ReAssert: Suggesting Repairs for Broken Unit Tests

Brett Daniel
Vilas Jagannath
Danny Dig
Darko Marinov
This is Alice
Her unit tests pass

```java
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

```java
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?

```java
assert Equals(3.0, cart.getTotalPrice());
```
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

```java
assertEquals(3.0, cart.getTotalPrice());
```
Strategy: Replace Literal

```java
assertEquals(6.0, cart.getTotalPrice());
```
void testAddTwoDifferentProducts() { 
    Cart cart = ... 
    ... 
    checkCart(cart, 3.0, ...);
}

void checkCart( 
    Cart cart, double total, ...) {
    assertEquals(total, cart.getTotalPrice());
    ... 
}
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);
Product expected = ...  
Product actual = ...  
{
  assertEquals(                   , actual.getPrice());
  assertEquals(         , actual.getDescription());
}

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
TestClass#testMethod

Compiled Test Classes → Instrument → Instrumented Classes

Execute → Test Passes → Repaired!

Test Fails → Recorded Stack Trace

Find Location to Repair → Annotated Parse Tree

Choose Strategy and Apply → Modified Parse Tree

Recompile → None Apply → Not Repaired

Not Repaired
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

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

```java
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

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

Structure: `assertEquals` with literal

`: Replace Literal in Assertion strategy`

```java
assertEquals(6.0, cart.getTotalPrice());
```
Recompile and Repeat

```java
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

<table>
<thead>
<tr>
<th></th>
<th>Repairs?</th>
<th>Useful?</th>
<th>Regressions?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case Studies</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Controlled User Study</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Failures in Open-Source Software</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Case Studies

- Repairs?
  - 100% (37 of 37) Confirmed by user

- Useful?
  - 78% (29 of 37)

- Regressions?
  - 22% (8 of 37) Unconfirmed
Controlled User Study

- Repairs?: 97% (131 of 135)
- Useful?: 86% (113 of 131)
- Regressions?: 9% (12 of 131)

Matching repairs vs. 8 introduced by the control group
Failures in Open-Source Software
Failures in Open-Source Software

- Checkstyle: 45% repaired (76 of 170)
- JDepend: 15% repaired
- JFreeChart: 20% repaired
- Lucene: 30% repaired
- PMD: 5% repaired
- XStream: 35% repaired (60 of 170)
Unrepairable Failures

- Nondeterminism

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

- Multiple contexts

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

- No applicable strategies

```java
if (...) {
    expected = 3.0;
}
assertEquals(expected, cart.getTotalPrice());
```
Test-Driven Development

Make tests fail
...by changing tests
...by changing SUT

Test Repair

Make tests pass
...by changing SUT
...by changing tests