Can genetic improvement enhance online code snippets?

Sherlock A. Licorish
University of Otago
Dunedin, New Zealand
sherlock.licorish@otago.ac.nz
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 Savarinimethu
Department of Information Science
University of Otago
Dunedin, New Zealand
tony.savarinimethu@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). However, many developers are under the impression that these are not suitable for implementation in production systems due to concerns of 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 spirit of stack overflow is coders helping coders”
- NissanConnect EV mobile app
Impact of stack overflow code snippets on software cohesion: a preliminary study

Mashal Ahmad  
Lero, School of Computer Science  
University College Dublin  
Dublin, Ireland.  
mashal.ahmad@ucdconnect.ie

Mel Ó Cinnéide  
Lero, School of Computer Science  
University College Dublin  
Dublin, Ireland  
mel.ocinneide@ucd.ie

Abstract—Developers frequently copy code snippets from publicly-available resources such as Stack Overflow (SO). While this may lead to a ‘quick fix’ for a development problem, little is understood about the impact of code snippets on software cohesion. To measure the code quality, we used the class cohesion metrics LSCC and CC. Our study involves a sample of 378 GitHub (GH) classes with code snippets copied from Stack Overflow.

But we still use online code snippets!
Current State of Play

Understanding stack overflow code quality: A recommendation of caution

Sarah Meldrum, Sherlock A. Licorish *, Caitlin A. Owen, Bastin Tony Roy Savarimuthu
Department of Information Science, University of Otago, Dunedin, New Zealand

ARTICLE INFO
Article history:
Received 10 January 2020

ABSTRACT
Community Question and Answer (CQA) platforms use the power of online groups to solve problems, or gain information. While these websites host useful information, it is
Current State of Play

Question: Can genetic improvement enhance online code snippets?
Preliminary Agenda

- 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 is used to identify faults, https://pmd.github.io/

- Genetic improvement tool GIN is 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-Threading 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 patched files no longer have any performance issues – according to PMD

• 58 patches (from 44 unique files) 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
Next Steps

• 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.

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

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

Markus Wagner
School of Computer Science, The University of Adelaide
Adelaide, Australia
markus.wagner@adelaide.edu.au

ABSTRACT
Research studies are increasingly critical of publicly available code due to evidence of faults. This has led researchers to explore ways to improve such code, with static analysis and genetic code improvement previously singled out. Previous work has evaluated the feasibility of these techniques, using PMD (a static analysis tool) and GIN (a program repair tool) for enhancing Stack Overflow Java code snippets. Results reported in this regard pointed to the potential of these techniques, especially in terms of GIN’s removal

1 INTRODUCTION/MOTIVATION
On the premise that code-hosting websites such as Stack Overflow¹ and HackerRank² have become the cornerstone for software developers seeking solutions to their coding challenges [4], and because such code can at times possess faults [3, 3, 7], there have been efforts aimed at automating code improvement on such portals [5, 9]. In particular, Licorish and Wagner [5] use the PMD static analysis tool to detect performance faults for a sample of Stack Overflow Java code snippets, before performing mutations on these snippets.
Thanks!

Sarah Meldrum
Caitlin Owen
Tony Savarimuthu
Markus Wagner

The Science for Technological Innovation
National Science Challenge Funding

Commerce Research
Grant Funding

It Will Never Work in Theory
April 2023 Lightning Talks

Thank you!
Genetic improvement enhances online code snippets!

Your thoughts?

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