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

Situation

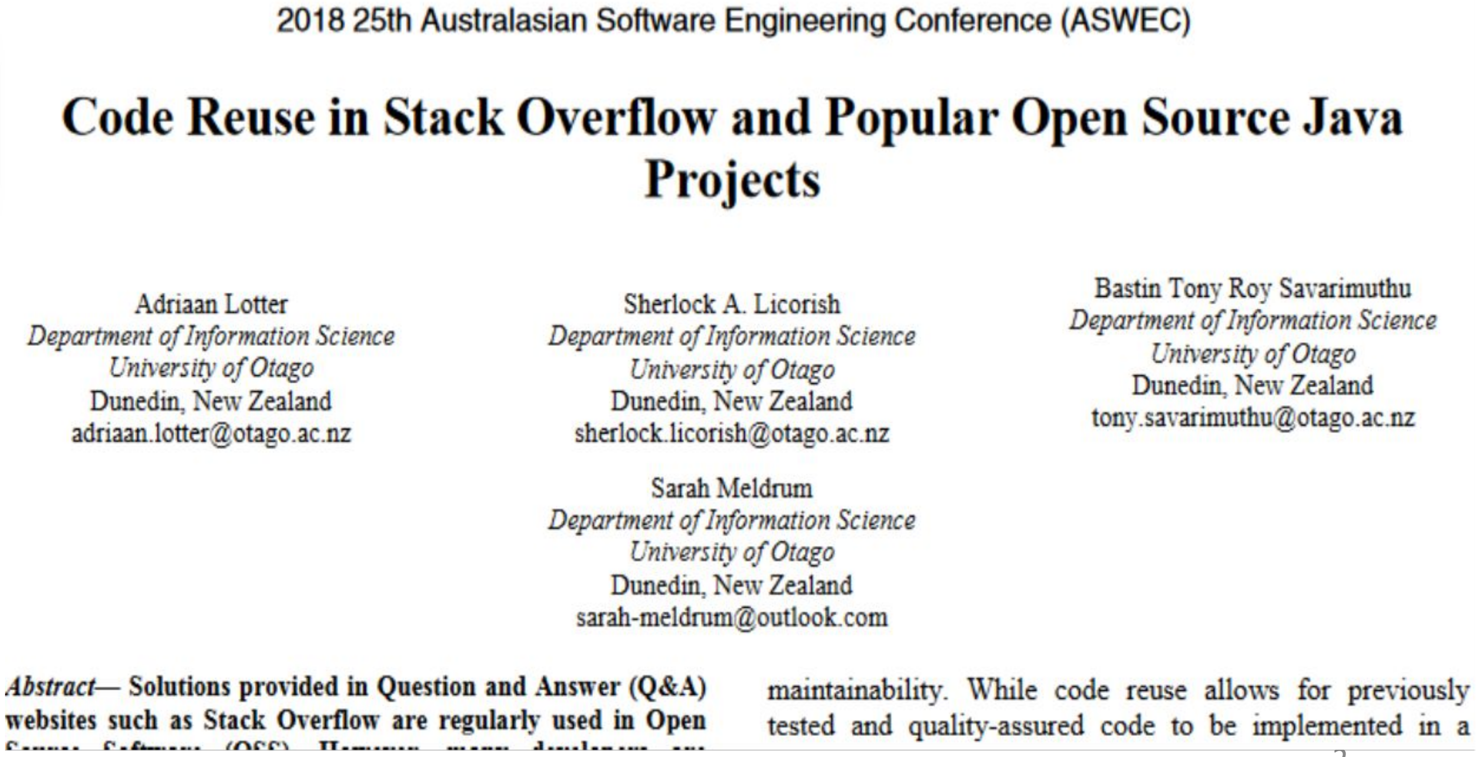
reddit

stackoverflow

CODE PROJECT
For those who code

Quora

superuser



2018 25th Australasian Software Engineering Conference (ASWEC)

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

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

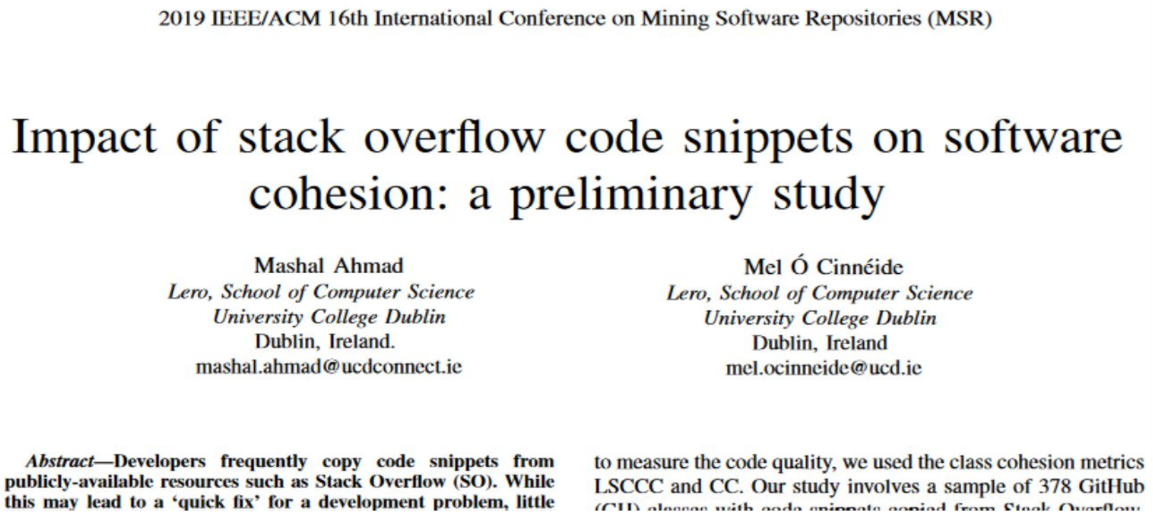
Sarah Meldrum
Department of Information Science
University of Otago
Dunedin, New Zealand
sarah-meldrum@outlook.com

***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**

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



2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)

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**

to measure the code quality, we used the class cohesion metrics LSCCC and CC. Our study involves a sample of 378 GitHub (GH) classes with code snippets copied from Stack Overflow

"App explanation: the sprit of stack overflow is coders helping coders"

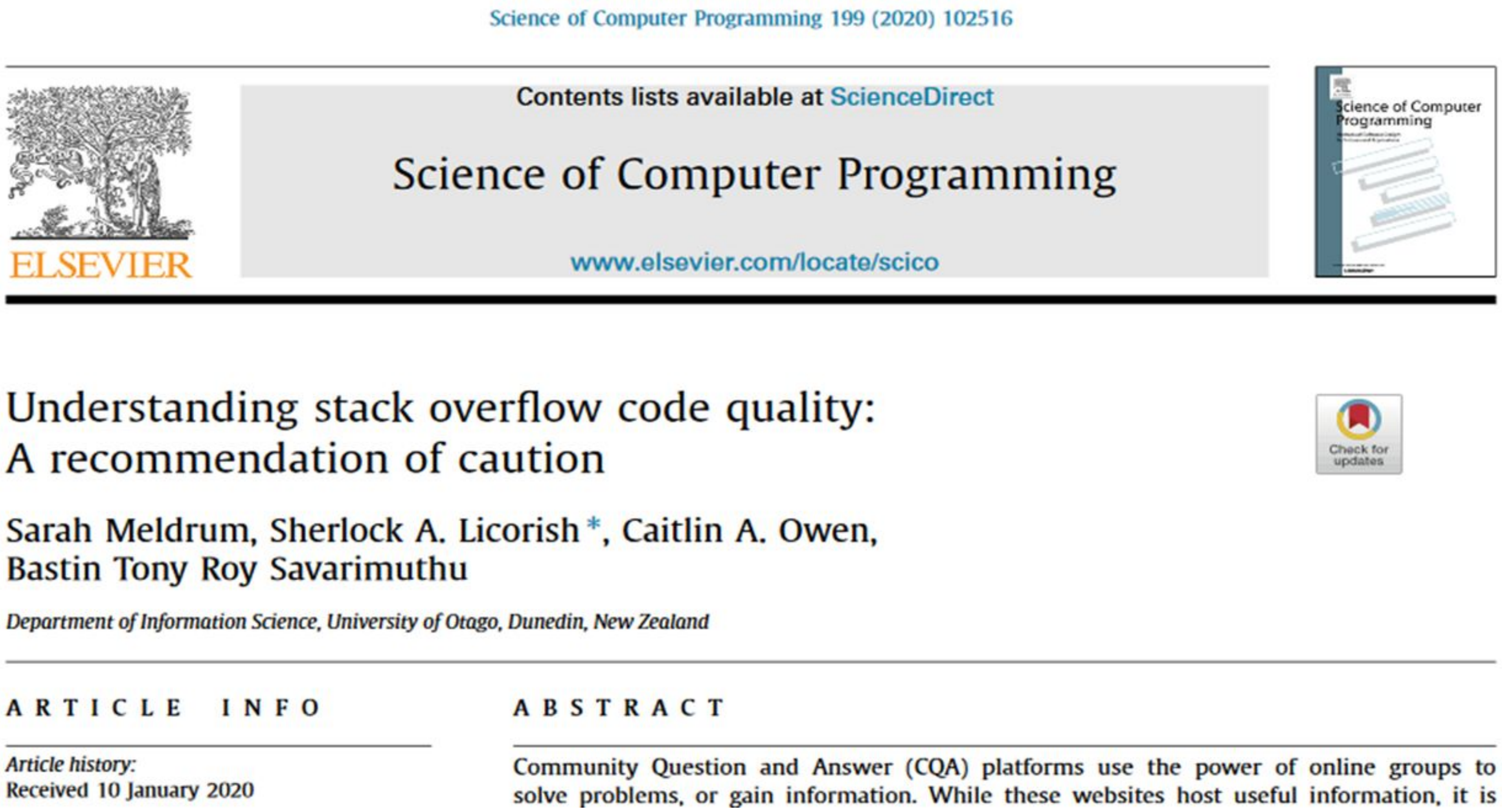
- *NissanConnect EV mobile app*

Phase I

Combining GIN and PMD for Code Improvements

https://arxiv.org/abs/2202.01490

Static Analysis Identifies Flaws



Science of Computer Programming 199 (2020) 102516

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Understanding stack overflow code quality: A recommendation of caution

Sarah Meldrum, Sherlock A. Licorish *, Caitlin A. Owen, Bastin Tony Roy Savarimuthu

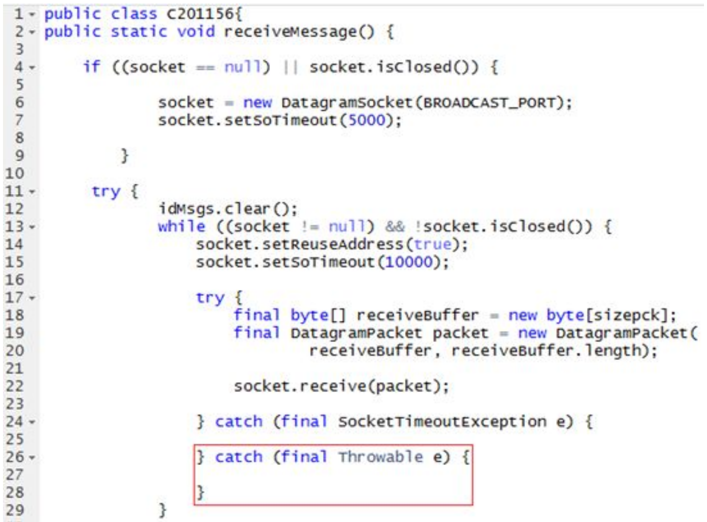
Department of Information Science, University of Otago, Dunedin, New Zealand

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



```
public class C201156{
public static void receiveMessage() {

    if ((socket == null) || socket.isClosed()) {

            socket = new DatagramSocket(BROADCAST_PORT);
            socket.setSoTimeout(5000);

        }

     try {
            idMsgs.clear();
            while ((socket != null) && !socket.isClosed()) {
                socket.setReuseAddress(true);
                socket.setSoTimeout(10000);

                try {
                    final byte[] receiveBuffer = new byte[sizepck];
                    final DatagramPacket packet = new DatagramPacket(
                            receiveBuffer, receiveBuffer.length);

                    socket.receive(packet);

                } catch (final SocketTimeoutException e) {

                } catch (final Throwable e) {

                }
            }
```

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

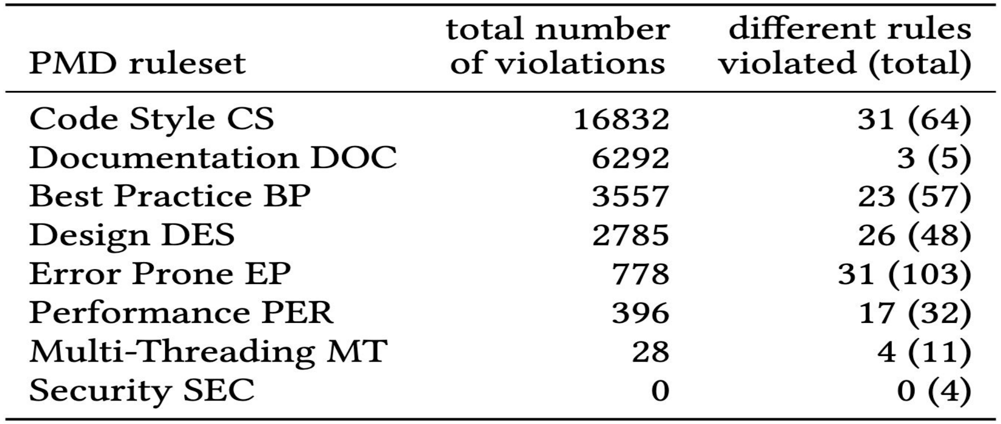
- Static checker PMD used to identify faults, https://pmd.github.io/

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

- We focus on performance related faults in Stack Overflow’s code

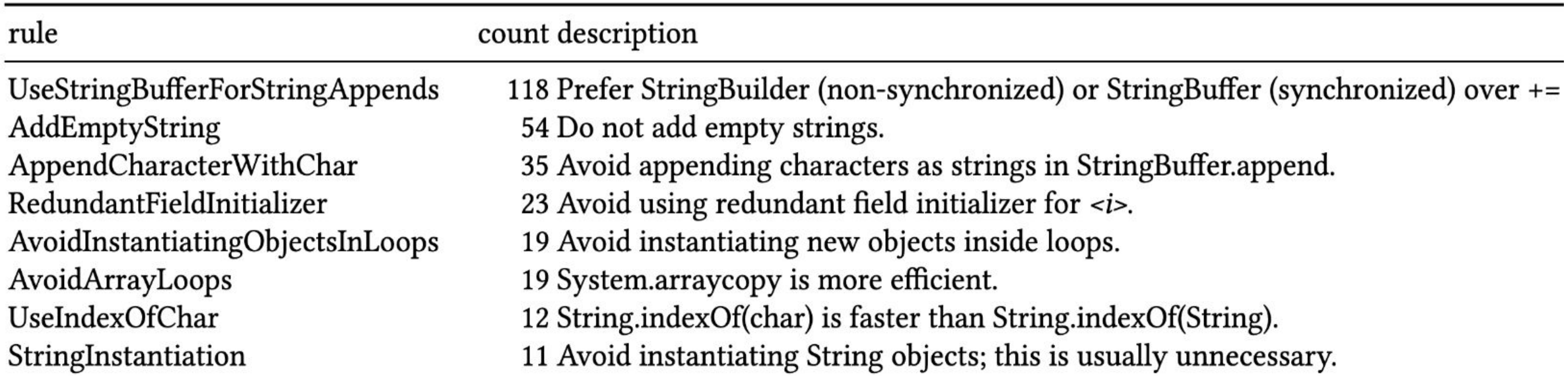
Characterizing PMD’s Treatment of 8,010 Snippets

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



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

- Examples of performance related rule violations:



rule	count	description
UseStringBufferForStringAppends	118	Prefer StringBuilder (non-synchronized) or StringBuffer (synchronized) over +=
AddEmptyString	54	Do not add empty strings.
AppendCharacterWithChar	35	Avoid appending characters as strings in StringBuffer.append.
RedundantFieldInitializer	23	Avoid using redundant field initializer for *<i>*.
AvoidInstantiatingObjectsInLoops	19	Avoid instantiating new objects inside loops.
AvoidArrayLoops	19	System.arraycopy is more efficient.
UseIndexOfChar	12	String.indexOf(char) is faster than String.indexOf(String).
StringInstantiation	11	Avoid instantiating String objects; this is usually unnecessary.

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)

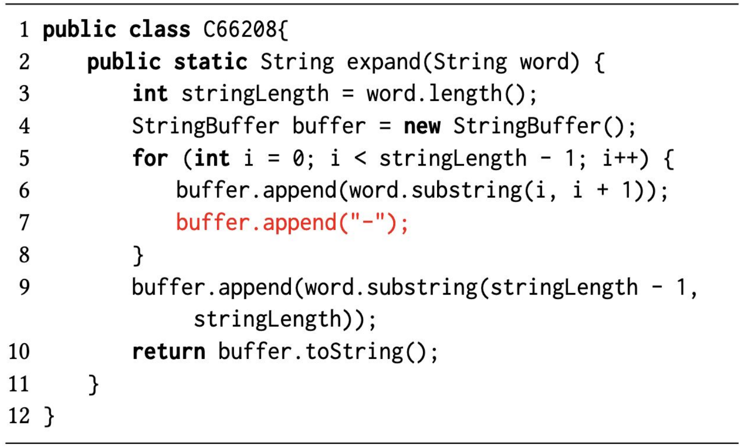
- 770 patches: files no longer have any performance issues – according to PMD

- 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

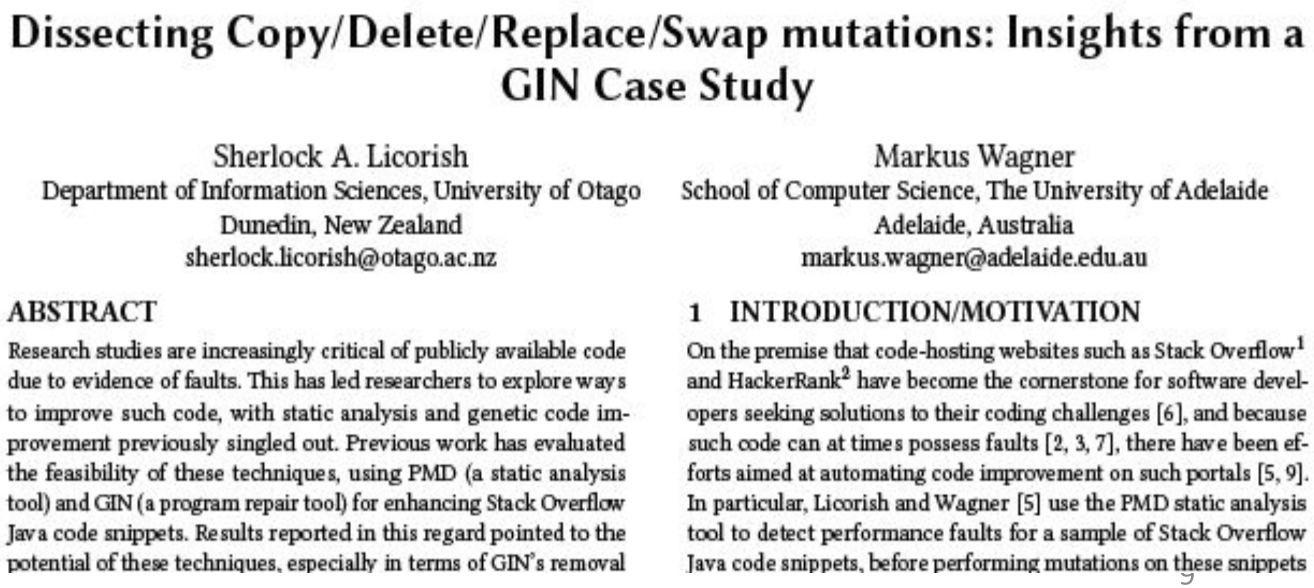
  - Delete edits perform best against the AvoidInstantiatingObjectsInLoops violations

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.



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

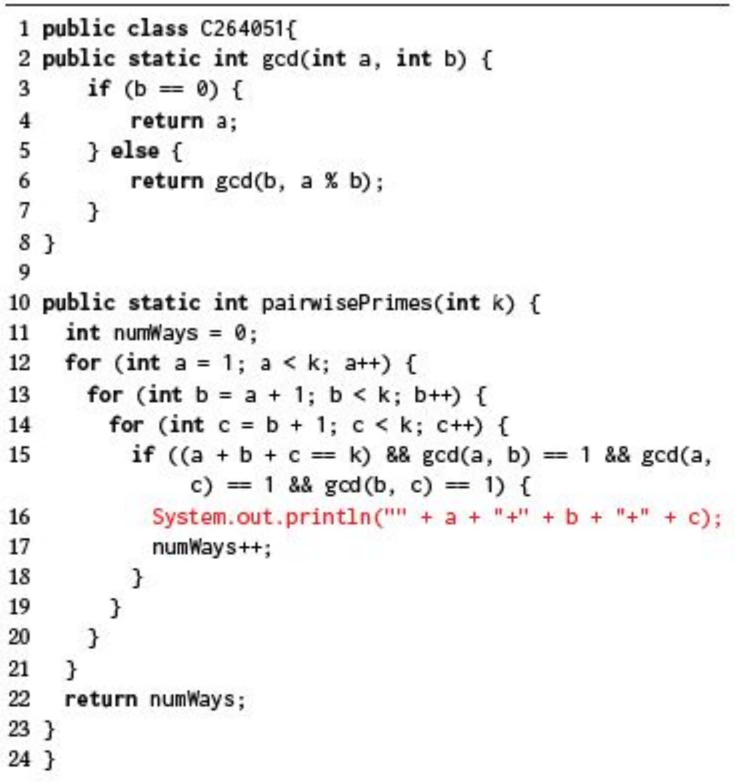
On the premise that code-hosting websites such as Stack Overflow[1] and HackerRank[2] have become the cornerstone for software developers seeking solutions to their coding challenges [6], and because such code can at times possess faults [2, 3, 7], there have been efforts aimed at automating code improvement 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

Phase II

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

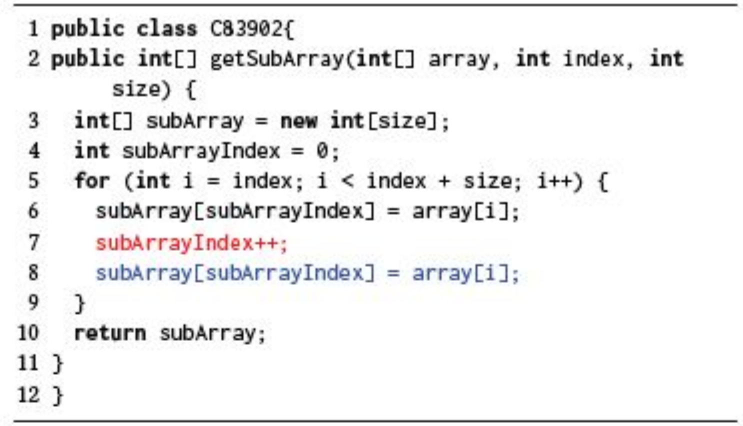
https://cs.adelaide.edu.au/~markus/pub/2022gi-cdrw.pdf

Effectiveness of GIN’s Mutations



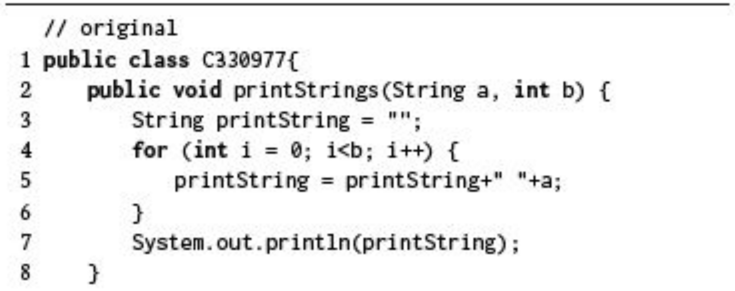
```
1  public class C264051{
2  public static int gcd(int a, int b) {
3      if (b == 0) {
4          return a;
5      } else {
6          return gcd(b, a % b);
7      }
8  }
9
10 public static int pairwisePrimes(int k) {
11   int numWays = 0;
12   for (int a = 1; a < k; a++) {
13     for (int b = a + 1; b < k; b++) {
14       for (int c = b + 1; c < k; c++) {
15         if ((a + b + c == k) && gcd(a, b) == 1 && gcd(a,
                c) == 1 && gcd(b, c) == 1) {
16           System.out.println("" + a + "+" + b + "+" + c);
17           numWays++;
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 C83902{
2  public int[] getSubArray(int[] array, int index, int
        size) {
3   int[] subArray = new int[size];
4   int subArrayIndex = 0;
5   for (int i = index; i < index + size; i++) {
6     subArray[subArrayIndex] = array[i];
7     subArrayIndex++;
8     subArray[subArrayIndex] = array[i];
9   }
10  return subArray;
11 }
12 }
```

Listing 7: Code snippet C83902 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 C330977{
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:

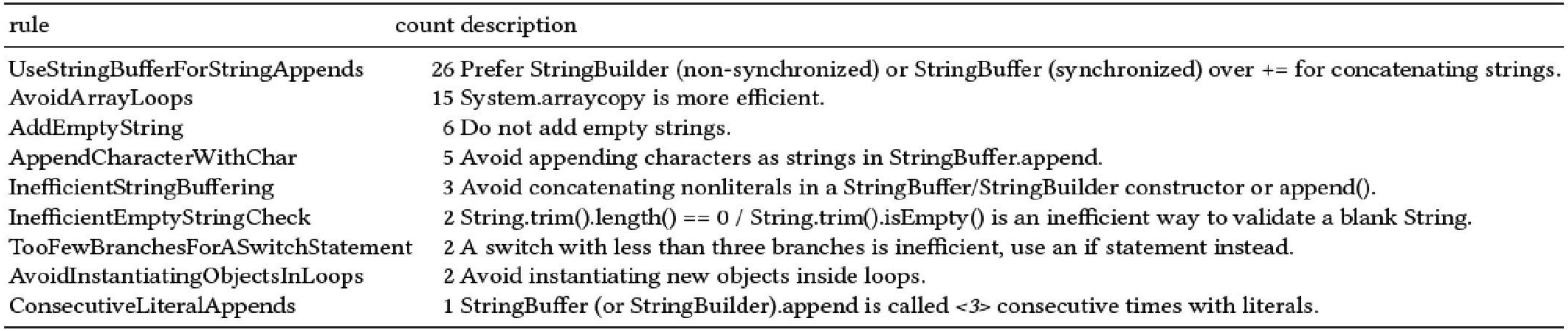
  (1) Describing the change to the semantics of the program

  (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



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

- 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