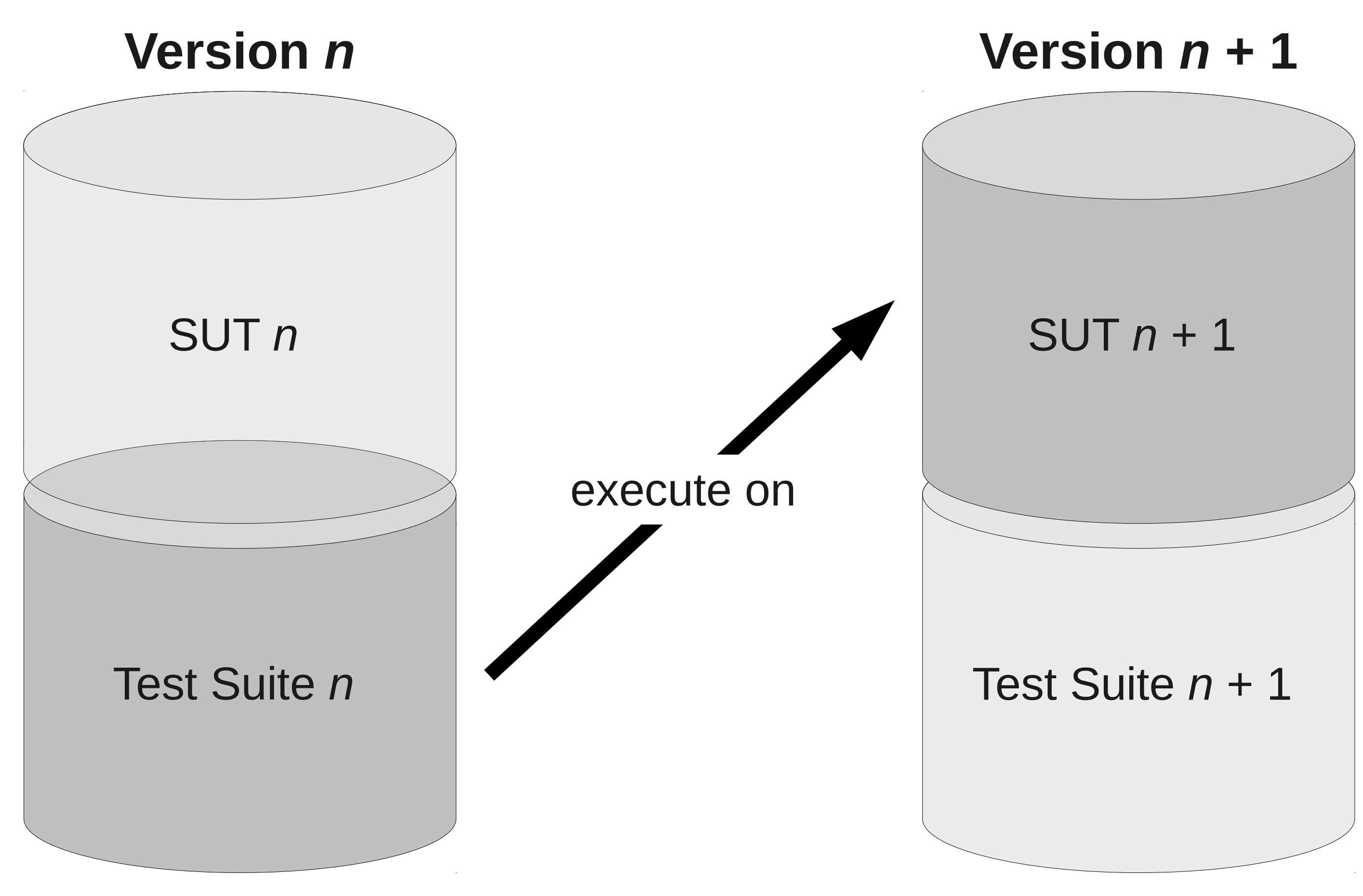
Evaluation

Q1: How many failures can ideal **literal replacement** repair?

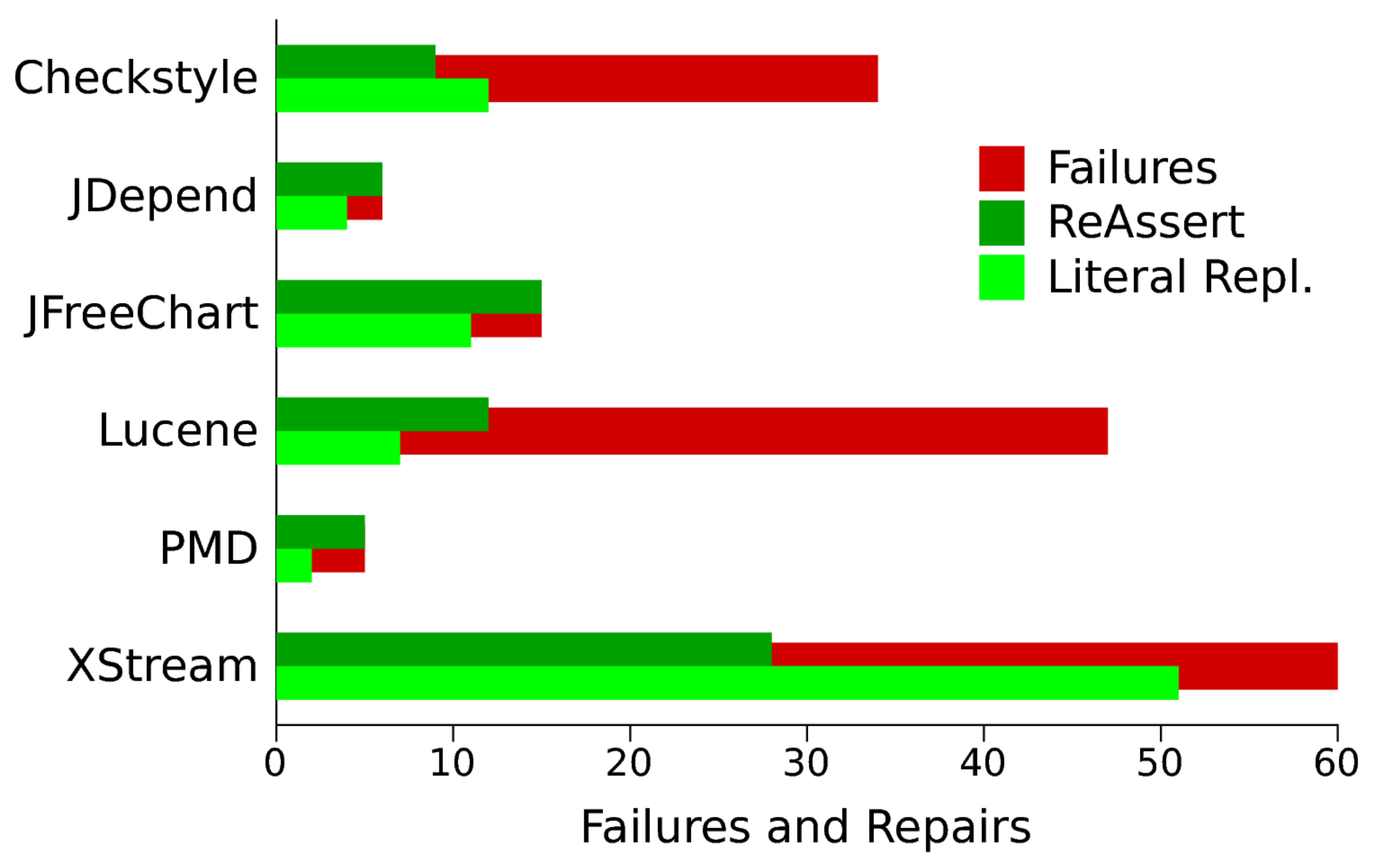
Q2: How do ReAssert and literal replacement **compare**?

Q3: Can **symbolic execution** discover literals?

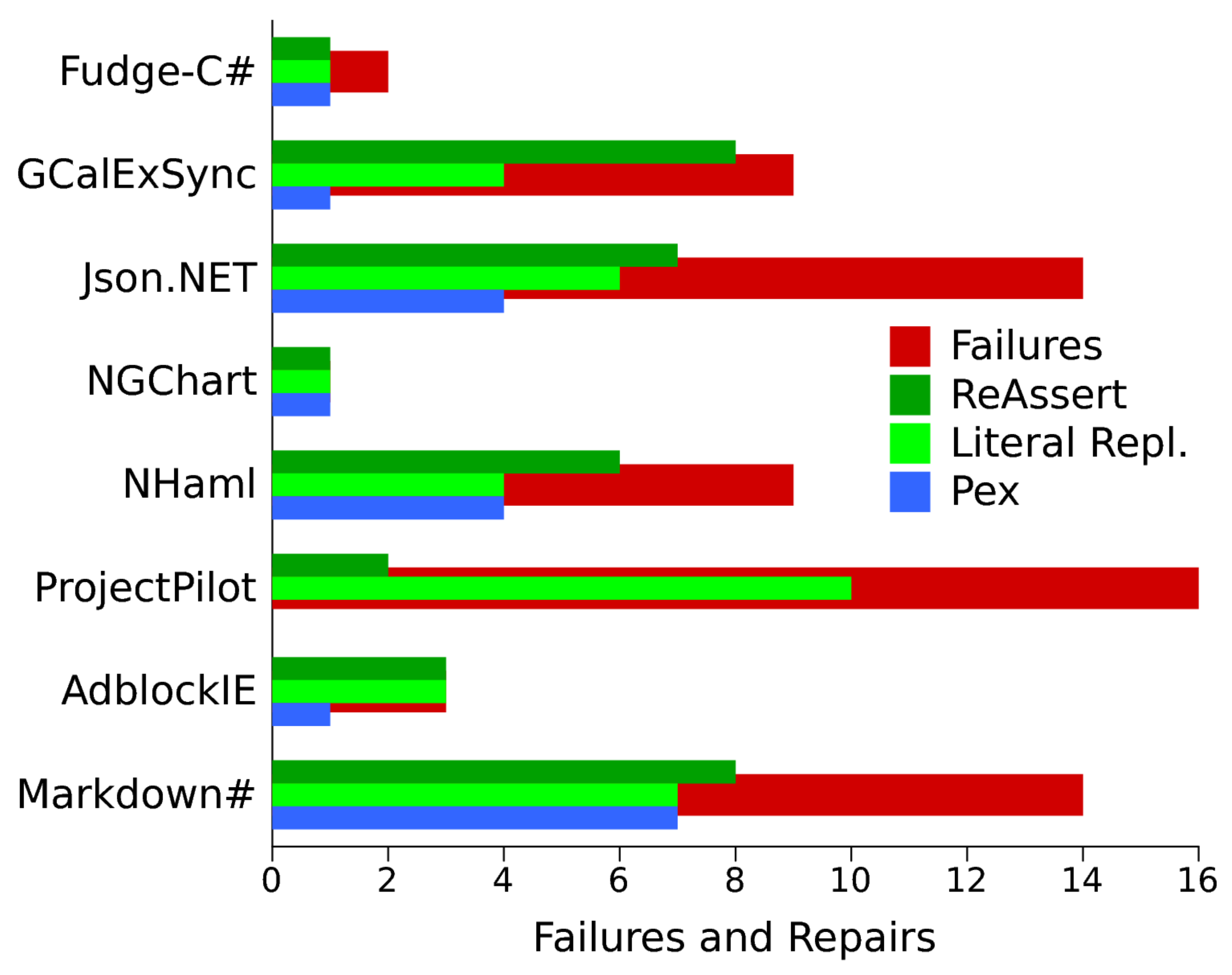
Failures in Open-Source Software



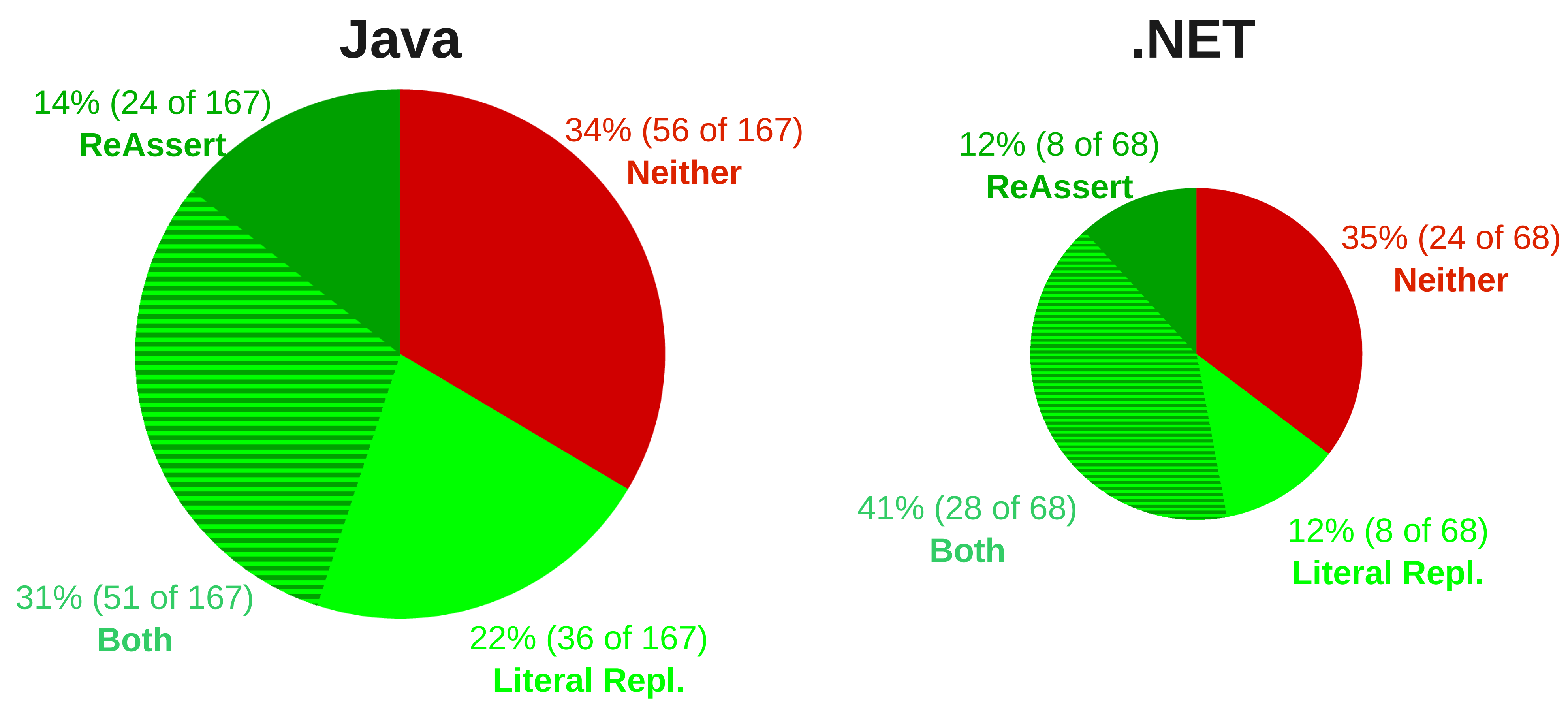
Open Source Software - Java



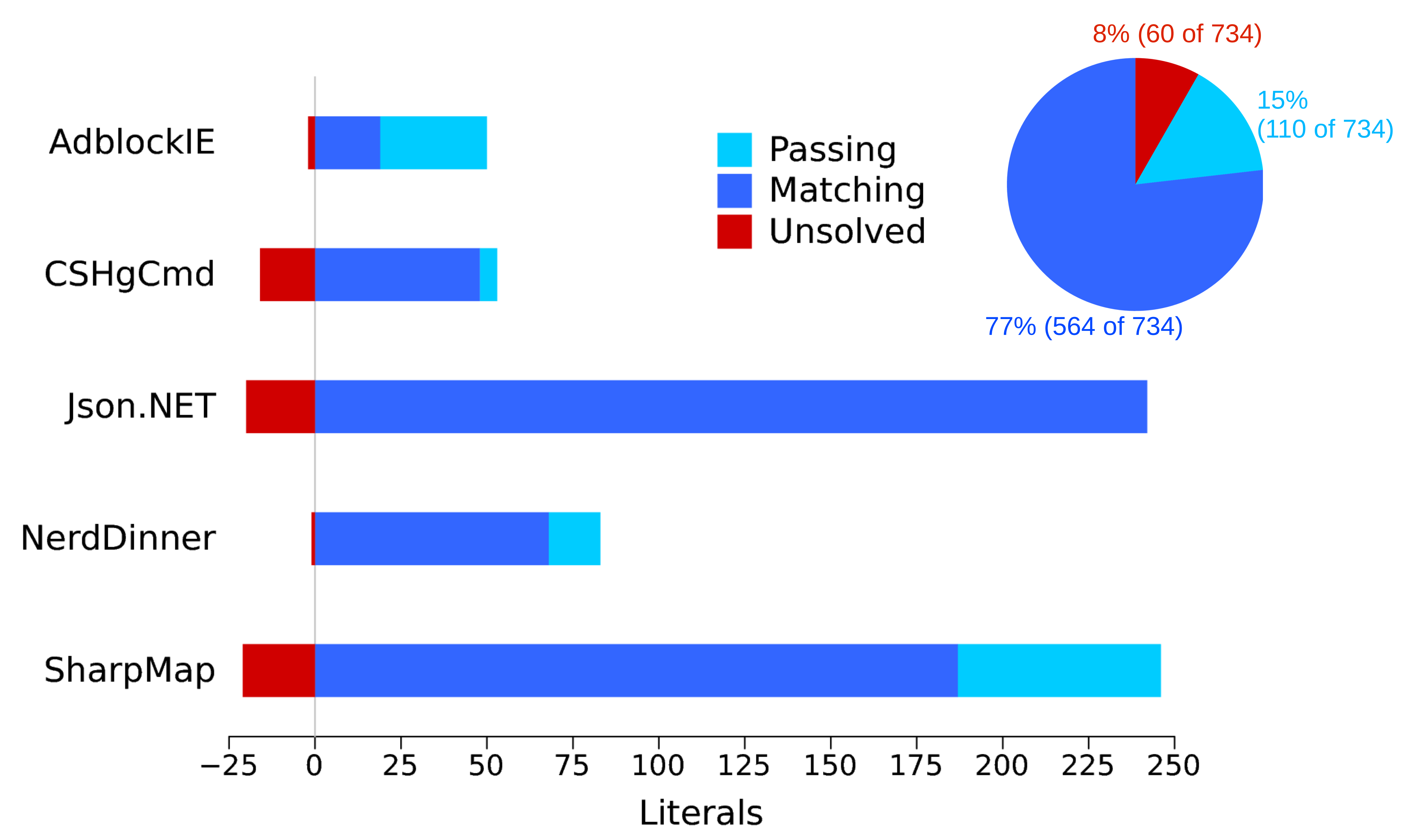
Open Source Software - .NET



ReAssert vs. Literal Replacement



Recreate Literals



Results

Q1: How many failures can ideal **literal replacement** repair?

About half

Q2: How do ReAssert and literal replacement **compare**?

12% to 22% improvement when combined

Q3: Can **symbolic execution** discover literals?

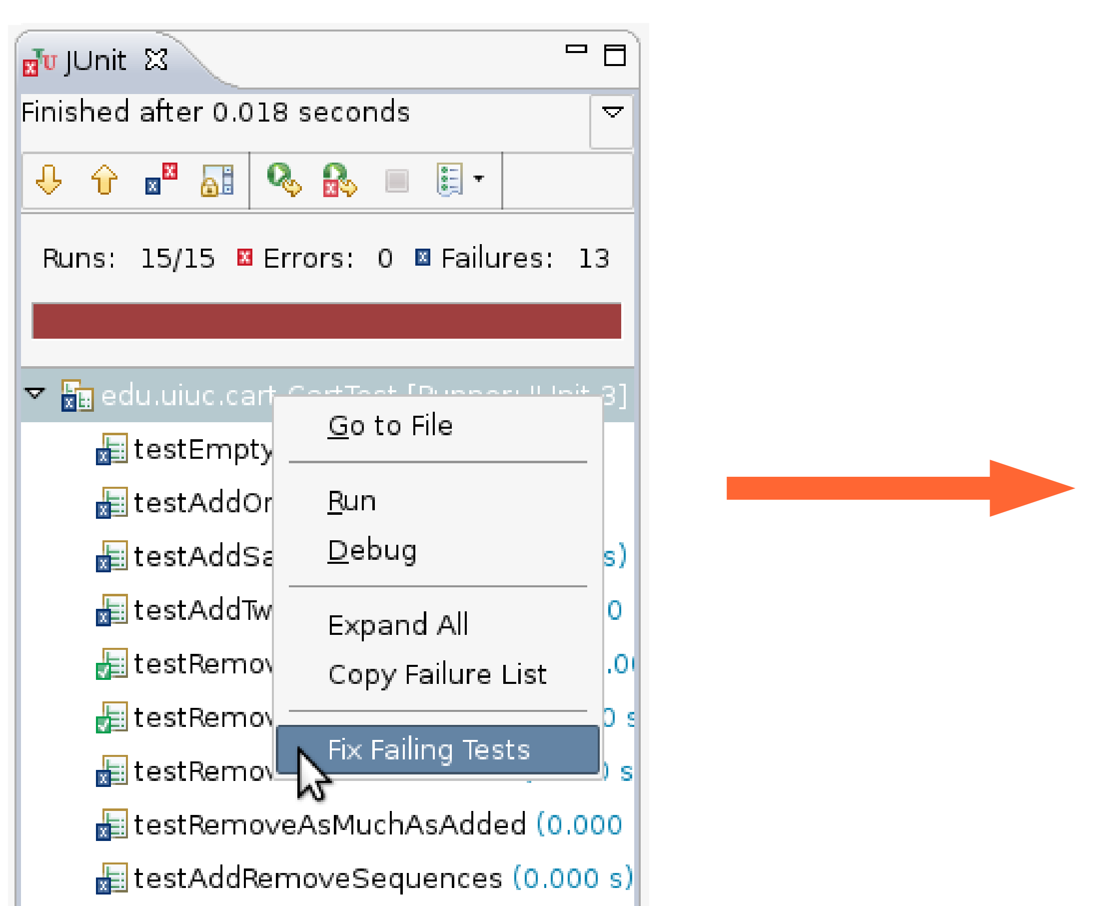
Yes: 52% to 92% of literals

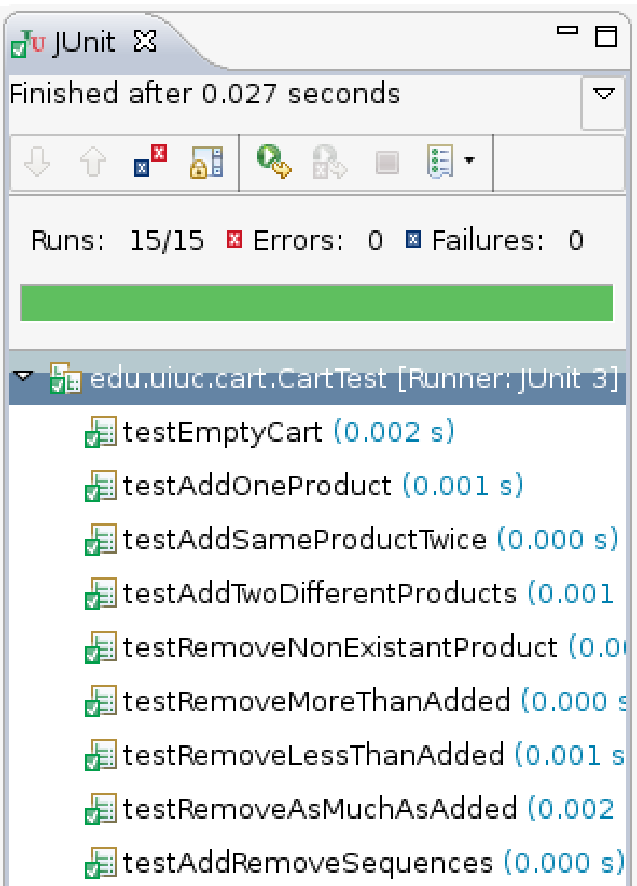
Thanks

- Nikolai Tillmann for help with Pex
- Milos Gligoric, Vilas Jagannath, and Viktor Kuncak for valuable comments
- Anonymous colleagues for potential evolutions
- Anonymous reviewers

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

ReAssert





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