A Fact of a Software Life
Not just “a”.....

- Windows 2000 shipped with more than 63,000 KNOWN bugs
- In 2005, almost 300 bugs were appearing daily in Mozilla, according to one of its developer

Manual Repair?
Not just “a”.....

- Windows 2000 shipped with more than 63,000 KNOWN bugs
- In 2005, almost 300 bugs were appearing daily in Mozilla, according to one of its developer

Manual Repair?
Automatic Repair!

GenProg  (one of several others)

(a) Buggy program. Line 1920 throws an *Array Index Out of Bound* exception when `getShort(iCode, pc + 1)` is equal to or larger than `strings.length` or smaller than 0.

(b) Patch generated by GenProg.
Automatic Repair

GenProg (HW!)

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

(a) Buggy program. Line 1920 throws an Array Index Out of Bound exception when getShort (iCode, pc + 1) is equal to or larger than strings.length or smaller than 0.

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

(b) Patch generated by GenProg.
Automatic Patch Generation
Learned from Human-Written Patches

DONGSUN KIM, JAECHANG NAM,
JAEWOO SONG, AND SUNGHUN KIM

Hong Kong University of Science and technology, China
Pattern-based Automatic Program Repair (PAR)

- Manual Observations on Human Written Patches
  - Fix Patterns!

- PAR
  - Fix Patterns -> Fix Templates -> Automatic Patches

- Empirical Evaluation
  - Test PAR on 119 Real Bugs
Manual Observations of Human Written Patches

- Patch Collection – 62,656 human written patches from Eclipse JDT
- Common Patches Mining
  - Identify patches as additive, subtractive, or altering
  - Examine root causes of bugs & how patches specifically resolved the bugs
- Group similar patches into common patterns
Common Fix Patterns!

Top eight patterns cover almost 30% of all patches observed!

TABLE I: Common fix patterns identified from Eclipse JDT’s patches.

<table>
<thead>
<tr>
<th>Fix Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>Altering method parameters</td>
</tr>
<tr>
<td>Calling another method with the same parameters</td>
</tr>
<tr>
<td>Calling another overloaded method with one more parameter</td>
</tr>
<tr>
<td>Changing a branch condition</td>
</tr>
<tr>
<td>Adding a null checker</td>
</tr>
<tr>
<td>Initializing an object</td>
</tr>
<tr>
<td>Adding an array bound checker</td>
</tr>
<tr>
<td>Adding a class-cast checker</td>
</tr>
<tr>
<td>Pattern</td>
</tr>
<tr>
<td>----------------------------------------------</td>
</tr>
<tr>
<td>Altering method parameters</td>
</tr>
<tr>
<td>Calling another method with the same parameters</td>
</tr>
<tr>
<td>Changing a branch condition</td>
</tr>
</tbody>
</table>
PAR – In Action

- (1) Identifies fault locations
- (2) Uses fix templates to generate program variants
- (3) Evaluates program variants by fitness function (computes number of passing tests of patch candidate)
- (4) If candidate passes all tests, then SUCCESS!
  
  ELSE Repeat (2) & (3)
Algorithm 1: Patch generation using fix templates in PAR.

Input: fitness function $Fit$: Program $\to \mathbb{R}$
Input: $T$: a set of fix templates
Input: $PopSize$: population size
Output: $Patch$: a program variant that passes all test cases

1. $Pop \leftarrow initialPopulation(PopSize)$;
2. repeat
3.     $Pop \leftarrow apply(Pop,T)$;
4.     $Pop \leftarrow select(Pop,PopSize,Fit)$;
5. until $\exists Patch$ in $Pop$ that passes all test cases;
6. return $Patch$
Fault Localization

- Statistical fault localization based on test cases
- Assumes that a statement visited by failing tests is more likely to be a defect than other statements

- (1) Executes two groups of tests: passing and failing
- (2) Records the statement coverage of both test case groups
  - (a) Covered by both groups
  - (b) Covered only by Passing group
  - (c) Covered only by Failing group
  - (d) Not covered by either group

Assign a value of 0.1 to statements in (a), 1.0 to (c), 0.0 otherwise
Fix Template

• AST Analysis – Scans program’s AST and analyzes fault location and adjacent location

• Context Check – Examines whether the program can be edited by a template by inspecting the analyzed AST

• Program Editing – If possible, rewrite the program’s AST based on the script in the template

Fig. 4: Null pointer checker fix template. This template inserts an if() statement checking whether objects are null.
Fitness Evaluation

- Fitness Function
  - (Program variant, test cases) -> Compute value representing the number of passing test cases of the variant
  - Resulting fitness value used for evaluating and comparing program variants in a population
  - Based on fitness values of program variants, PAR chooses program variants by tournament selection
Examples:

(a) Buggy Program: the underlined statement is a fault location.

```
if (kidMatch != -1) return kidMatch;
for (int i = num; i < state.parenCount; i++)
    state.paren[i].length = 0;
state.parenCount = num;
```

**<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;
    }
```

**OUTPUT:** a new program variant

(b) After applying a fix template: a patch generated by PAR. As shown in the fix template, corresponding statements have been edited.

Fig. 5: Real example of applying a fix template to NativeRegExp.java to fix Rhino Bug #76683.
Examples:

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

(a) Buggy program. Line 1920 throws an Array Index Out of Bound exception when getShort(iCode, pc + 1) is equal to or larger than strings.length or smaller than 0.

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

(b) Patch generated by GenProg.
Examples:

```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;
```

(d) Patch generated by PAR.
Evaluation of PAR

- Two Research Questions
  
  - RQ1: (Fixability) How many bugs are fixed successfully?
  
  - RQ2: (Acceptability) Which approach can generate more acceptable bug patches?
Experimental Design

- Collected 119 bugs from open source projects
- Applied both PAR and GenProg to each bug to generate patches
- Examined how many bugs were successfully fixed by each (RQ1)
- Conducted user study to compare patch quality (RQ2)

<table>
<thead>
<tr>
<th>Subject</th>
<th># bugs</th>
<th>LOC</th>
<th># statements</th>
<th># test cases</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rhino</td>
<td>17</td>
<td>51,001</td>
<td>35,161</td>
<td>5,578</td>
<td>interpreter</td>
</tr>
<tr>
<td>AspectJ</td>
<td>18</td>
<td>180,394</td>
<td>139,777</td>
<td>1,602</td>
<td>compiler</td>
</tr>
<tr>
<td>log4j</td>
<td>15</td>
<td>27,855</td>
<td>19,933</td>
<td>705</td>
<td>logger</td>
</tr>
<tr>
<td>Math</td>
<td>29</td>
<td>121,168</td>
<td>80,764</td>
<td>3,538</td>
<td>math utils</td>
</tr>
<tr>
<td>Lang</td>
<td>20</td>
<td>54,537</td>
<td>40,436</td>
<td>2,051</td>
<td>helper utils</td>
</tr>
<tr>
<td>Collections</td>
<td>20</td>
<td>48,049</td>
<td>35,335</td>
<td>11,577</td>
<td>data utils</td>
</tr>
<tr>
<td>Total</td>
<td>119</td>
<td>483,004</td>
<td>351,406</td>
<td>25,051</td>
<td></td>
</tr>
</tbody>
</table>
Experimental Design

- All six projects written in Java
- All six projects commonly used in literature and have well maintained bug report
- Randomly selected 15-29 bugs per project
- For each bug collected all available test cases
- Conducted 100 runs for each bug per approach. Total runs (100 * 119 * 2 = 23,800)
- Each run stopped when it took more than 10 generations or 8 hours, which meant it failed at creating a successful patch.
RQ1 Fixability

- **PAR 27  GenProg 16**
- 5 bugs fixed by both but patch for each were different. To be used for RQ2
- Fix patterns were generated from Eclipse JDT but applied to bugs of other projects
- Only used limited number of fix patterns. Can improve fixability?
- GenProg 10% in Java vs 50% in C?

<table>
<thead>
<tr>
<th>Subject</th>
<th># bugs</th>
<th># bugs fixed by GenProg</th>
<th># bugs fixed by PAR</th>
<th># bugs fixed by both</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rhino</td>
<td>17</td>
<td>7</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>AspectJ</td>
<td>18</td>
<td>0</td>
<td>9</td>
<td>0</td>
</tr>
<tr>
<td>log4j</td>
<td>15</td>
<td>0</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>Math</td>
<td>29</td>
<td>5</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Lang</td>
<td>20</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Collections</td>
<td>20</td>
<td>3</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>119</strong></td>
<td><strong>16</strong></td>
<td><strong>27</strong></td>
<td><strong>5</strong></td>
</tr>
</tbody>
</table>
RQ2 (Acceptability)

Formulated two null hypotheses

- $H_{10}$: Patches generated by PAR and GenProg have no acceptability difference from each other.
- $H_{20}$: Patches generated by PAR have no acceptability difference from human-written patches.

The corresponding alternative hypotheses are:

- $H_{1a}$: PAR generates more acceptable patches than GenProg.
- $H_{2a}$: Patches generated by PAR are more acceptable than human-written patches.
RQ2 (Acceptability)

Subjects
- Two groups (CS students and Developers)
- 17 software engineering graduate students with Java experience
- 68 developers (online developer communities and software companies)

Study Design
- Five sessions.
- One of five bugs per session fixed by both PAR and GenProg
- Each session explained bug in detail
- Session listed three anonymized patches (human, PAR, GenProg)
- Participant asked to compare and rank
RQ2 (Acceptability)

- **Result – Students**
  - PAR patches consistently ranked higher than GenProg patches
  - Average ranking of PAR patches = 1.57 (SD = 0.68)
  - Average ranking of GenProg patches = 2.67 (SD = 0.64)
  - Ranking differences between Par and GenProg are statistically significant (p-value = 0.000 < 0.05)
  - Based on results, reject null hypothesis H10 for student group

**TABLE V: Average rankings evaluated by 17 students (standard deviation is shown in parentheses). The lower values indicate that the patch obtained higher rankings on average by the evaluators.**

<table>
<thead>
<tr>
<th>Bugs</th>
<th>Human</th>
<th>PAR</th>
<th>GenProg</th>
</tr>
</thead>
<tbody>
<tr>
<td>Math #280</td>
<td>1.33 (0.62)</td>
<td>2.27 (0.59)</td>
<td>2.40 (0.83)</td>
</tr>
<tr>
<td>Rhino #114493</td>
<td>2.00 (0.54)</td>
<td>1.33 (0.62)</td>
<td>2.67 (0.72)</td>
</tr>
<tr>
<td>Rhino #192226</td>
<td>1.47 (0.64)</td>
<td>1.67 (0.62)</td>
<td>2.67 (0.72)</td>
</tr>
<tr>
<td>Rhino #217379</td>
<td>1.69 (0.70)</td>
<td>1.50 (0.63)</td>
<td>2.81 (0.40)</td>
</tr>
<tr>
<td>Rhino #76683</td>
<td>2.13 (0.51)</td>
<td>1.07 (0.26)</td>
<td>2.80 (0.41)</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td>1.72 (0.67)</td>
<td>1.57 (0.68)</td>
<td>2.67 (0.64)</td>
</tr>
</tbody>
</table>
RQ2 (Acceptability)

- **Result – Developers**
  - Similarly PAR ranked higher than GenProg patches except one
  - Average ranking of PAR patches = 1.82 (SD = 0.80)
  - Average ranking of GenProg patches = 2.36 (SD = 0.90)
  - Ranking differences between Par and GenProg are statistically significant (p-value = 0.016 < 0.05)
  - Based on results, reject null hypothesis H10 for developer group
RQ2 (Acceptability)

- Results of comparative studies:
  - PAR patches consistently have higher rankings than GenProg patches
  - Results are statistically significant
  - Implication? PAR can generate more acceptable patches than GenProg

- Ranking differences not statistically significant between Par and human written patches
- Implication? Patches generated by PAR are comparable to human written patches
RQ2 (Acceptability)

- Indirect Patch Comparison
  - Compare acceptability of all 43 patches (27 PAR, 16 GenProg) to human-written patches
- Web online Survey
  - Each session showed anonymized patches (one human written and one corresponding PAR or GenProg)
  - Patches generated by PAR more acceptable (21% + 28%) than GenProg patches (20% + 12%)

<table>
<thead>
<tr>
<th>Selection</th>
<th># response</th>
<th>Selection</th>
<th># response</th>
</tr>
</thead>
<tbody>
<tr>
<td>PAR</td>
<td>130 (21%)</td>
<td>GenProg</td>
<td>68 (20%)</td>
</tr>
<tr>
<td>Both</td>
<td>175 (28%)</td>
<td>Both</td>
<td>40 (12%)</td>
</tr>
<tr>
<td>Human</td>
<td>229 (37%)</td>
<td>Human</td>
<td>176 (51%)</td>
</tr>
<tr>
<td>Not Sure</td>
<td>87 (14%)</td>
<td>Not Sure</td>
<td>60 (17%)</td>
</tr>
<tr>
<td>Total</td>
<td>621 (100%)</td>
<td>Total</td>
<td>344 (100%)</td>
</tr>
</tbody>
</table>
What about the other 92?

- 92 out of 119 bugs not patched
- Branch conditions (28%)
  - Cannot generate predicates to satisfy branch conditions at fault locations by using fix templates
- No matching pattern (72%)
  - Cannot generate a patch because no fix template has appropriate editing scripts
Threats to Validity

- Systems are all open source projects
  - Patches of closed-source projects may have different patterns
- Some user studies participants may not be qualified
  - Could not verify qualifications of developers
Manually inspected human-written patches and discovered common fix patterns

Used fix patterns to generate automatic patches (PAR)

Evaluated the patches against GenProg patches and human written patches

PAR was more successful than GenProg generated 27 successful patches vs 16 by GenProg. PAR patches comparable to human-written patches.

Future Work

- Automatic fix template mining
- Balance test case generation
Questions:

▶ Quiz!

▶ What is the scientific question? the answer?

▶ What’s the key new idea that allows answering it?

▶ How do you measure the success of the answer?
Questions:

Can we identify additional Threats to Validity?
Questions:

Which one do you think is more efficient? GenProg or PAR?
Questions:

Patch hunting within program vs Patch hunting outside of program

Which is better?
Questions:

» PAR vs GenProg
» Apples to apples comparison?
Sources used:

- “Automatic Patch Generation Learned from Human-Written Patches” by Kim et al
- “Automatically Finding Patches Using Genetic Programming” by Weimer et al
- Several Minions and otherwise images from throughout the web
- One slide from Professor Yuriy Brun’s class!