On Symbolic Test Repair

Presented by

Ritwika Ghosh
rghosh9@illinois.edu

November 6, 2015
Introduction

- Regression Testing: Important aspect of software development.
- What: Uncovering (possible) new bugs after changes to existing system.
- Why: Rather obvious.
- How: Rigorous development, and vigilant maintenance.
- Problem?
Regression:
"when you fix one bug, you introduce several newer bugs."

- Problem?
Regression:
"when you fix one bug, you introduce several newer bugs."

- Problem? Broken Tests.

Stefan Pölt, FRA IN/P
Outline

An Example
  Effect on tests
  On repair quality

Background
  ReAssert Strategies
  Symbolic Execution in Testing
  Challenges

Evaluation
  Experiments
  Results
  A few questions
Methods which change

```java
public class Example {
    ...
    public int returnsAnInteger() {...}
    public String returnsAString() {...}
}
```

Unit tests involving them

```java
public void testingThoseMethods() {
    Example example;
    assertEquals(3, example.returnsAnInteger());
    assertEquals("the integer is 3", example.returnsAString());
}
```
The test

```java
assertEquals(3, example.returnsAnInteger());
```

fails, because the method changed.
The test

    assertEquals(3, example.returnsAnInteger());

fails, because the method changed.

Delete?
An Example
Effect on Tests

- The test
  
  ```java
  assertEquals(3, example.returnsAnInteger());
  ```

  fails, because the method changed.
- Delete?
- Repairing tests: problem?
An Example
On repair quality

- ReAssert suggests repairs, confirm or reject, reduces effort.
- What isn’t a good repair?
An Example
On repair quality

- ReAssert suggests repairs, confirm or reject, reduces effort.
- What isn’t a good repair?
  ```java
  assertEquals(3, example.returnsAnInteger());
  assertTrue(true);
  ```
- What is?
  ```java
  ```
ReAssert suggests repairs, confirm or reject, reduces effort.

What isn’t a good repair?

```java
assertEquals(3, example.returnsAnInteger());
```

`true` ↓

```
assertTrue(true);
```

What is?

1. Make all tests pass.
An Example
On repair quality

- ReAssert suggests repairs, confirm or reject, reduces effort.
- What isn’t a good repair?
  
  ```java
  assertEquals(3, example.returnsAnInteger());
  ↓
  assertTrue(true);
  ```

- What is?
  
  1. Make all tests pass.
  2. Minimal test code changes.
ReAssert suggests repairs, confirm or reject, reduces effort.

What isn’t a good repair?

```java
assertEquals(3, example.returnsAnInteger());
```  

What is?

1. Make all tests pass.
2. Minimal test code changes.
3. Unchanged system under test.
An Example
On repair quality

- ReAssert suggests repairs, confirm or reject, reduces effort.
- What isn’t a good repair?
  ```java
  assertEquals(3, example.returnsAnInteger());
  ↓
  assertTrue(true);
  ```
- What is?
  1. Make all tests pass.
  2. Minimal test code changes.
  3. Unchanged system under test.
  4. No more bugs.
For Test Repair, SUT is oracle and not test suite.
Background
ReAssert Strategies

- For Test Repair, SUT is oracle and not test suite.
- Code structure, failure type, runtime values.
- Assertion failure:
  ```java
  assertEquals(3, example.example().returnsAnInteger());
  ```
- Replace literal '3' by literal '6': Trace declaration-use path strategy.
- Customized repair strategies.
ReAssert was first general purpose test repair tool.
- Good performance in case study and controlled user study.
- Not very good for open source, sub-optimal repairs.
- Main cause of issues: Expected values.
Modifications to expected values.

```java
int x = 5;
String expected = "the integer is " + x;
assertEquals(expected, example.returnsAString());
```

Naive non-useful repair:

```java
int x = 5;
String expected = "the integer is 6"; // ReAssert repair
assertEquals(expected, example.returnsAString());
```
Conclusion: changing literal values in test code works many times.

ReAssert issues: Couldn’t reliably identify required literals.

Idea: Use symbolic execution to discover literals that may repair test.

1. Stack trace to find location of failing assertion
2. Analyze source code to determine "expected side" of assertion.
3. Symbolic treatment to expected side.
4. Symbolic execution, solve accumulated constraints.
5. Replace appropriate literals.
Symbolic Execution with Pex

- Why suddenly Pex?
Why suddenly Pex? JPF wasn’t available at the time. :) .

Total fraction of repairable tests turned out to be $\sim$ identical for Java and .Net.

Literal replacement was often more useful than ReAssert suggested repairs.

Pex solved 53% – 92% of cases that ideal literal replacement would solve.
int someInt;
if (someCondition) {
    someInt = 3;
}
else {
    someInt = -3;
}
assertEquals(someInt, example.returnsAnInteger());
1. Stack trace to find location of failing assertion
2. Analyze source code to determine "expected side" of assertion.
3. Symbolic treatment to expected side.
4. Symbolic execution, solve accumulated constraints.
5. Replace appropriate literals.
int someInt;
if (someCondition) {
    someInt = PexChooseValue<int>("x");
}
else {
    someInt = PexChooseValue<int>("y");
}
assertEquals(someInt, example.returnsAnInteger());
Identify expected computation.

Glob target = new Glob("*eggs");
Assert.IsTrue(target.IsMatch("hamandeggs"));
Assert.IsTrue(target.IsMatch("eggs"));
Assert.IsFalse(target.IsMatch("hamandeggsandbacon"));

Find expected literals.
Correct choice.
Multiple Failures.
Evaluation Criteria

- How many failures can be repaired by replacing literals in test code?
- Comparision of literal replacement and ReAssert?
- How well can symbolic execution discover appropriate literals?
Evaluation

Setup

- Standard unit tests: JUnit for Java, .NET tests converted to run under Pex.
- Pex version 0.91.50418.0 in Microsoft Visual Studio 2009 on a dual-processor 1.8Ghz laptop.

<table>
<thead>
<tr>
<th>Application</th>
<th>Description</th>
<th>Tests</th>
</tr>
</thead>
<tbody>
<tr>
<td>Checkstyle</td>
<td>Code style checker</td>
<td>143</td>
</tr>
<tr>
<td>JDepend</td>
<td>Design quality metrics</td>
<td>53</td>
</tr>
<tr>
<td>JFreeChart</td>
<td>Chart creator</td>
<td>1696</td>
</tr>
<tr>
<td>Lucene</td>
<td>Text search engine</td>
<td>663</td>
</tr>
<tr>
<td>PMD</td>
<td>Java program analysis</td>
<td>448</td>
</tr>
<tr>
<td>XStream</td>
<td>XML serialization</td>
<td>726</td>
</tr>
<tr>
<td>checkstyle.sourceforge.net</td>
<td></td>
<td></td>
</tr>
<tr>
<td>clarkware.com/software/JDepend.html</td>
<td></td>
<td></td>
</tr>
<tr>
<td>jfree.org/jfreechart/</td>
<td></td>
<td></td>
</tr>
<tr>
<td>lucene.apache.org</td>
<td></td>
<td></td>
</tr>
<tr>
<td>pmd.sourceforge.net</td>
<td></td>
<td></td>
</tr>
<tr>
<td>xstream.codehaus.org</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Application</th>
<th>Description</th>
<th>Tests</th>
</tr>
</thead>
<tbody>
<tr>
<td>AdblockIE</td>
<td>Ad blocker for Internet Explorer</td>
<td>6</td>
</tr>
<tr>
<td>CSHgCmd</td>
<td>C# interface to Mercurial</td>
<td>16</td>
</tr>
<tr>
<td>Fudge-CSHarp</td>
<td>Binary message encoding</td>
<td>73</td>
</tr>
<tr>
<td>GCalExchangeSync</td>
<td>Google Calendars and</td>
<td>33</td>
</tr>
<tr>
<td>code.google.com/p/google-calendar-connectors/</td>
<td>Exchange Server interoperability</td>
<td>673</td>
</tr>
<tr>
<td>Json.NET</td>
<td>JSON serialization</td>
<td>673</td>
</tr>
<tr>
<td>MarkdownSharp</td>
<td>Convert structured text to HTML</td>
<td>48</td>
</tr>
<tr>
<td>NerdDinner</td>
<td>Lunch planning website</td>
<td>68</td>
</tr>
<tr>
<td>NGChart</td>
<td>Wrapper for Google Charts API</td>
<td>25</td>
</tr>
<tr>
<td>NHaml</td>
<td>XHTML template system</td>
<td>53</td>
</tr>
<tr>
<td>ProjectPilot</td>
<td>Source code statistics and metrics</td>
<td>103</td>
</tr>
<tr>
<td>SharpMap</td>
<td>Geospatial mapping</td>
<td>49</td>
</tr>
<tr>
<td>adblockie.codeplex.com</td>
<td></td>
<td></td>
</tr>
<tr>
<td>bitbucket.org/hg/cshgcmd/</td>
<td></td>
<td></td>
</tr>
<tr>
<td>github.com/FudgeMsg/Fudge-CSHarp/</td>
<td></td>
<td></td>
</tr>
<tr>
<td>code.google.com/p/google-calendar-connectors/</td>
<td></td>
<td></td>
</tr>
<tr>
<td>json.codeplex.com</td>
<td></td>
<td></td>
</tr>
<tr>
<td>code.google.com/p/markdownsharp/</td>
<td></td>
<td></td>
</tr>
<tr>
<td>nerdinner.codeplex.com</td>
<td></td>
<td></td>
</tr>
<tr>
<td>code.google.com/p/ngchart/</td>
<td></td>
<td></td>
</tr>
<tr>
<td>code.google.com/p/nhaml</td>
<td></td>
<td></td>
</tr>
<tr>
<td>code.google.com/p/projectpilot/</td>
<td></td>
<td></td>
</tr>
<tr>
<td>sharpmap.codeplex.com</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 1: Subject applications
Results

- How many failures can be repaired by replacing literals in test code?
- Comparison of literal replacement and ReAssert?
- How well can symbolic execution discover appropriate literals?
A few questions

1. Basing test repair on the SUT could produce passing tests for broken SUTs. How do you address this issue?

2. Looks like general debugging is extremely hard to solve and I didn’t see author’s discussion explaining why fixing bugs is generally possible (maybe a math proof or explain it in a meta level, etc)

3. In the code example in section 4.1, the literal in the “else” branch was not repaired because this branch was never executed. Is it able to mark LIB.is15 as another symbolic so that the symbolic execution engine can lead the program to the “else” branch? In this way, the “else” branch can be repaired as well. Is Symbolic Test Repair able to do this automatically? If no, is it easy to add this feature?
A few questions: Contd.

1. Does auto-fixing of testing code just make the test pass? If the logic in the function is wrong, modify test to make it pass is could not tell if this function works right.
2. What are the cases that can not be repaired using their test repair technique?
3. What is the difference between regression tests and general tests?
4. Java is at least as popular as .Net, why the state-of-the-art symbolic engine for java is far less powerful than Pex?
5. is this approach suitable for larger tests?(integration tests...)
6. What features of a project can affect the efficiency of automatic repairing with ReAssert and Lit. Repl? As we can see the ratio varies largely between projects such as Lucene and XStream
Thank you.