ReAssert: Suggesting Repairs for Broken Unit Tests

Brett Daniel
Danny Dig
Vilas Jagannath
Darko Marinov

Tihomir Gvero

IBM Research – Almaden. San Jose, CA. August 10, 2010
Google. Mountain View, CA. August 11, 2010
Passing Unit Tests

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

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

Bad Repair!

```java
assertTrue(true);
```
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());
```
assertEquals(6.0, cart.getTotalPrice());
Temporary Variable

double expTotal = 3.0;

...  

assertEquals(expTotal, cart.getTotalPrice());
double expTotal = 6.0;
...

assertEquals(expTotal, 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);
Expand Accessors

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

Expected and actual accessors equal
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
Execute

execute

```java
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)
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(\texttt{CartTest.java:6})
...
Choose Strategy and Apply

Failure type: assertion failure
Recorded values: literals

\[
\text{assertEquals}(3.0, \text{cart.getTotalPrice}());
\]

Structure: assertEqual with literal

- Replace Literal in Assertion strategy

\[
\text{assertEquals}(6.0, \text{cart.getTotalPrice}());
\]
Recompile and Repeat

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

```java
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>

Red checks indicate issues.
Case Studies

- Basset
- DPJ Compiler
- DPJizer

Legend:
- Green: Confirmed Repairs
- White: Unconfirmed Repairs
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

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 $n$

Version $n + 1$

SUT $n + 1$

Test Suite $n + 1$

execute on
Failures in Open-Source Software

- **Checkstyle**: 15/30 (50% repaired, 50% unrepaired)
- **JDepend**: 5/10 (50% repaired, 50% unrepaired)
- **JFreeChart**: 12/20 (60% repaired, 40% unrepaired)
- **Lucene**: 30/60 (50% repaired, 50% unrepaired)
- **PMD**: 3/6 (50% repaired, 50% unrepaired)
- **XStream**: 20/40 (50% repaired, 50% unrepaired)

Overall, 45% of the failures were repaired (76 out of 170).
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

```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());
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());
<table>
<thead>
<tr>
<th>Insight</th>
<th>Many failures can be repaired by changing literal values in test code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem</td>
<td>ReAssert could not determine which literals needed to change and how</td>
</tr>
<tr>
<td>Hypothesis</td>
<td>Symbolic execution can discover literals that cause a test to pass</td>
</tr>
</tbody>
</table>

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

Branches introduce path constraints

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

Dynamic symbolic execution

Nondeterministic choice generator produces concrete values

Solve constraints to execute alternate paths

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

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?
ReAssert vs. Literal Replacement

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

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

AdblockIE

CSHgCmd

Json.NET

NerdDinner

SharpMap

77% (564 of 734)

8% (60 of 734)

15% (110 of 734)

-25 0 25 50 75 100 125 150 175 200 225 250

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