ReAssert: Suggesting Repairs for Broken Unit Tests

Brett Daniel
Danny Dig
Vilas Jagannath
Darko Marinov

Tihomir Gvero

CS527 Topics in Software Engineering
August 26, 2010
Passing Unit Tests

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

testAddTwoDifferentProducts()

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: Suggesting Repairs for Broken Unit Tests
Brett Daniel, Vilas Jagannath, Danny Dig, Darko Marinov
ASE 2009. Auckland, New Zealand
Confirm or Reject Suggestions

public void testRedPenCoupon() {
    Cart cart = new Cart();
    cart.addProduct(new RedPen);
    cart.addProduct(new RedPen);
    cart.addCoupon(new AnniversaryCoupon);
    
    assertEquals(3.0, cart.getTotalPrice());
    assertEquals("Discount: -$3.00, Total: $3.00",
                cart.getPrintedBill());
}

public void testRedPenCoupon() {
    Cart cart = new Cart();
    cart.addProduct(new RedPen);
    cart.addProduct(new RedPen);
    cart.addCoupon(new AnniversaryCoupon);
    
    assertEquals(6.0, cart.getTotalPrice());
    assertEquals("Discount: -$0.00, Total: $6.00",
                cart.getPrintedBill());
}
ReAssert Reduces Effort
What is a Good Repair?

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

*Bad 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

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

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

Replace in code

Record actual value
Temporary Variable

double expTotal = 3.0;

... 

assertEquals(expTotal, cart.getTotalPrice());
double expTotal = 6.0;
...
assertEquals(expTotal, cart.getTotalPrice());
Failure in Helper Method

```java
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);
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());
}

Expected and actual accessor differs

Actual accessor differs
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
assertEquals(3.0, cart.getTotalPrice());

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

duced.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());
```

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

::*Replace Literal in Assertion* strategy

```java
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

<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

Control Group

Written Tests

Provided Tests

ReAssert Group

Written Tests

Provided Tests

Failures

- Matching Repairs
- Unmatching Repairs
- Unrepaired
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

Version $n$

SUT $n$

Test Suite $n$

execute on

Version $n + 1$

SUT $n + 1$

Test Suite $n + 1$
Failures in Open-Source Software

- Checkstyle
- JDepend
- JFreeChart
- Lucene
- PMD
- XStream

45% (76 of 170)
Evaluating ReAssert

Q1: How many failures can ReAssert \text{repair}?

45\% in open source software

Q2: Are ReAssert's suggested repairs \text{useful}?

78\% to 86\% approved by users

Q3: Does ReAssert \text{reveal} or \text{hide} regressions?

Both, comparable to manual edits
Unrepairable Failures

• Nondeterminism

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

• Multiple contexts

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

• Multiple Expected Values
  ```java
  double expTotal;
  if (HAS_TAX) {
      expTotal = 3.15;
  }
  else {
      expTotal = 3.0;
  }
  assertEquals(expTotal, cart.getTotalPrice());
  ```

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

• Expected Object Comparison
  ```java
  Product expProduct = new Product("Red pen", 3.0);
  assertEquals(expProduct, cart.getItem(0));
  ```
double expTotal;
if (HAS_TAX) {
    expTotal = 3.15;
}
else {
    expTotal = 3.0;
}

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

// Question mark indicates where more code is needed
assertEquals(expTotal, cart.getTotalPrice());
```
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

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

Branches introduce path constraints

Solve constraints to execute alternate paths

Dynamic symbolic execution

Nondeterministic choice generator produces concrete values

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

```java
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

```java
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

```java
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

```java
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

![Bar chart showing failures and repairs for different Java projects: Checkstyle, JDepend, JFreeChart, Lucene, PMD, XStream. The chart uses colors to represent failures, ReAssert, and Literal Repl. with XStream showing the highest values.]
Open Source Software - .NET

Failures and Repairs
ReAssert vs. Literal Replacement

Java
- ReAssert: 14% (24 of 167)
- Neither: 34% (56 of 167)
- Both: 22% (36 of 167)
- Literal Repl.: 31% (51 of 167)

.NET
- ReAssert: 12% (8 of 68)
- Neither: 35% (24 of 68)
- Both: 41% (28 of 68)
- Literal Repl.: 12% (8 of 68)
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 colleague 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