Automatic Patch Generation Learned from Human Written Patches

Dongsun Kim, Jaechang Nam, Jaewoo Song, and Sunghun Kim
The Hong Kong University of Science and Technology, China

slide author names omitted for FERPA compliance
Introduction – Automatic Program Repair

• Two types of approaches
  • Testing-based (GenProg)
  • Specification-based

• GenProg

Identify suspicious statements in program (Fault Localization)

Perform random mutation and crossover operations to modify the suspicious statements with other program statements to generate program variants

Test program variants on test suit. Keep generating variants unless one passes all test cases of test suit

Random Mutation
Problem Statement

• Using random mutation and crossover operations generates many nonsensical patches. How can we address this limitation?

• How about analyzing existing human-written patches to identify common patterns and use them to generate patches?
Creating Patterns – Manual Analysis of Human-written patches

- Analyzed 62,656 Human-written patches from open-source project Eclipse JDT
- Focus on Semantic rather than Syntactic changes
- Analysis Involved:
  1. Does patch involves any semantic changes?
  2. What is the root cause of the bug and resolution of the corresponding patch?
  3. Group similar patches considering 1 and 2
- Reduce manual inspection time using *groums*
# Fix Patterns

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Altering method parameters</td>
<td><code>obj.method(v1,v2) → obj.method(v1,v3)</code></td>
</tr>
<tr>
<td>Calling another method with the same parameters</td>
<td><code>obj.method1(param) → obj.method2(param)</code></td>
</tr>
<tr>
<td>Calling another overloaded method with one more parameter</td>
<td><code>obj.method(v1) → obj.method(v1,v2)</code></td>
</tr>
<tr>
<td>Changing a branch condition</td>
<td><code>if(a == b) → if(a == b &amp;&amp; c != 0)</code></td>
</tr>
<tr>
<td>Initializing an object</td>
<td><code>Type obj; → Type obj = new Type()</code></td>
</tr>
<tr>
<td>Adding a “null”, “array-out-of-bound” and, “class cast” checker</td>
<td><code>obj.m1() → if(obj!=null){obj.m1()}</code></td>
</tr>
</tbody>
</table>
Fix Template

• Derived from Fix pattern
• Find out the difference between Abstract Syntax Trees (ASTs) of a program before and after applying human patch
• Transform this difference into editing scripts
• Example
Pattern-Based Automatic Program Repair (PAR)

- Analyze existing human-written patches to identify common fix patterns
- Utilize these fix patterns to create Fix templates
- Use Fix templates to automatically fix the bugs in code.
PAR-Example of Bug Repair

### Buggy Program

```
01 if (kidMatch != -1) return kidMatch;
02 for (int i = num; i < state.parenCount; i++)
03 {
04     state.paren[i].length = 0;
05 }
06 state.parenCount = num;
```

### Suspicious Statement obtained using Fault Localization

```
state.paren[i].length = 0;
```

### <Null Pointer Checker>

- **INPUT:** state.paren[i].length = 0;
- **1. Analyze:** Extract obj refer to state, state.paren[i]
- **2. Context Check:** object references? PASS
- **3. Edit:** INSERT

```
... + if( state != null && state.paren[i] != null ) {
    state.paren[i].length = 0;
} ... 
```

### Repaired Program

```
01 if (kidMatch != -1) return kidMatch;
02 for ( ... )
03 {
04+    if( state != null && state.paren[i] != null ) {
05        state.paren[i].length = 0;
06    }
07     state.parenCount = num;
```

### Fix Template
Research Questions

• RQ1: How many bugs can the PAR fix? (Fixability)

• RQ2: Are patches generated by PAR sensible? (Acceptability)
RQ1: How many bugs can the PAR fix? (Fixability)

- Applied PAR to 119 real bugs from 5 open source Java projects (Rhino, AspectJ, log4J, Math, Lang, and Collections)
- PAR fixed 27 out of 119 bugs
- GenProg fixed 16 bugs.
- 5 bugs fixed by both techniques
- **PAR can fix more bugs than GenProg**
RQ2: Are patches generated by PAR sensible? (Acceptability)

• Conducted two user studies to compare the patch acceptability (89 students and 164 developers)

• Direct Patch Comparison Rankings (17 students, 68 developers, 5 patches)
  • 1.72 (Human-written) vs 1.57 (PAR) vs 2.67 (GenProg) in case of students
  • 1.81 (Human-written) vs 1.82 (PAR) vs 2.36 (GenProg) in case of developers

• Indirect Patch Comparison Results (72 students, 96 developers, 43 patches)
  • PAR acceptable in 305 (49%) out of 621 sessions (PAR: 21% + both: 28%)
  • GenProg acceptable in 108 (32%) out of 344 sessions (GenProg: 20% + both: 12%).

• Patches generated by PAR are more Acceptable (comparable to human-written patches) than GenProg
• Using random mutation and crossover operations generates many nonsensical patches. How can we address this limitation?

• Use PAR

Buggy Program

```java
1918 if (lhs == DBL_MRK) lhs = ...;
1919 if (lhs == undefined) {  
  1920   lhs = strings[getShort(iCode, pc + 1)];
1921 }
1922 Scriptable calleeScope = scope;
```

GenProg

```java
1918 if (lhs == DBL_MRK) lhs = ...;
1919 if (lhs == undefined) {
  1920+   lhs = ((Scriptable)lhs).defaultValue(null);
1921 }
1922 Scriptable calleeScope = scope;
```

Human

```java
1918 if (lhs == DBL_MRK) lhs = ...;
1919 if (lhs == undefined) {
  1920   i = getShort(iCode, pc + 1);
1921+   if (i != -1)
1922+     lhs = strings[i];
1923 }
1924 Scriptable calleeScope = scope;
```

PAR

```java
1918 if (lhs == DBL_MRK) lhs = ...;
1919 if (lhs == undefined) {
  1920+   if (getShort(iCode, pc + 1) < strings.length &&
           getShort(iCode, pc + 1) >= 0)
1921+     {
1922       lhs = strings[getShort(iCode, pc + 1)];
1923+     }
1924 }  
1925 Scriptable calleeScope = scope;
```
Major Contributions

• Manual inspection of human-written patches to identify and create common fix patterns.

• A novel patch generation approach – **Pattern-Based Automatic Program Repair (PAR)** that utilizes *Fix templates* derived from *Fix patterns*.

• Empirical evaluation of the approach by applying PAR to 119 real bugs and performing user study involving 89 students and 164 developers.
Discussion

Patch Acceptability

Buggy Program

1918 if (lhs == DBL_MRK) lhs = ...;
1919 if (lhs == undefined) {
1920+    lhs = ((Scriptable)lhs).getDefaultValue(null);
1921+  }
1922 Scriptable calleeScope = scope;

Gives array-index-out-of-bound error

PAR

1918 if (lhs == DBL_MRK) lhs = ...;
1919 if (lhs == undefined) {
1920+   i = getShort(iCode, pc + 1);
1921+   if (i != -1)
1922+     lhs = strings[i];
1923  } 1924 Scriptable calleeScope = scope;

GenProg

1918 if (lhs == DBL_MRK) lhs = ...;
1919 if (lhs == undefined) {
1920+   if (getShort(iCode, pc + 1) < strings.length &&
1921+     getShort(iCode, pc + 1) >= 0)
1922+     {
1923+       lhs = strings[getShort(iCode, pc + 1)];
1924+     }
1925 Scriptable calleeScope = scope;

Human
Discussion

Generalizability, Scalability of approach

• Top 8 Fix patterns identified are found to be used in many (almost 30%) of the real patches

• patches of closed-source projects may have different patterns
Discussion

Types of Bugs fixed, Validity of comparison with GenProg