University of Illinois at Urbana-Champaign

On Test Repair Using Symbolic Execution

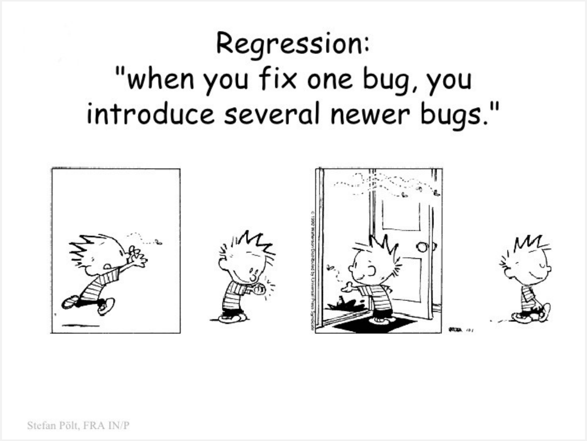
Presented by

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."

Stefan Pölt, FRA IN/P

- Problem?

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

Stefan Pölt, FRA IN/P

- Problem? Broken Tests.
- Automated repair of broken tests (ReAssert [Daniel et al. '09] + symbolic execution [Daniel et al. '10]

Outline

An Example
- Effect on tests
- On repair quality

Background
- ReAssert Strategies
- Symbolic Execution in Testing
- Challenges

Evaluation
- Experiments
- Results
- A few questions

An Example

Methods which change

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

Unit tests involving them

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

An Example
Effect on Tests

- The test

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

  fails, because the method changed.

An Example
Effect on Tests

- The test

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

  fails, because the method changed.
- Delete?

An Example
Effect on Tests

- The test

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

  fails, because the method changed.
- Delete?
- Repairing tests : problem?
- ReAssert [Daniel et al. '09] : automated test repair tool.

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?

```
assertEquals(3,example.returnsAnInteger());
                          ↓
                 assertTrue(true);
```

- What is?

An Example
On repair quality

- ReAssert suggests repairs, confirm or reject, reduces effort.
- What isn't a good repair?

```
assertEquals(3,example.returnsAnInteger());
                        ↓
              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?

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

$$\downarrow$$

```
assertTrue(true);
```

- What is?
  1. Make all tests pass.
  2. Minimal test code changes.

An Example
On repair quality

- ReAssert suggests repairs, confirm or reject, reduces effort.
- What isn't a good repair?

```
assertEquals(3,example.returnsAnInteger());
                    ↓
            assertTrue(true);
```

- 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?

```
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.

Background
ReAssert Strategies

- 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 :

  `assertEquals(3,example.returnsAnInteger());`
- Replace literal '3' by literal '6' : Trace declaration-use path strategy.
- Customized repair strategies.

Background
Reassert Evaluation (pre-Symbolic-test-repair

- 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.

Background
Motivating Example

- Modifications to expected values.

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

- Naive non-useful repair:

```
int x = 5;
String expected = "the integer is 6"; //ReAssert repair
assertEquals(expected,example.returnsAString());
```

Background
Motivation for symbolic execution

- 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?

Symbolic Execution with 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.

Background

An Example of Symbolic Test Repair

```
int someInt;
if (someCondition) {
    someInt = 3;
}
else {
    someInt = -3;
}
assertEquals(someInt,example.returnsAnInteger());
```

Background
Symbolic Execution steps

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.

Background
An Example of Symbolic Test Repair : Contd.

```
int someInt;
if (someCondition) {
    someInt = PexChooseValue<int>("x");
}
else {
    someInt = PexChooseValue<int>("y");
}
assertEquals(someInt,example.returnsAnInteger());
```

Background
Challenges

- Identify expected computation.

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

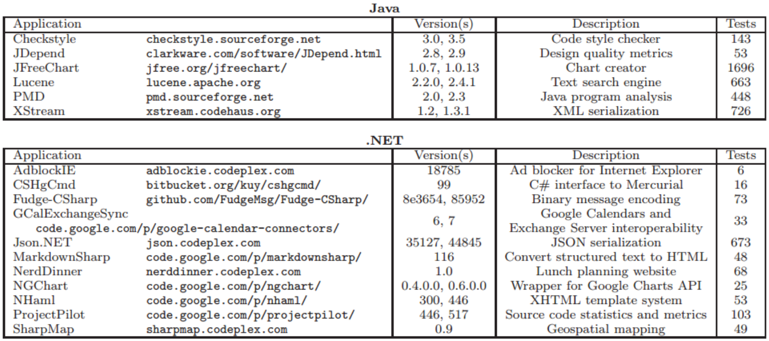
- Find expected literals.
- Correct choice.
- Multiple Failures.

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.



Java

Application		Version(s)	Description	Tests
Checkstyle	`checkstyle.sourceforge.net`	3.0, 3.5	Code style checker	143
JDepend	`clarkware.com/software/JDepend.html`	2.8, 2.9	Design quality metrics	53
JFreeChart	`jfree.org/jfreechart/`	1.0.7, 1.0.13	Chart creator	1696
Lucene	`lucene.apache.org`	2.2.0, 2.4.1	Text search engine	663
PMD	`pmd.sourceforge.net`	2.0, 2.3	Java program analysis	448
XStream	`xstream.codehaus.org`	1.2, 1.3.1	XML serialization	726

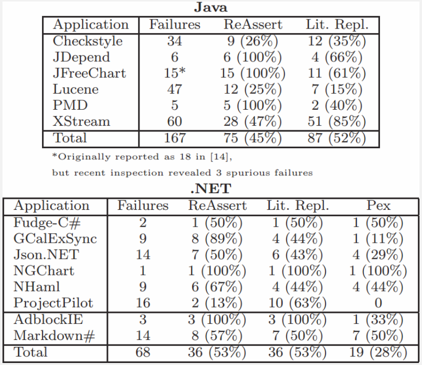
.NET

Application		Version(s)	Description	Tests
AdblockIE	`adblockie.codeplex.com`	18785	Ad blocker for Internet Explorer	6
CSHgCmd	`bitbucket.org/kuy/cshgcmd/`	99	C# interface to Mercurial	16
Fudge-CSharp	`github.com/FudgeMsg/Fudge-CSharp/`	8e3654, 85952	Binary message encoding	73
GCalExchangeSync	`code.google.com/p/google-calendar-connectors/`	6, 7	Google Calendars and Exchange Server interoperability	33
Json.NET	`json.codeplex.com`	35127, 44845	JSON serialization	673
MarkdownSharp	`code.google.com/p/markdownsharp/`	116	Convert structured text to HTML	48
NerdDinner	`nerddinner.codeplex.com`	1.0	Lunch planning website	68
NGChart	`code.google.com/p/ngchart/`	0.4.0.0, 0.6.0.0	Wrapper for Google Charts API	25
NHaml	`code.google.com/p/nhaml/`	300, 446	XHTML template system	53
ProjectPilot	`code.google.com/p/projectpilot/`	446, 517	Source code statistics and metrics	103
SharpMap	`sharpmap.codeplex.com`	0.9	Geospatial mapping	49

Figure 1: Subject applications

Results

- 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?



Java

Application	Failures	ReAssert	Lit. Repl.
Checkstyle	34	9 (26%)	12 (35%)
JDepend	6	6 (100%)	4 (66%)
JFreeChart	15*	15 (100%)	11 (61%)
Lucene	47	12 (25%)	7 (15%)
PMD	5	5 (100%)	2 (40%)
XStream	60	28 (47%)	51 (85%)
Total	167	75 (45%)	87 (52%)

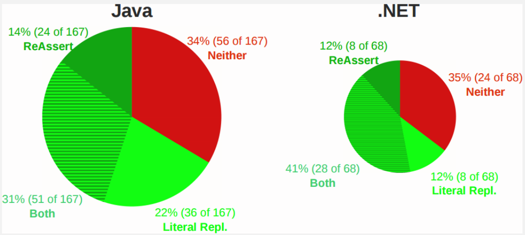
*Originally reported as 18 in [14], but recent inspection revealed 3 spurious failures

.NET

Application	Failures	ReAssert	Lit. Repl.	Pex
Fudge-C#	2	1 (50%)	1 (50%)	1 (50%)
GCalExSync	9	8 (89%)	4 (44%)	1 (11%)
Json.NET	14	7 (50%)	6 (43%)	4 (29%)
NGChart	1	1 (100%)	1 (100%)	1 (100%)
NHaml	9	6 (67%)	4 (44%)	4 (44%)
ProjectPilot	16	2 (13%)	10 (63%)	0
AdblockIE	3	3 (100%)	3 (100%)	1 (33%)
Markdown#	14	8 (57%)	7 (50%)	7 (50%)
Total	68	36 (53%)	36 (53%)	19 (28%)

Results

- 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?



A few questions

1. Basing test repair on the SUT could produce passing tests for broken SUTs. How do you address this issue?
2. 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. is this approach suitable for larger tests?(integration tests...)

Conclusions

- ReAssert (the precursor of this work) itself was a significant idea that addresses a crucial part of software development.
- Symbolic test repair is extremely attractive theoretically, and proves to be (adequately?) successful in the experimental setup of this work .
- Not complete (quite possibly the only possible alternative).
- Some updates to this work are in order.

Thank you .