Dissecting Copy/Delete/Replace/Swap mutations: Insights from a GIN Case Study

Sherlock A. Licorish  
University of Otago  
Dunedin, New Zealand  
sherlock.licorish@otago.ac.nz

Markus Wagner*  
The University of Adelaide  
Adelaide, Australia  
markus.wagner@adelaide.edu.au

Repo: https://github.com/markuswagnergithub/combining_sa_and_gi
Code Reuse in Stack Overflow and Popular Open Source Java Projects

Adriaan Lotter  
Department of Information Science  
University of Otago  
Dunedin, New Zealand  
adriaan.lotter@otago.ac.nz

Sherlock A. Licorish  
Department of Information Science  
University of Otago  
Dunedin, New Zealand  
sherlock.licorish@otago.ac.nz

Sarah Meldrum  
Department of Information Science  
University of Otago  
Dunedin, New Zealand  
sarah-meldrum@outlook.com

Bastin Tony Roy Savarimuthu  
Department of Information Science  
University of Otago  
Dunedin, New Zealand  
tony.savarimuthu@otago.ac.nz

Abstract—Solutions provided in Question and Answer (Q&A) websites such as Stack Overflow are regularly used in Open Source Software (OSS) projects. However, users are often uncertain about the reliability and maintainability. While code reuse allows for previously tested and quality-assured code to be implemented in a
Challenge

• Snippets online can often be incorrect, insecure, and incomplete

• We have observed errors in Stack Overflow code

• These observations extend to students’ work, across multiple universities

• Errors have also been reported by open source developers, proprietary developers, and end users… the software development community

“App explanation: the sprit of stack overflow is coders helping coders”
- NissanConnect EV mobile app
Phase I

Combining GIN and PMD for Code Improvements

https://arxiv.org/abs/2202.01490
Question: Can genetic improvement improve the health of snippets?
Phase I - Empirical Approach

• Snippets (8,010) extracted from Stack Overflow for 2014, 2015, and 2016 using Stack Overflow’s data explorer

• Answer posts which contained at least one “<code>” tag and were from a question tagged from Java were then sampled

• Static checker PMD used to identify faults, https://pmd.github.io/

• Genetic improvement tool GIN used for code repair, https://github.com/gintool/gin

• We focus on performance related faults in Stack Overflow’s code
Characterizing PMD’s Treatment of 8,010 Snippets

• PMD finds 30,668 rule violations in 3,034 snippets, covering 135 of its 324 rules:

<table>
<thead>
<tr>
<th>PMD ruleset</th>
<th>total number of violations</th>
<th>different rules violated (total)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Style CS</td>
<td>16832</td>
<td>31 (64)</td>
</tr>
<tr>
<td>Documentation DOC</td>
<td>6292</td>
<td>3 (5)</td>
</tr>
<tr>
<td>Best Practice BP</td>
<td>3557</td>
<td>23 (57)</td>
</tr>
<tr>
<td>Design DES</td>
<td>2785</td>
<td>26 (48)</td>
</tr>
<tr>
<td>Error Prone EP</td>
<td>778</td>
<td>31 (103)</td>
</tr>
<tr>
<td>Performance PER</td>
<td>396</td>
<td>17 (32)</td>
</tr>
<tr>
<td>Multi-Threadining MT</td>
<td>28</td>
<td>4 (11)</td>
</tr>
<tr>
<td>Security SEC</td>
<td>0</td>
<td>0 (4)</td>
</tr>
</tbody>
</table>

• Examples of performance related rule violations:

<table>
<thead>
<tr>
<th>rule</th>
<th>count</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UseStringBufferForStringAppends</td>
<td>118</td>
<td>Prefer StringBuilder (non-synchronized) or StringBuffer (synchronized) over +=</td>
</tr>
<tr>
<td>AddEmptyString</td>
<td>54</td>
<td>Do not add empty strings.</td>
</tr>
<tr>
<td>AppendCharacterWithChar</td>
<td>35</td>
<td>Avoid appending characters as strings in StringBuffer.append.</td>
</tr>
<tr>
<td>RedundantFieldInitializer</td>
<td>23</td>
<td>Avoid using redundant field initializer for &lt;i&gt;.</td>
</tr>
<tr>
<td>AvoidInstantiatingObjectsInLoops</td>
<td>19</td>
<td>Avoid instantiating new objects inside loops.</td>
</tr>
<tr>
<td>AvoidArrayLoops</td>
<td>19</td>
<td>System.arraycopy is more efficient.</td>
</tr>
<tr>
<td>UseIndexOfChar</td>
<td>12</td>
<td>String.indexOf(char) is faster than String.indexOf(String).</td>
</tr>
<tr>
<td>StringInstantiation</td>
<td>11</td>
<td>Avoid instantiating String objects; this is usually unnecessary.</td>
</tr>
</tbody>
</table>
Characterizing GIN’s Single-edit Space

- GIN’s RandomSampler samples and runs 17,986 unique single-edit patches (DeleteLine, ReplaceLine, CopyLine, and SwapLine; and DeleteStatement, ReplaceStatement, CopyStatement, and SwapStatement; in total 31.4% compile)

- 770 patches: files no longer have any performance issues – according to PMD

- 58 (for 44 unique files) patches produce compilable code without performance issues
  - 36 are Delete edits that delete the offending code
  - most others either replace or modify the offending code

Example: Code snippet C66208 with error AppendCharacterWithChar, mutation DeleteStatement(64). The deleted statement is shown in red. For more examples, see the GI@GECCO paper “Dissecting Copy/Delete/Replace/Swap mutations: Insights from a GIN Case Study”.

```java
public class C66208{
    public static String expand(String word) {
        int stringLength = word.length();
        StringBuffer buffer = new StringBuffer();
        for (int i = 0; i < stringLength - 1; i++) {
            buffer.append(word.substring(i, i + 1));
            buffer.append("-");
        }
        buffer.append(word.substring(stringLength - 1, stringLength));
        return buffer.toString();
    }
}
```

- Non-uniform effects of edits types
  - Copy edits attract disproportionally many violations
  - Delete edits perform best against the AvoidInstantiatingObjectsInLoops violations
Future Work/Threats

• Better static analysis:
  - Mitigate false positive and trivial warnings
  - Improve parsing of non-compilable code
  - Crowd-source rules

• Better automated program improvement:
  - Bias sampling towards desired effects
  - Better code transformations
  - Other non-functional properties

Threat: GIN is normally accompanied by unit test suites to assess the validity of mutants. This work does not adopt such tests, and thus our successful patches that cleared performance issues and resulted in compilable code could have been inflated.
Phase II

Dissecting Copy/Delete/Replace/Swap mutations: Insights from a GIN Case Study

Effectiveness of GIN’s Mutations

```
1 public class C264051{
2   public static int gcd(int a, int b) {
3       if (b == 0) {
4           return a;
5       } else {
6           return gcd(b, a % b);
7       }
8   }
9
10   public static int pairwisePrimes(int k) {
11      int numWays = 0;
12      for (int i = 1; i < k; i++) {
13          for (int b = i + 1; b < k; b++) {
14              if ((i + b + c = k) && gcd(b, b) == 1 && gcd(a, c) == 1) {
15                  System.out.println("" + a + "+" + b + "+" + c);
16                  numWays++;
17              
18          }
19      }
20  }
21  }
22  return numWays;
23}
24```

Listing 4: Code snippet C264051 with error AddEmptyString, mutation DeleteLine(16). The deleted line is shown in red.

```
1 public class C33902{
2   public int[] getRandom(int[] array, int index, int size) {
3       int[] subArray = new int[size];
4       subArray[index] = 0;
5       for (int i = index; i < index + size; i++) {
6         subArray[i] = array[i];
7         subArrayIndex++;
8         array[ subArrayIndex ] = array[i];
9       }
10      return subArray;
11  }
12```

Listing 7: Code snippet C33902 with error AvoidArrayLoops, mutation ReplaceLine(6,7). The removed code is shown in red and the introduced code is shown in blue.

```
// original
1 public class C33977{
2   public void printStrings(String a, int b) {
3     String printString = "";
4     for (int i = 0; i < b; i++) {
5         printString = printString + a;
6     }
7     System.out.println(printString);
8  }
```

Question: How effective are mutations performed by GIN?
Phase II - Empirical Approach

• The 58 single-edit mutations (of 44 different snippets) that no longer show any performance issues and the code is compilable

• One issue is removed in 54 cases, and two issues are removed in four cases

• We manually annotate the 58 mutations with a focus on whether or not a human would deem the mutation acceptable, by:
  (1) Describing the change to the semantics of the program
  (2) Answering the question: “Are the semantics retained? Possible answers: yes/mostly/no”

• We performed two rounds of analyses to ensure consistency in the manual analysis performed by the two authors
GIN’s Repair Observations

• PMD performance-related errors in the original 44 code snippets

<table>
<thead>
<tr>
<th>rule</th>
<th>count</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UseStringBufferForStringAppends</td>
<td>26</td>
<td>Prefer StringBuilder (non-synchronized) or StringBuffer (synchronized) over <code>+=</code> for concatenating strings.</td>
</tr>
<tr>
<td>AvoidArrayLoops</td>
<td>15</td>
<td>System.arraycopy is more efficient.</td>
</tr>
<tr>
<td>AddEmptyString</td>
<td>6</td>
<td>Do not add empty strings.</td>
</tr>
<tr>
<td>AppendCharacterWithChar</td>
<td>5</td>
<td>Avoid appending characters as strings in StringBuffer.append.</td>
</tr>
<tr>
<td>InefficientStringBuffering</td>
<td>3</td>
<td>Avoid concatenating nonliterals in a StringBuffer/StringBuilder constructor or append().</td>
</tr>
<tr>
<td>InefficientEmptyStringCheck</td>
<td>2</td>
<td>String.trim().length() == 0 / String.trim().isEmpty() is an inefficient way to validate a blank String.</td>
</tr>
<tr>
<td>TooFewBranchesForASwitchStatement</td>
<td>2</td>
<td>A switch with less than three branches is inefficient, use an if statement instead.</td>
</tr>
<tr>
<td>AvoidInstantiatingObjectsInLoops</td>
<td>2</td>
<td>Avoid instantiating new objects inside loops.</td>
</tr>
<tr>
<td>ConsecutiveLiteralAppends</td>
<td>1</td>
<td>StringBuffer (or StringBuilder).append is called <code>&lt;3&gt;</code> consecutive times with literals.</td>
</tr>
</tbody>
</table>

• Repair observations
  • 36 of the 58 mutations are the result of DeleteStatement and DeleteLine operations
  • code semantics are retained in only two cases, most of the semantics are retained in six cases, and the semantics undergo a major change in the remaining 50 cases
  • almost all fixing mutations remove the offending code (thereby changing the semantics)
  • PMD should still be reporting the performance-related issue AvoidArrayLoops in two cases
Implications/Threats

- It appears like `DeleteStatement` and `DeleteLine` mutations result in fewer syntactic code anomalies than the other operations.

- GIN’s fixes tend to come at the expense of changes in code semantics, thus necessitating deeper contextual probing of repair outcomes.

- Removing offending code can be an effective program repair strategy.

- PMD parsing seems at times to be confused by GIN’s mutations, pointing to the need to improve the AST pipeline.

- False negatives may be as detrimental as false positive in invalidating static analysis techniques.

**Threat:** Under normal operation, GIN may strive for code correctness by repeated patch generation given the outcomes of test cases, which was replaced by our manual analysis.
Dissecting Copy/Delete/Replace/Swap mutations: Insights from a GIN Case Study

Sherlock A. Licorish
University of Otago
Dunedin, New Zealand
sherlock.licorish@otago.ac.nz

Markus Wagner*
The University of Adelaide
Adelaide, Australia
markus.wagner@adelaide.edu.au