

ReAssert: Suggesting Repairs for Broken Unit Tests

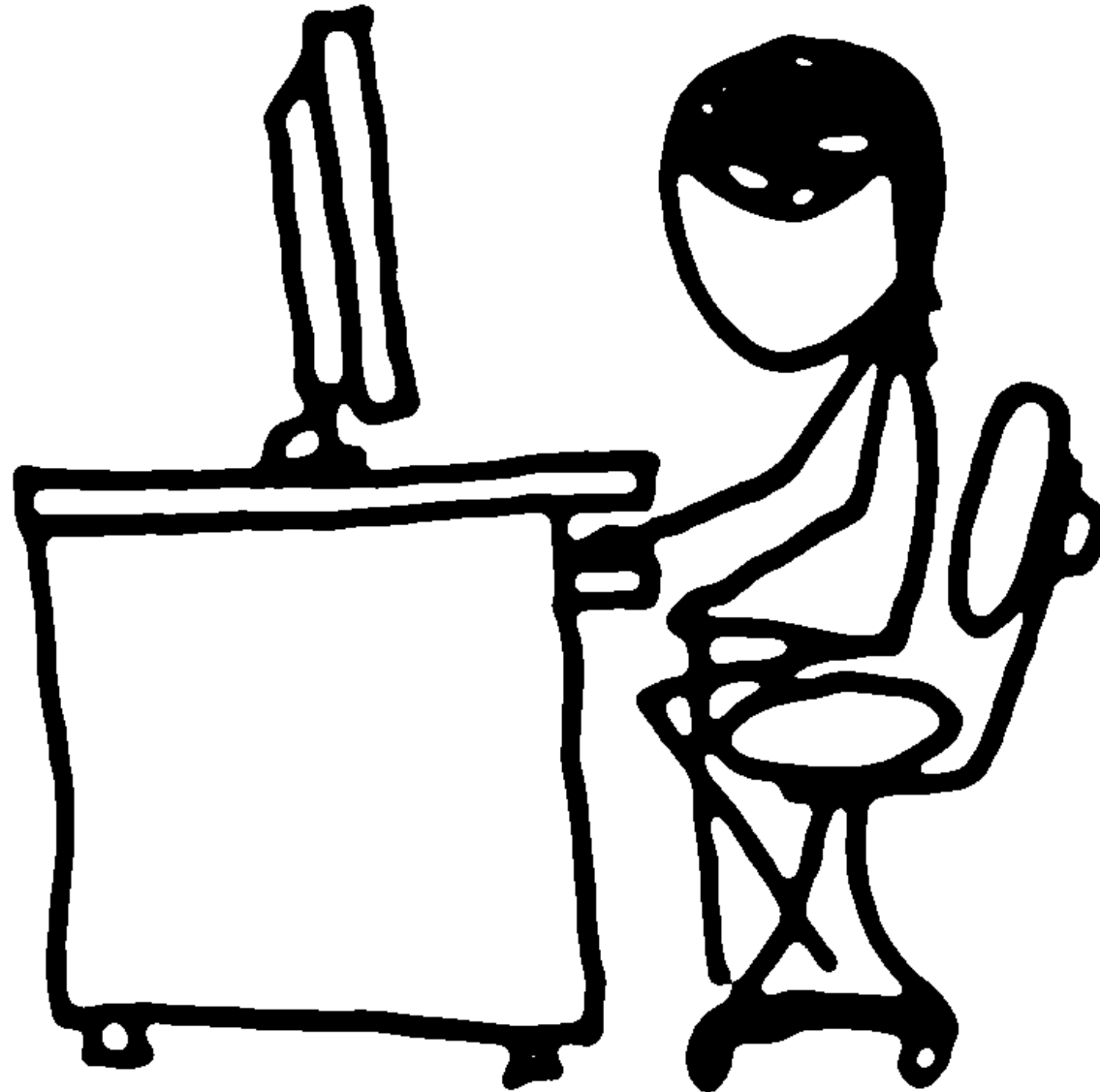
Brett Daniel
Vilas Jagannath
Danny Dig
Darko Marinov



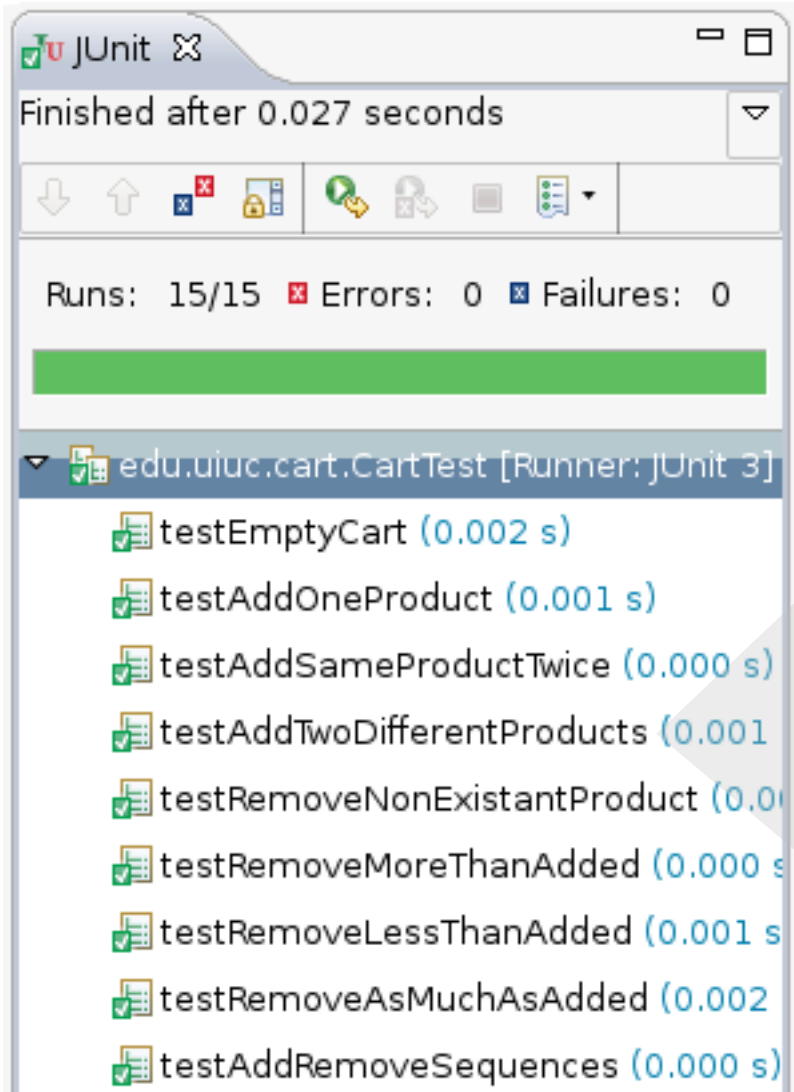
I L L I N O I S

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

This is Alice



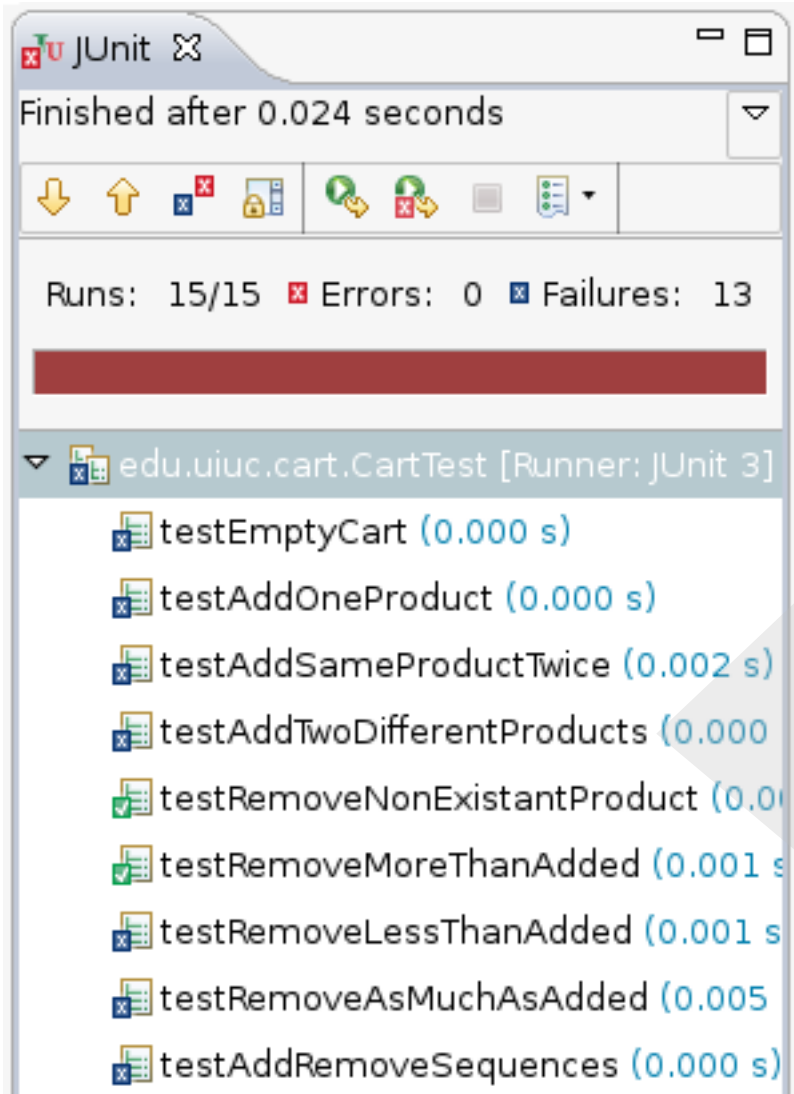
Her unit tests pass



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

```
public void testAddTwoDifferentProducts() {  
    Cart cart = ...  
    assertEquals(3.0, cart.getTotalPrice());  
    assertEquals(  
        "Discount: -$1.00, Total: $3.00",  
        cart.getPrintedBill());  
}
```

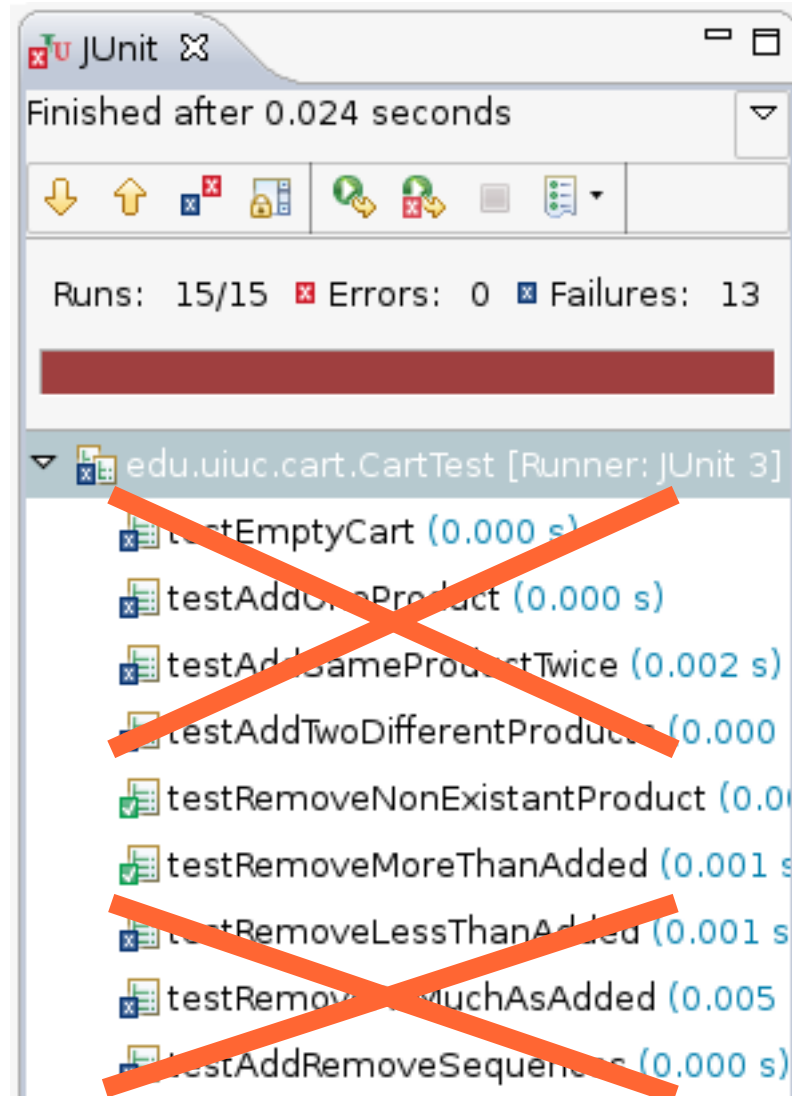
But requirements change



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

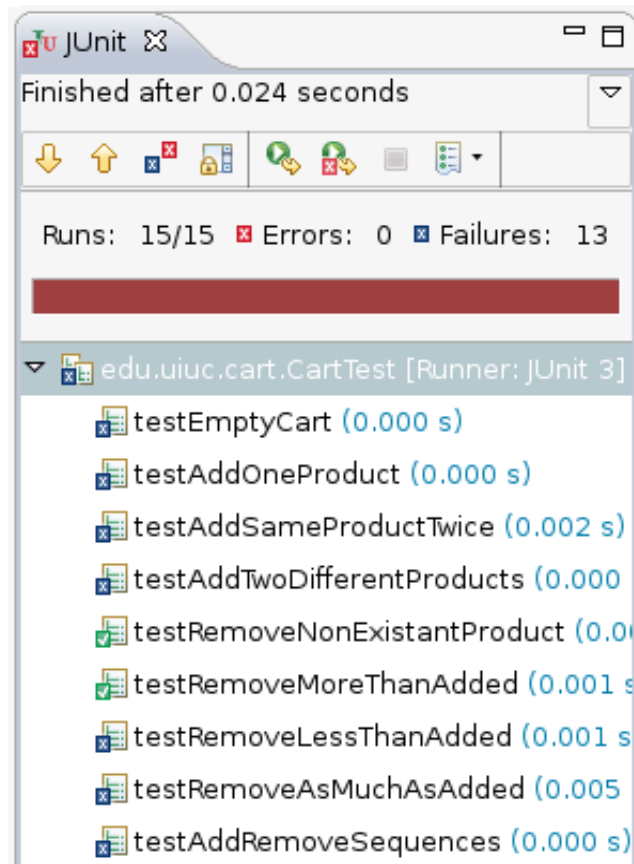
```
public void testAddTwoDifferentProducts() {  
    Cart cart = ...  
    assertEquals(3.0, cart.getTotalPrice());  
    assertEquals(  
        "Discount: -$1.00, Total: $3.00",  
        cart.getPrintedBill());  
}
```

She can delete broken tests

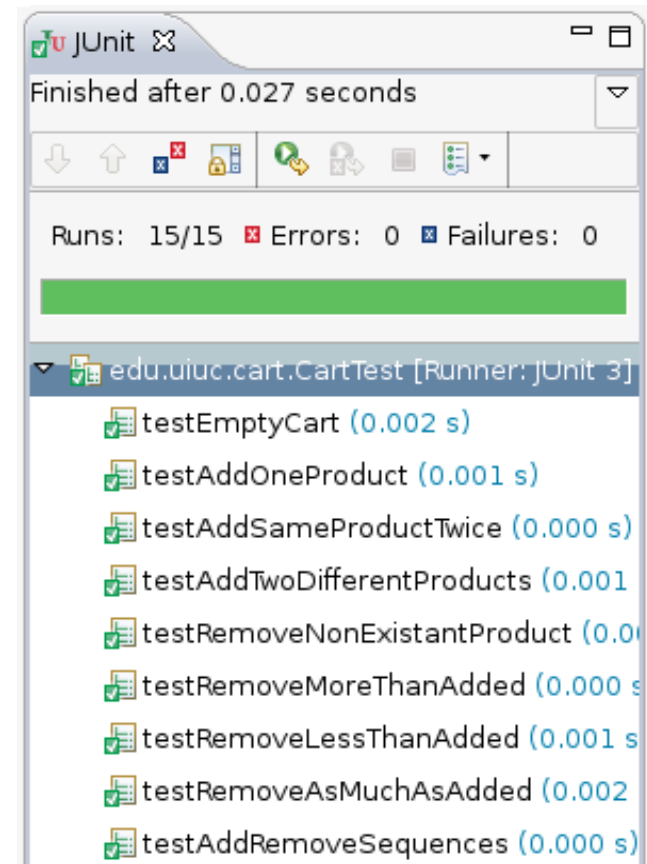


But that reduces the quality of the test suite.

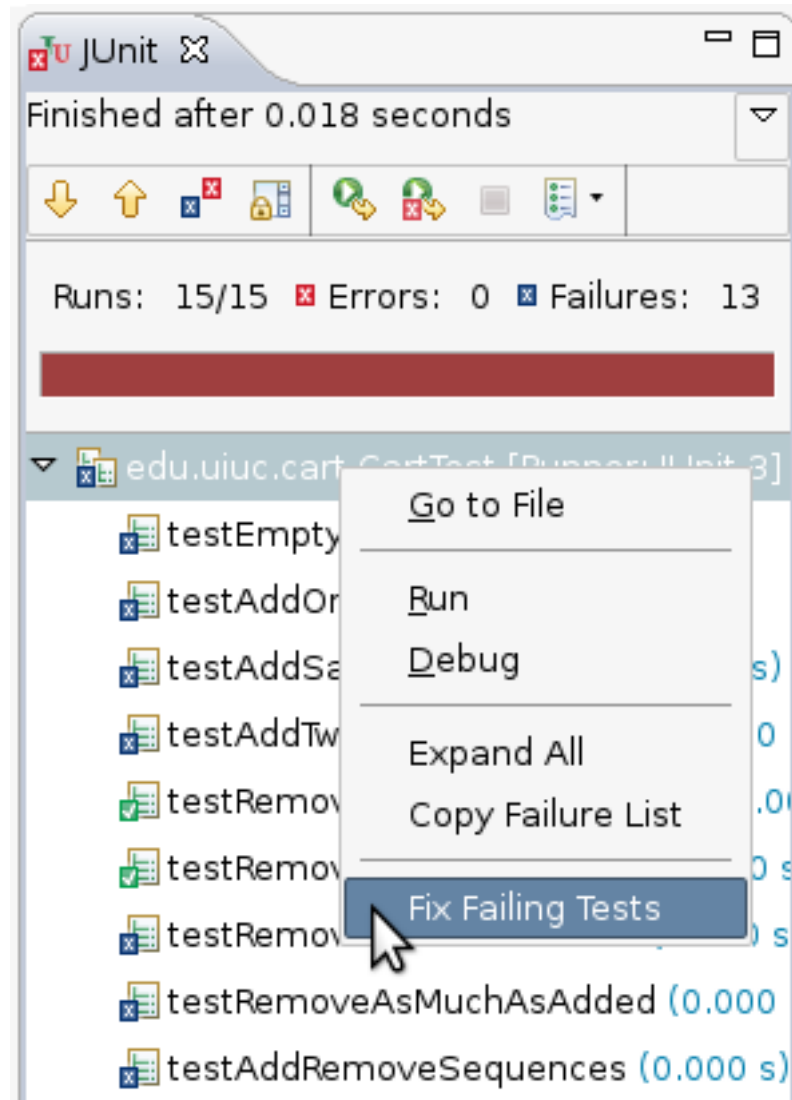
Repairing tests is preferable



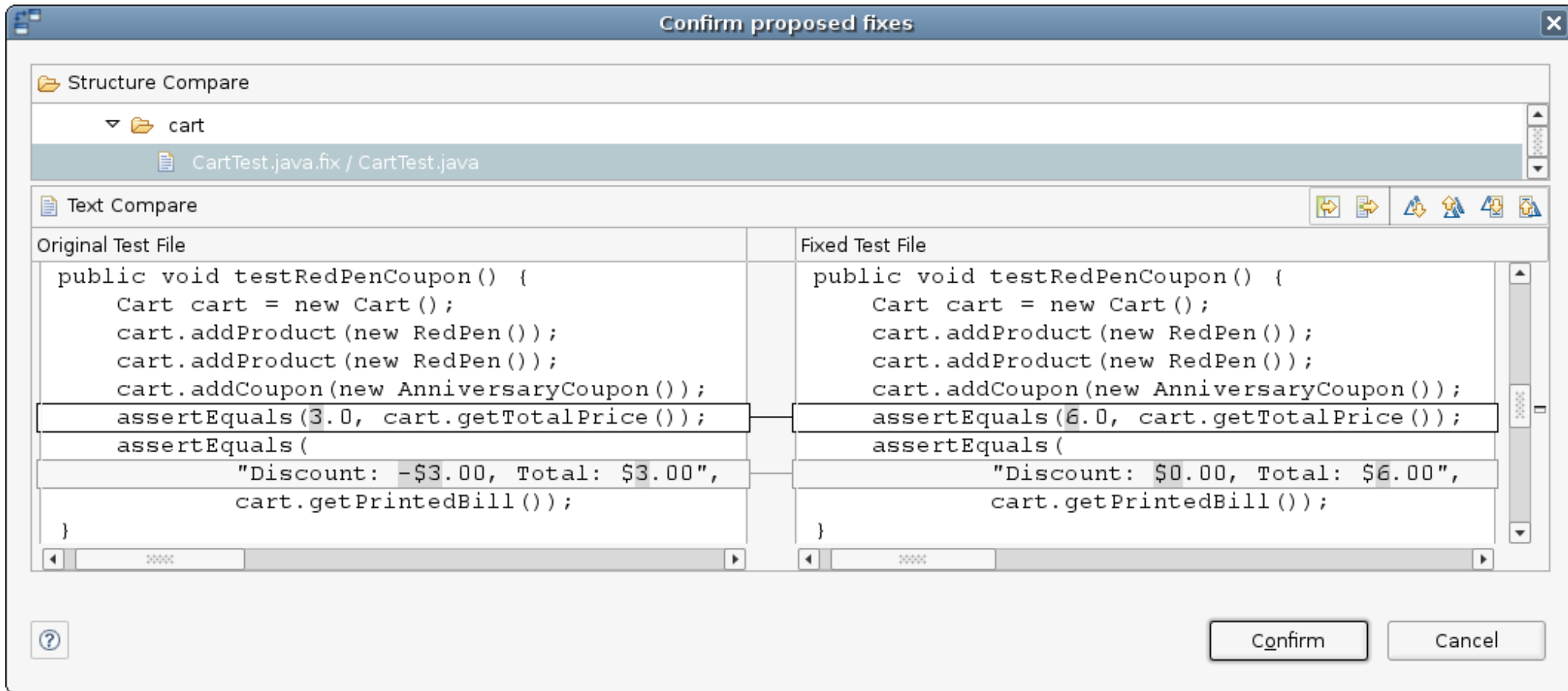
But that requires
a lot of time
and effort



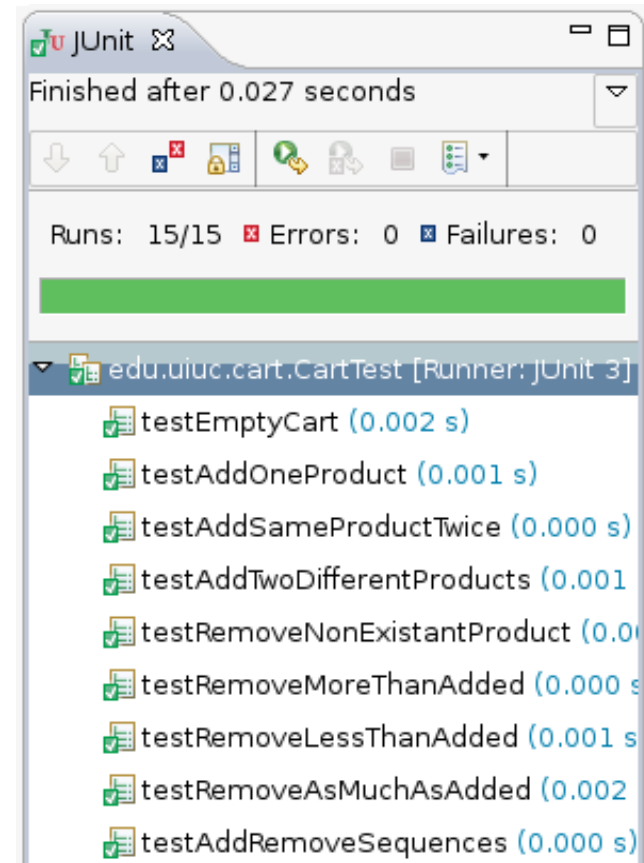
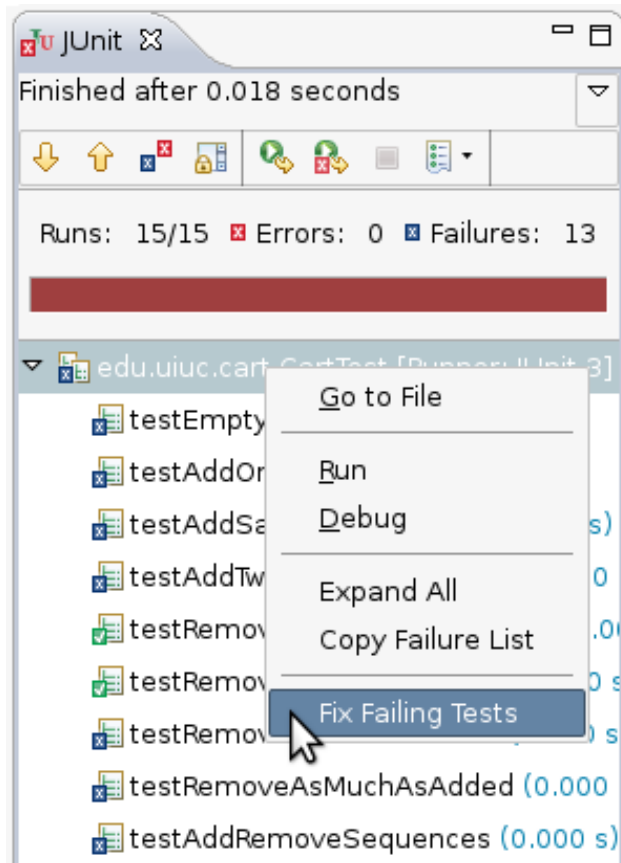
ReAssert suggests repairs



Alice decides whether to apply



ReAssert reduces effort



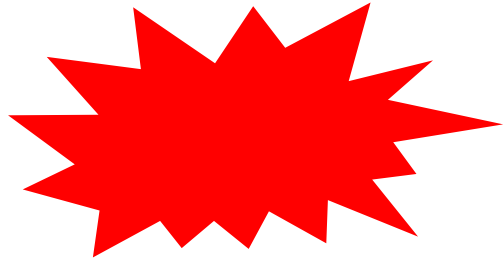
What is a Good Repair?

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



 `assertTrue(true);`

Repair Criteria



Good Repair

Make tests **pass**

Make **minimal changes** to test code (not SUT)

Require developer **approval**

Produce **understandable** test code

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

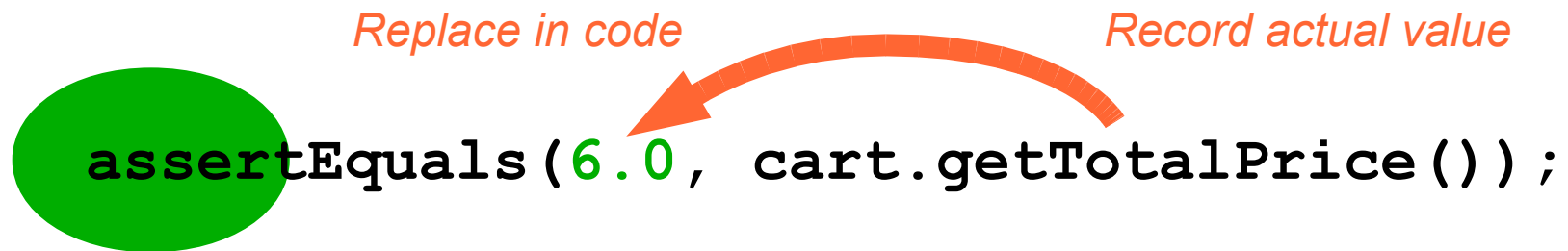


```
assertEquals(3.0, cart.getTotalPrice());
```

Strategy: Replace Literal


Replace in code *Record actual value*

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



Failure in Helper Method

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

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

Strategy: Trace Declaration-Use Path

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

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

Replace in code

Trace declaration-use path

Record actual value

Object (In)Equality Failure

```
Product expected = ...
```


```
Product actual = ...
```

```
assertEquals(expected, actual);
```

Strategy: Expand Accessors

```
Product expected = ...  
Product actual = ...  
{  
    assertEquals(  
    assertEquals(  
}  
}
```

Expand accessors



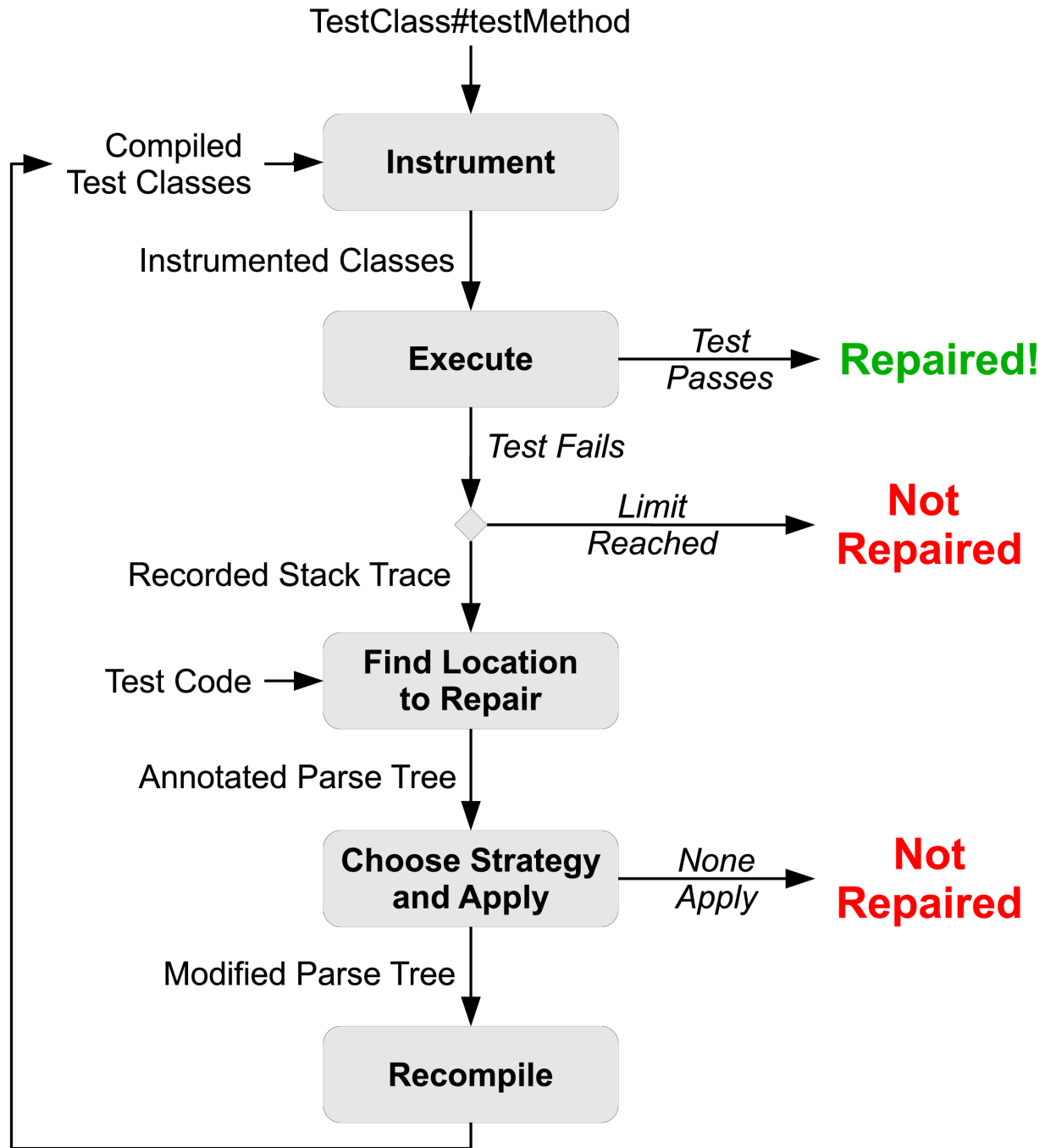
Strategy: Expand Accessors

```
Product expected = ...  
Product actual = ...
```

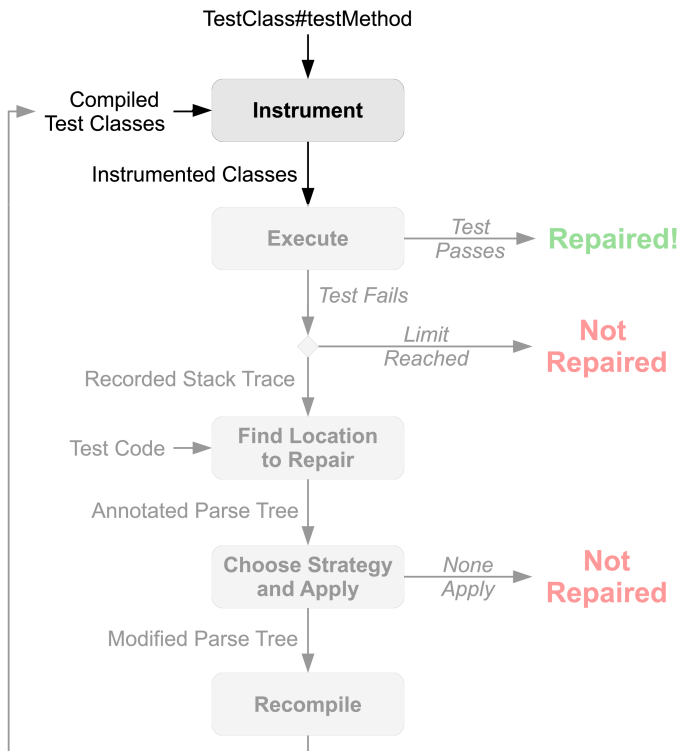
```
{  
  assertEquals(expected.getPrice(), actual.getPrice());  
  assertEquals("Red pen", actual.getDescription());  
}
```

*Expected and actual
accessors equal*

Actual accessor differs



Instrument

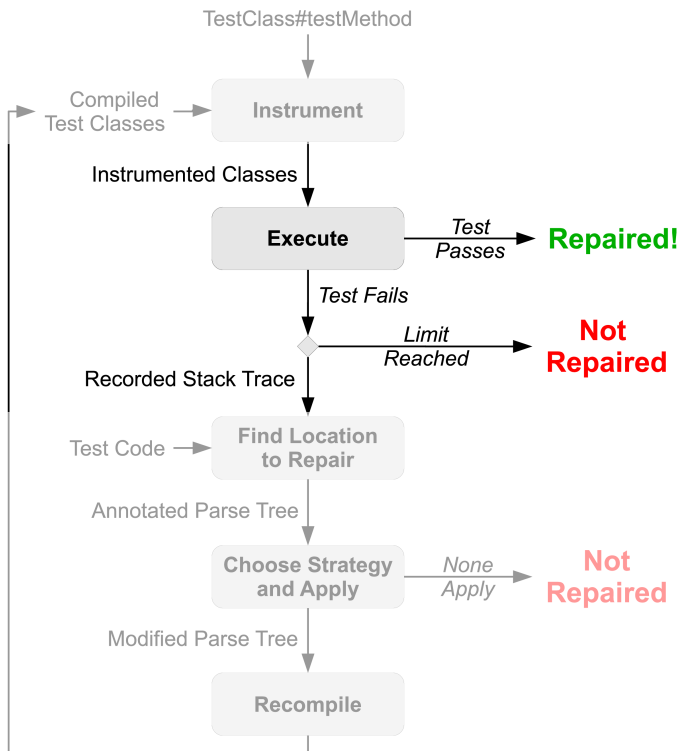


```
public static void assertEquals (  
    Object expected,  
    Object actual) {  
    try {  
        // ...assert expected.equals(actual)  
    }  
    catch (Error e) {  
        throw new RecordedAssertFailure(  
            e, expected, actual);  
    }  
}
```

If assertion fails...

...then record values
that caused failure

Execute

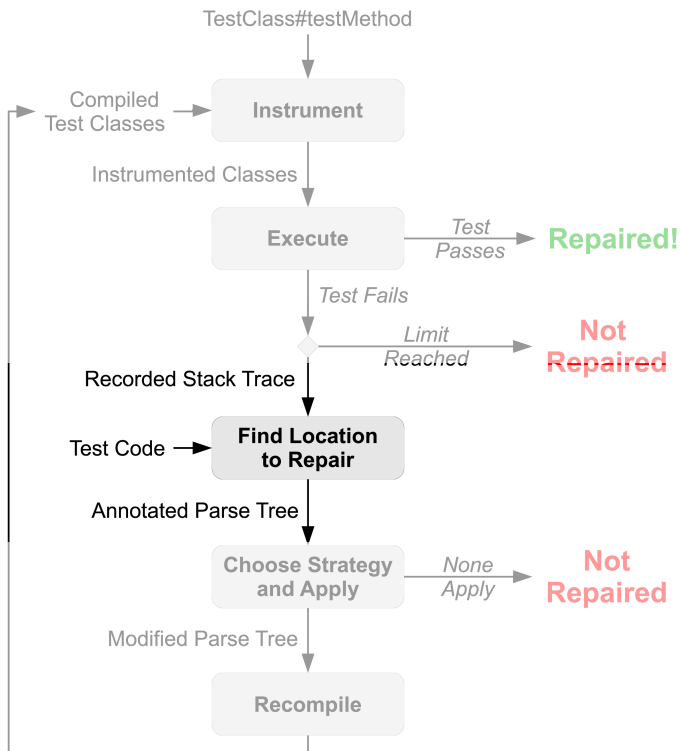


assertEquals(3.0, cart.getTotalPrice());

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

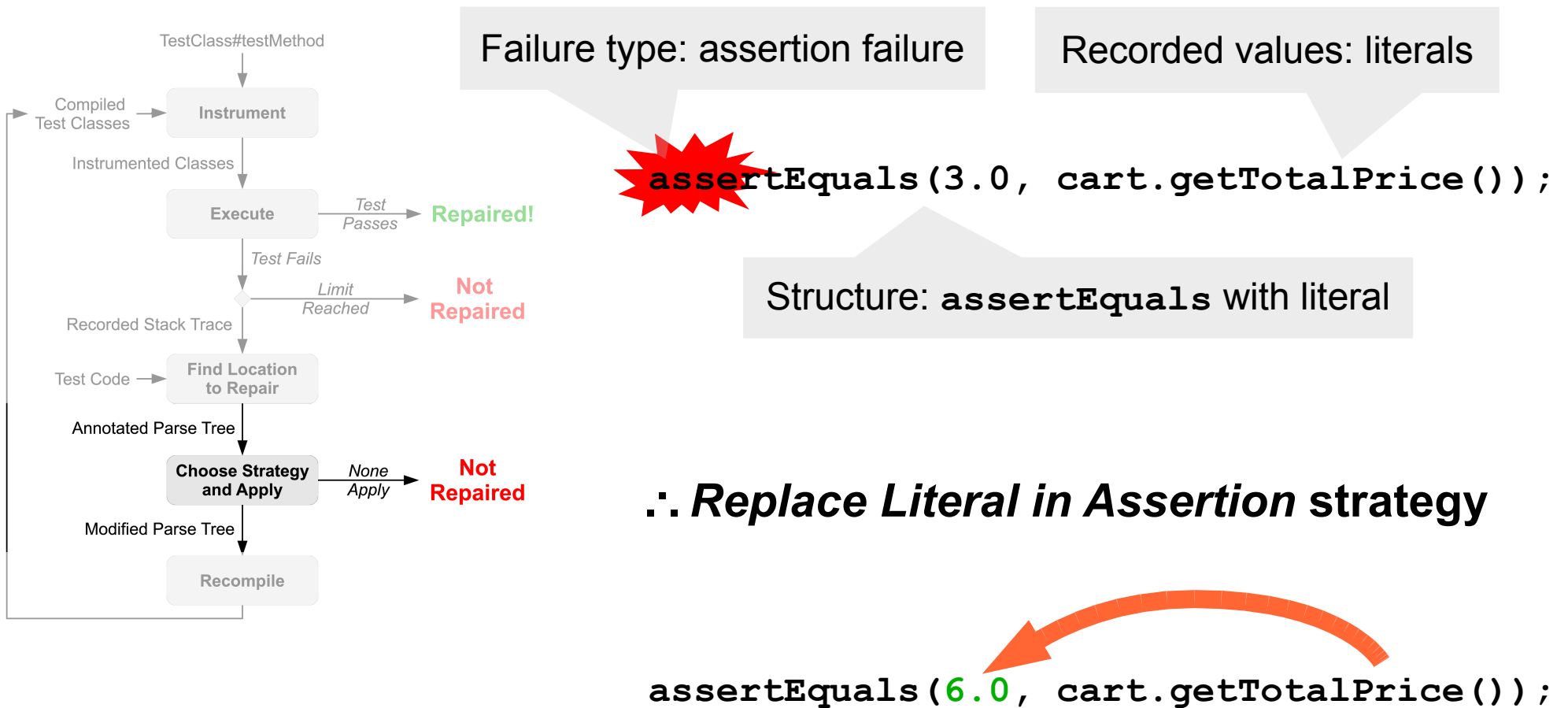
```
edu.illinois.reassert.RecordedAssertFailure:  
org.junit.AssertFailedError:  
expected:<3.0> but was:<6.0>  
    at org.junit.Assert.assertEquals(Assert.java:116)  
    at CartTest.testRedPenCoupon(CartTest.java:6)  
    ...
```

Find Repair Location

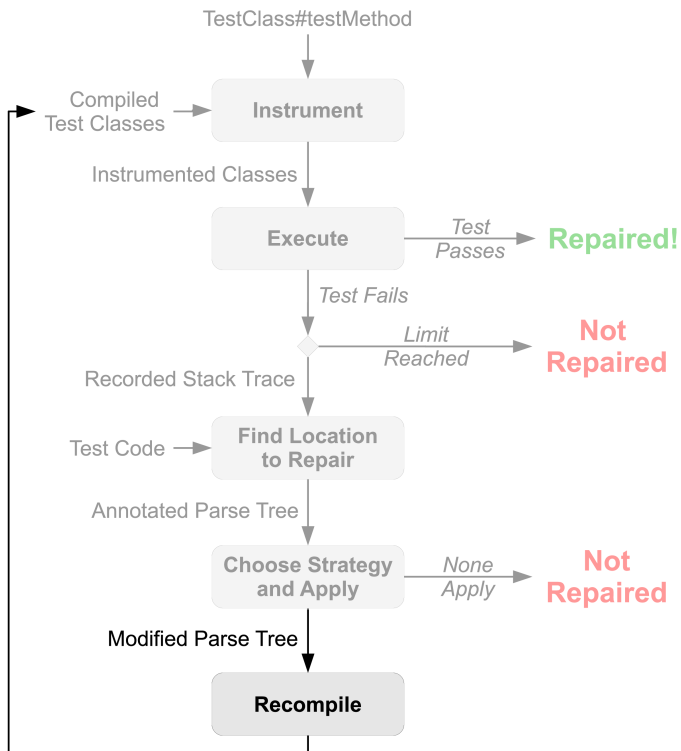


```
edu.illinois.reassert.RecordedAssertFailure:  
org.junit.AssertFailedError:  
expected:<3.0> but was:<6.0>  
    at org.junit.Assert.assertEquals(Assert.java:116)  
    at CartTest.testRedPenCoupon(CartTest.java:6)  
    ...
```

Choose Strategy and Apply



Recompile and Repeat



```
assertEquals(6.0, cart.getTotalPrice());  
assertEquals(  
    "Discount: -$1.00, Total: $3.00",  
    cart.getPrintedBill());
```

Evaluation

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

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

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

Evaluation

Repairs?

Useful?

Regressions?

Case Studies



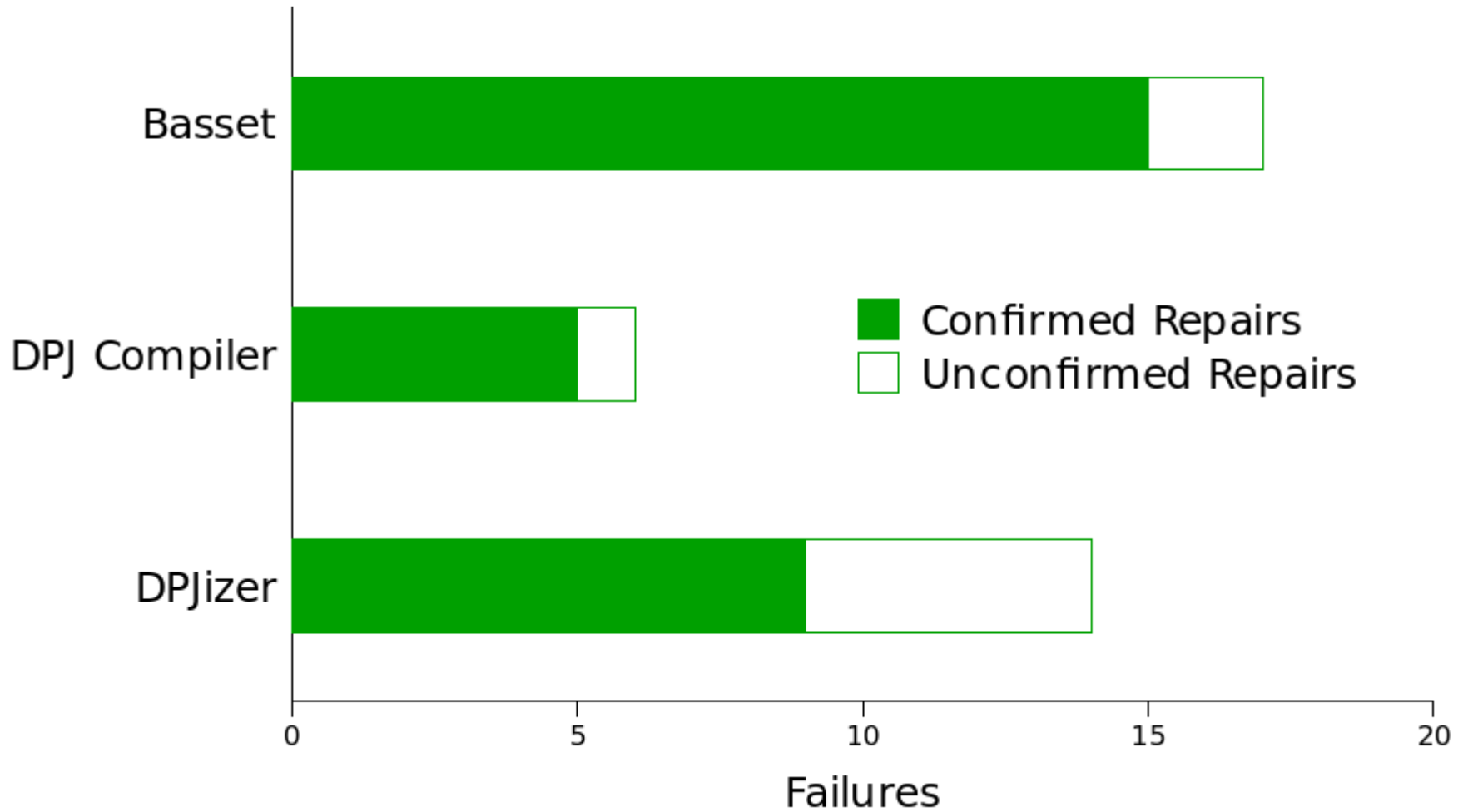
Controlled
User Study



Failures in
Open-Source
Software

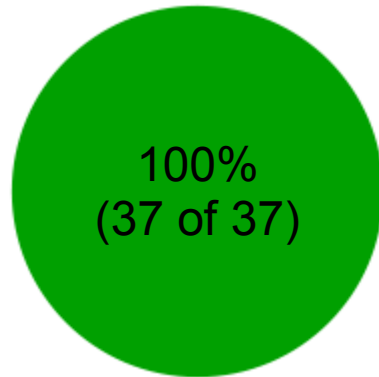


Case Studies

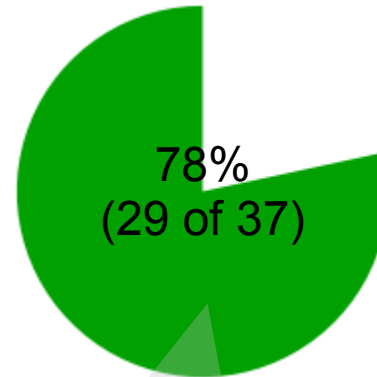


Case Studies

Repairs?

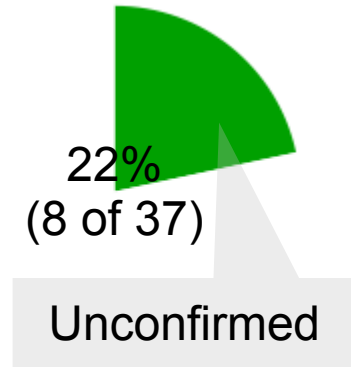


Useful?



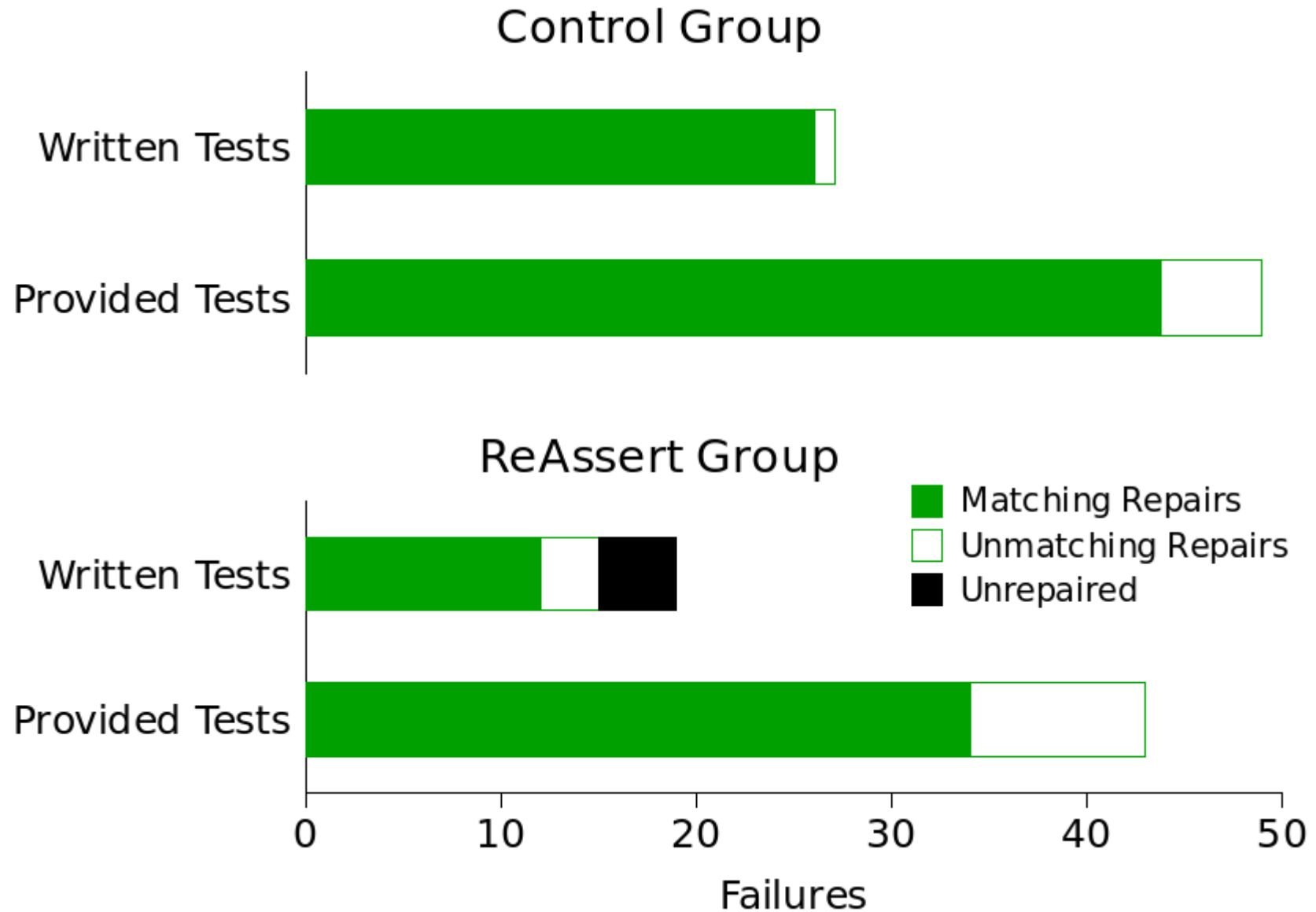
Confirmed by user

Regressions?



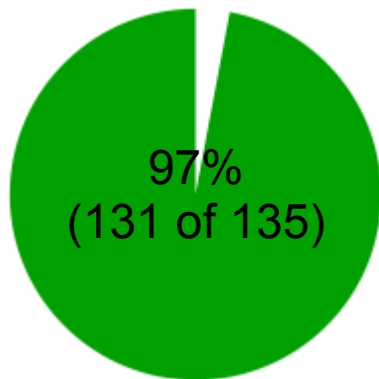
Unconfirmed

Controlled User Study

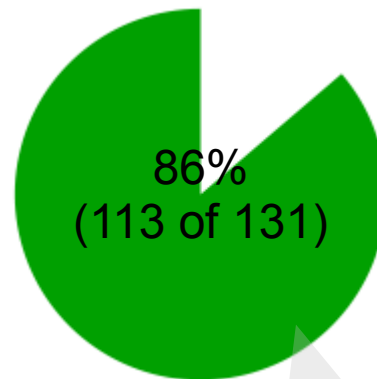


Controlled User Study

Repairs?

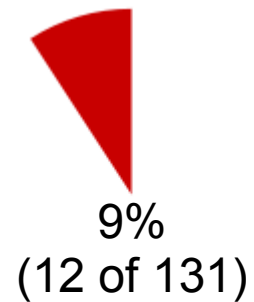


Useful?



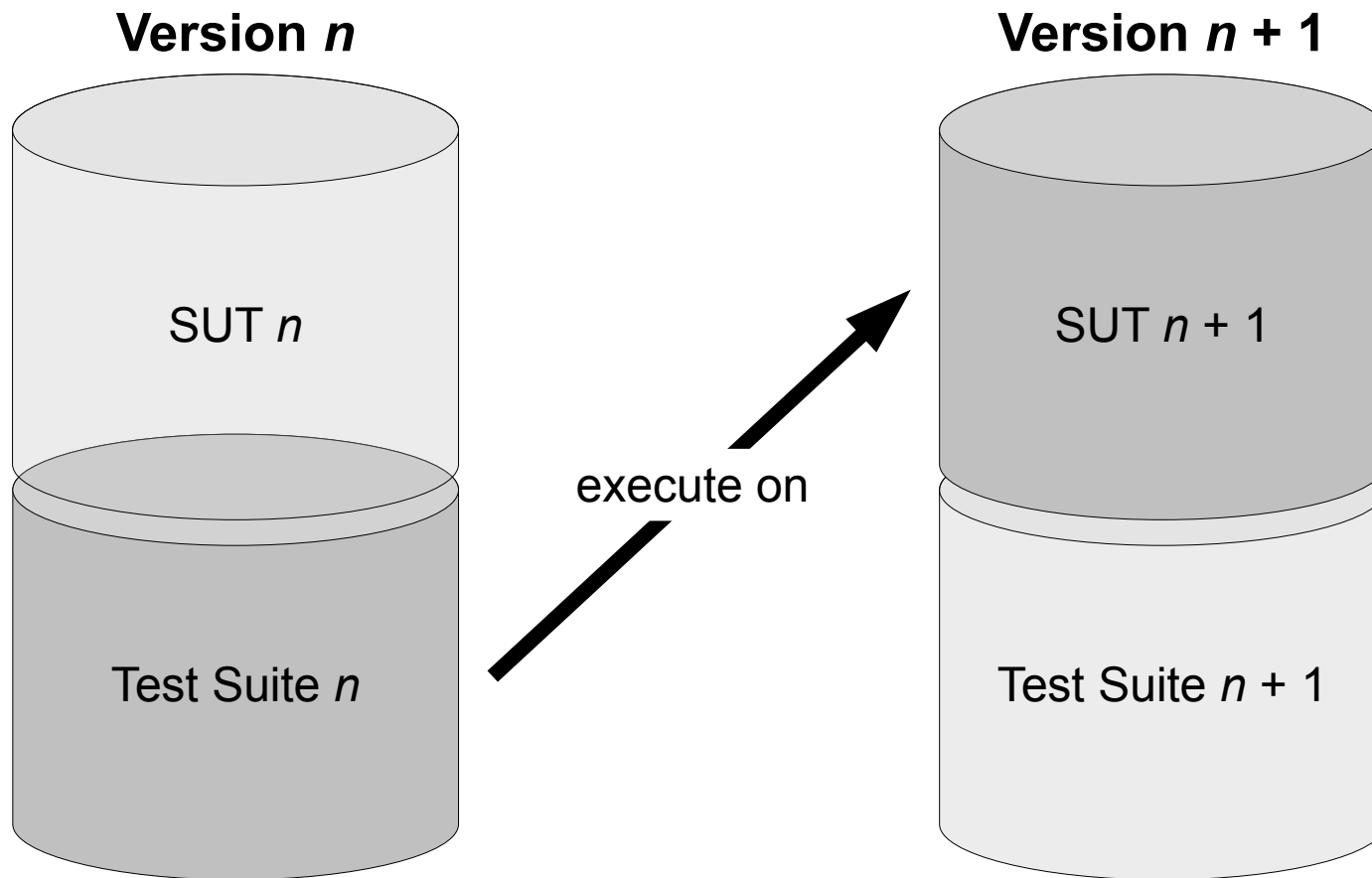
Matching repairs

Regressions?

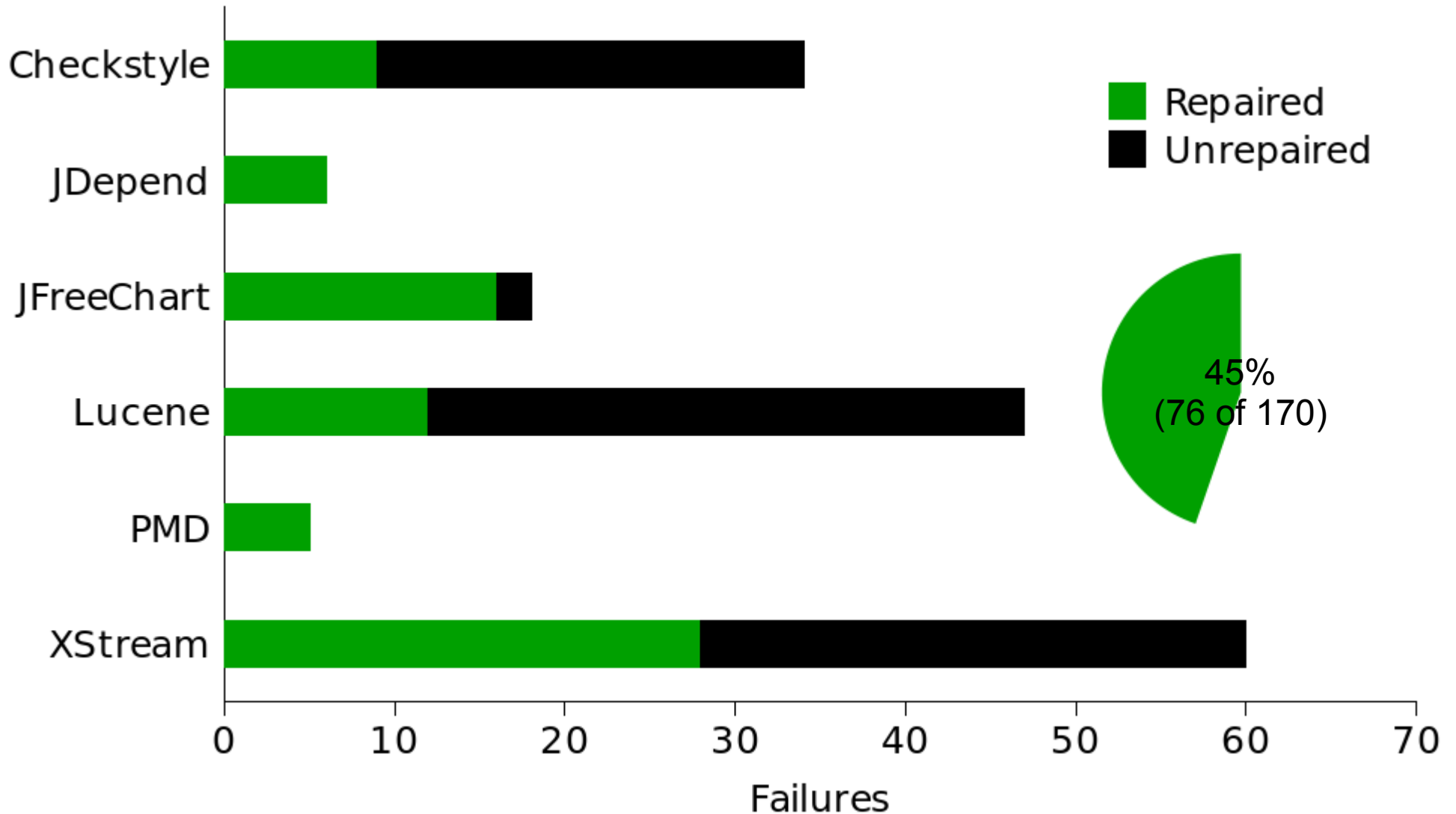


vs. 8 introduced by the control group

Failures in Open-Source Software



Failures in Open-Source Software




Unreparable Failures


- Nondeterminism

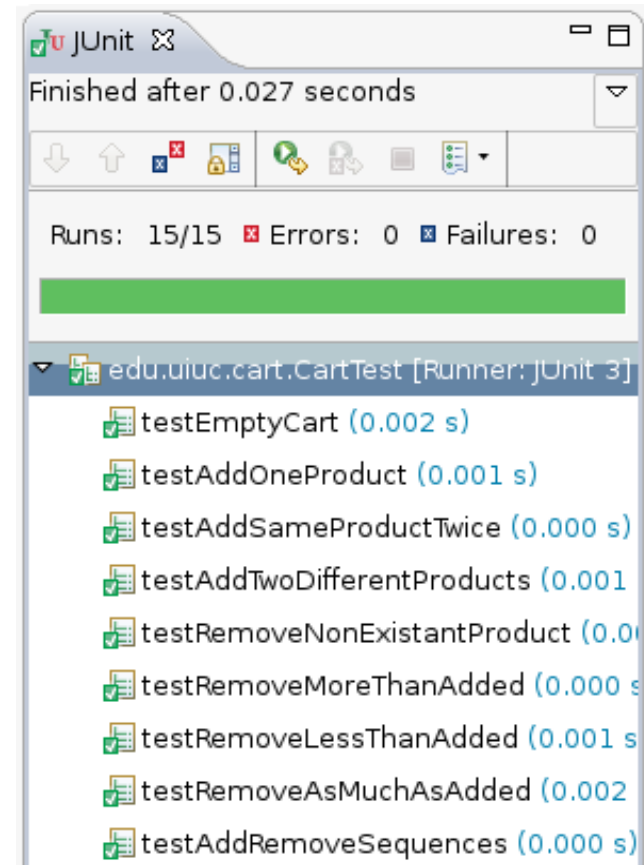
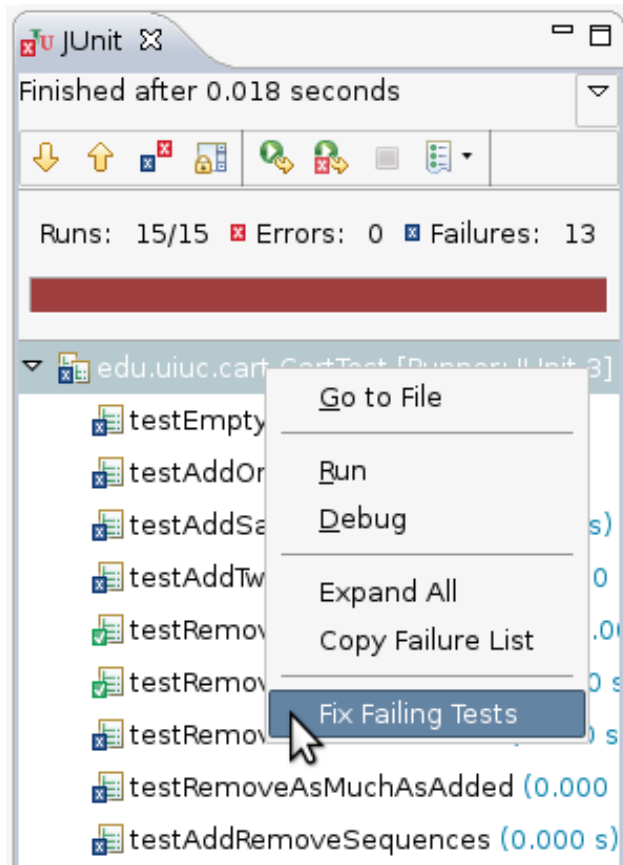
```
 assertEquals(..., cart.getPurchaseDate());
```

- Multiple contexts

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

- No applicable strategies

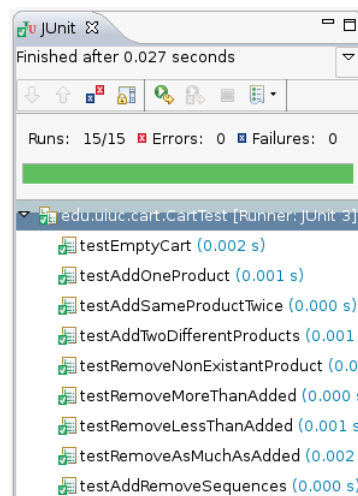
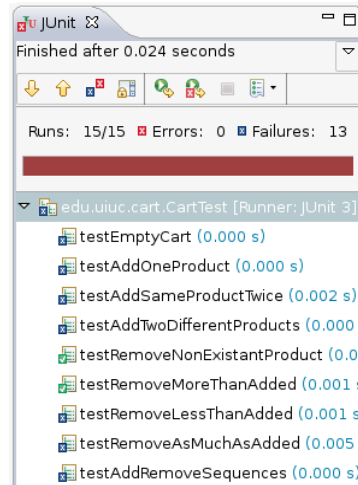
```
if (...) {  
    expected = 3.0;  
}  
 assertEquals(expected, cart.getTotalPrice());
```



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

Test-Driven Development

Test Repair



Make tests **fail**

...by changing tests

...by changing SUT

Make tests **pass**

...by changing SUT

...by changing tests