Identifying clones in comments

Nataliia Stulova

Software Composition Seminar
24.11.2020

UNIVERSITÄT BERN

Code clones

Code clones are fragments of source code that are identical or similar.

Copy-paste programming is a common anti-pattern in software engineering.

```
int array_a[], array_b[];
...

int sum_a = 0;
for (int i = 0; i < array_a.size(); i++)
   sum_a += array_a[i];
int average_a = sum_a / ;

int sum_b = 0;
for (int i = 0; i < array_b.size(); i++)
   sum_b += array_b[i];
int average_b = sum_b / 4;
```

Code clone types [1/4]

Exact clones: identical code fragments which may have some variations in whitespace, layout, and comments

```
if (a >= b) {
  c = d + b;
  // Comment1
  d = d + 1;
} else
  c = d - a; //Comment2

if (a>=b) { // Comment1'
  c=d+b;
  d=d+1;
}
else // Comment2'
  c=d-a;
```

Code clone types [2/4]

Renamed/parametrized clones: syntactically equivalent fragments with some variations in identifiers, literals, types, whitespace, layout and comments

```
if (a >= b) {
  c = d + b; // Comment1
  d = d + 1 ;
} else
  c = d - a; //Comment2

if (m >= n) { // Comment1'
  y = x + n;
  x = x + 5; //Comment3
} else
  y = x - m; //Comment2'
```

Code clone types [3/4]

Near miss clones: syntactically similar code with inserted, deleted, or updated statements

```
if (a >= b) {
  c = d + b; // Comment1
  d = d + 1;
} else
  c = d - a; //Comment2

if (a >= b) {
  c = d + b; // Comment1
  e = 1; // This statement is added
  d = d + 1;
} else
  c = d - a; //Comment2
```

Code clone types [4/4]

Semantic clones: semantically equivalent, but syntactically different code

$$n! = n \cdot (n-1) \cdot (n-2) \cdot (n-3) \cdot \cdots \cdot 3 \cdot 2 \cdot 1 \,.$$

```
// factorial of VALUE
int i, j=1;
for (i=1; i<=VALUE; i++)
  j=j*i;

int factorial(int n) {
  if (n == 0) return 1 ;
  else return n * factorial(n-1) ;
}
```

Comment clones VS code clones

- similar definition: fragments of source comments that are identical or similar
- comment clones are more critical for program comprehension than code clones, though both can lead to bugs introduction
- comment clones are harder to classify, as natural language structure is much less restricted

Comment clone detection

RepliComment is a tool that implements a heuristic-based approach to comment clone detection for Java.

It focuses on **method-level comment clones** within one source file (~single class in most cases).

RepliComment: Identifying Clones in Code Comments

Arianna Blasi · Alessandra Gorla

Università della Svizzera italiana (USI), Lugano, Switzerland

IMDEA Software Institute, Madrid, Spain

ABSTRACT

Code comments are the primary means to document implementation and ease program comprehension. Thus, their quality should be a primary concern to improve program maintenance. While a lot of effort has been dedicated to detect bad smell in code, little work focuses on comments. In this paper we start working in this direction by detecting clones in comments. Our initial investigation shows that even well known projects have several comment clones, and just as clones are bad smell in code, they may be for comments. A manual analysis of the clones we identified revealed several issues in real Java projects.

KEYWORDS

Code comments, Software quality, Clones, Bad smell

ACM Reference Format:

Arianna Blasi · Alessandra Gorla. 2018. RepliComment: Identifying Clones in Code Comments. In *Proceedings of ICPC '18: 26th IEEE/ACM International Conference on Program Comprehension , Gothenburg, Sweden, May 27–28, 2018 (ICPC '18),* 4 pages.
https://doi.org/10.1145/3196321.3196360

1 INTRODUCTION

It is standard practice for developers to document their projects by means of informal documentation in natural language. The Javadoc markup language, for instance, is the de-facto standard to document procedures and classes in Java projects. Similar semi-structured languages are available for other programming languages. Given that many projects have code comments as the only documentation to ease program comprehension, their quality should be of primary concern to improve code maintenance. The quality of code comments is important also because there are many techniques that use comments to automate software engineering tasks, such as generating test cases and synthesizing code [3, 12, 13, 15]. Without comments of high quality, the effectiveness of these techniques cannot be guaranteed.

Our research roadmap is to develop techniques to support developers in *identifying and fixing issues that affect the quality of comments*. As a starting point of our research, we aim to identify and report *comment clones*. Our main hypothesis is that clones in comments may be the result of bad practice, and just as clones in code, they should be identified and fixed.

Comment clones can highlight different issues: They may be instances of copy and paste errors, and therefore comments may not match their corresponding implementation. Otherwise, they may simply provide poor information, which may not be useful to understand the semantic of the implementation. Although these cases are not issues per se, our analysis shows that most of the times these clones point to documentation that could be improved.

Corazza et al. conducted a manual assessment on the coherence between the comments and the implementation, and found instances of comment clones [2]. Similarly, Arnaoudova et al. [1] found some comment clones in their study about Linguistic Antipatterns in software. 93% of interviewed developers considered such issues as a poor/very poor practice. These studies show that the comment clone problem exists and it is relevant for developers.

In this paper we present RepliComment, a technique that automatically identifies comment clones, which may be symptoms of issues that developers want to fix. We used RepliComment to analyze the code base of 10 well-established Java projects. Our preliminary evaluation highlights that even solid and well known projects contain comment clones, and several of them should be analyzed and fixed by developers to improve the quality of documentation.

The remainder of this paper is structured as follows: Section 2 presents some real examples of comment clones, which may identify issues, or may be legitimate cases. Section 3 describes RepliComment, our technique and corresponding prototype to identify comment clones. Section 4 presents the results of our preliminary evaluation. Section 5 discusses some related work, and Section 6 concludes and discusses the future research direction of this work.

2 COMMENT CLONES

We found that there exist very different types of comment clones. In this section we try to highlight the most common scenarios. The comment clones we are mostly interested in are the ones that are misleading and do not match the implementation they document.

One example of this type of clone is the following, which we found in the Google guava project in release 19:[1]

```
1 /**
2  * @return true if this matcher matches every character in the
3  * sequence, including when the sequence is empty.
```

[1] http://google.github.io/guava/releases/19.0/api/docs/com/google/common/base/CharMatcher.html#matchesNoneOf(java.lang.CharSequence)

ICPC '18, May 27–28, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5714-2/18/05…$15.00
https://doi.org/10.1145/3196321.3196360

Comment clone types [1/3]

Copy-paste comment clones: a comment is copied from a correctly documented method and pasted by mistake to another one, whose functionality differs completely.

Google Guava 19

```
public class CharMatcher {

/**
 * @return true if this matcher matches every character in the
 * sequence, including when the sequence is empty.
 */
public boolean matchesAllOf(CharSequence sequence) {...}

/**
 * @return true if this matcher matches every character in the
 * sequence, including when the sequence is empty.
 */
 public boolean matchesNoneOf(CharSequence sequence) {...}
```

Comment clone types [2/3]

Poor information clones: uninformative comments that are so generic that they can be used for different methods.

Apache Hadoop 2.6.5

```
private class UserGroupInformation {

/**
 * @return true or false
 */
@InterfaceAudience.Public
@InterfaceStability.Evolving
public synchronized static boolean isLoginKeytabBased() throws IOException {...}

/**
 * @return true or false
 */
public static boolean isLoginTicketBased() throws IOException {...}
```

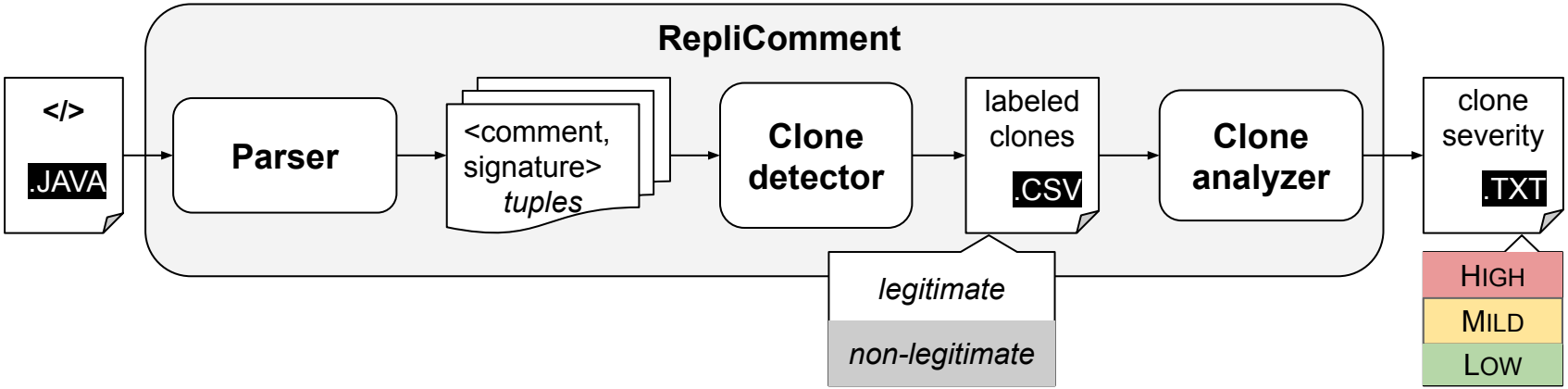
Comment clone types [3/3]

Legitimate clones: comments for methods that offer same so very similar functionality (e.g., polymorphic methods, overriding, common parameters,...).

Apache solr 7.1.0

```
public class SolrClient {

/**
 * Deletes a single document by unique ID
 * @param collection the Solr collection to delete the document from
 * @param id the ID of the document to delete
 */
public UpdateResponse deleteById(String collection, String id) {...}

/**
 * Deletes a single document by unique ID
 * @param id the ID of the document to delete
 */
public UpdateResponse deleteById(String id) {...}
```

RepliComment clone detection and severity classification



copy-paste	non-legitimate	high, mild, or low severity
poor information		
false positives		
legitimate		-

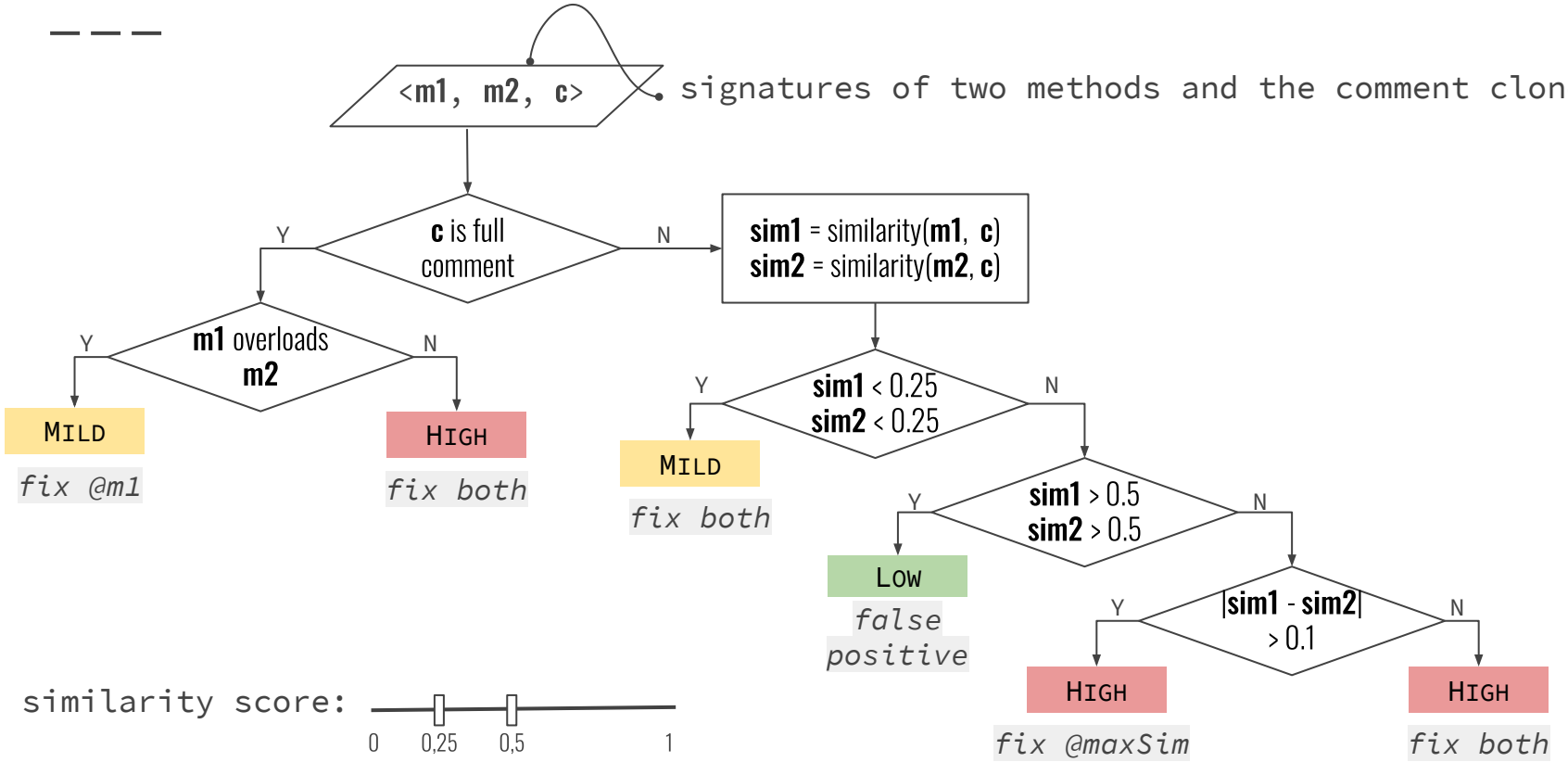
Filtering out legitimate clones

First, whole comment clones are never considered legitimate.

Second, comment parts are legitimate clones if they:

- are cloned for methods with same names
- document same exception type (in 4+ words)
- describe same parameter
- explain return value of the same non-primitive type
  (common in methods that update the class instance and return it, documented as “@return a reference to this”)

Comment clone severity analysis



Method-comment similarity

stems

Step 1: extract *natural language cues* from both code and comments and store as a bag-of-words

Step 2: measure similarity between two bags-of-words

Word Mover's Distance

		WMD similarity
comment	`[1:check, 1:one, 2:graph, 1:unsupport, 1:type, 1:throw, 1:illeg, 1:argument, 1:except ]`	
code v.0	`[1:void, 1:assert, 1:unsupport, 4:graph, 1:type, 1:throw, 1:illeg, 1:argument, 1:except, 1:first,1:second]`	70%
code v.1	`[1:void, 1:assert, 1:unsupport, 2:graph, 1:type, 1:throw, 1:illeg, 1:argument, 1:except, 1:g,]`	66% (**-4%**)

Example analysis message

<method1, method2, comment>

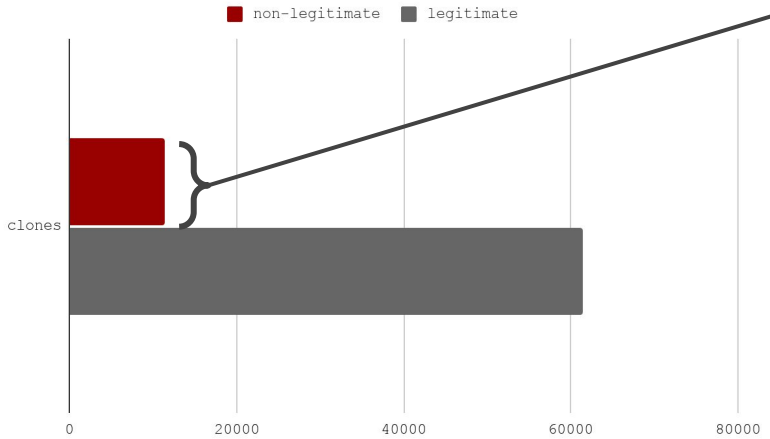
```
---- Record #53 file:2020_JavadocClones_log4j.csv ----
In class: org.apache.log4j.lf5.LogRecord
1) The comment you cloned:"(@return)The LogLevel of this record."
seems more related to <LogLevel getLevel()> than <Throwable
getThrown()>
It is strongly advised to document method <Throwable getThrown()> with
a different, appropriate comment.

---- Record #152 file:2020_JavadocClones_hadoop-hdfs.csv ----
In class: org.apache.hadoop.hdfs.util.LightWeightLinkedSet
1) The comment you cloned:"(@return)first element"
seems more related to <T pollFirst()> than <List pollN(int n)>
It is strongly advised to document method <List pollN(int n)> with
a different, appropriate comment.
```

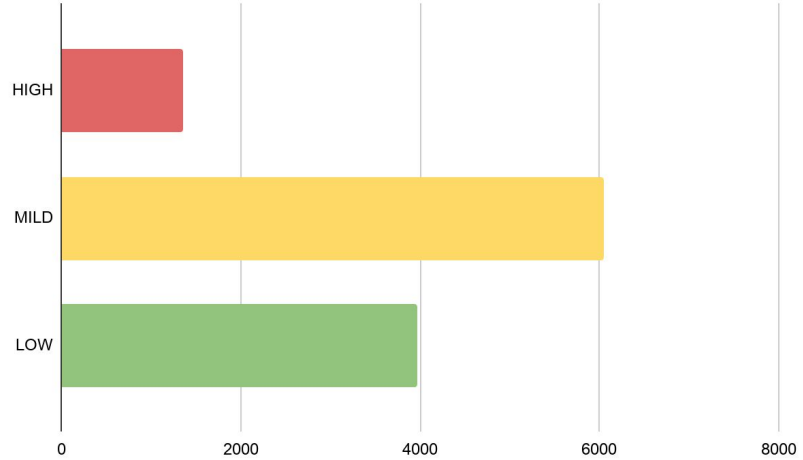
Evaluation

project	classes	LOC	GitHub★
elasticsearch-6.1.1	2906	300K	50K
hadoop-common-2.6.5	1450	180K	11K
vertx-core-3.5.0	461	48K	11K
spring-core-5.0.2	413	36K	38K
hadoop-hdfs-2.6.5	1319	262K	11K
log4j-1.2.17	213	21K	718
guava-19.0	469	70K	38K
rxjava-1.3.5	339	35K	43K
lucene-core-7.2.1	825	103K	4K
solr-7.1.0	501	50K	4K

Effectiveness of comment clone detection

Total: 11K non-legitimate and **61K** legitimate comment clones



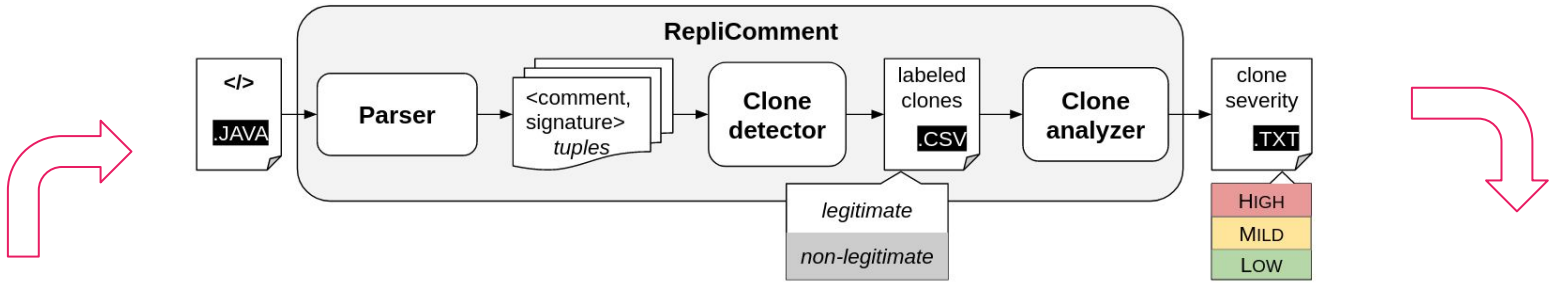


(non-legitimate clones by severity level)

Accuracy of comment clone classification

- **False positives**: we manually verify random 225 (~2%) samples of comment clone cases with HIGH, *MILD* and LOW severity level
  - **~ 21% (or 1 in 5 is a legitimate comment clone)**
- **False negatives**: we manually verify 200 random samples of legitimate comment clones (20 for each project)
  - only **1 case in which we disagreed** with the tool

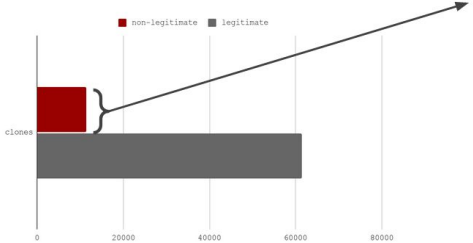
Identifying clones in comments: summary

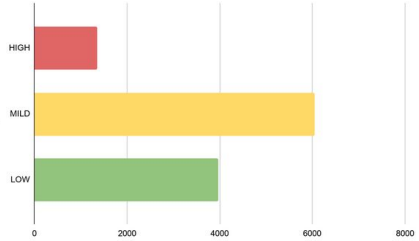


Google Guava 19

```java
public class CharMatcher {

/**
 * @return true if this matcher matches every character in the
 * sequence, including when the sequence is empty.
 */
public boolean matchesAllOf(CharSequence sequence) {...}

/**
 * @return true if this matcher matches every character in the
 * sequence, including when the sequence is empty.
 */
 public boolean matchesNoneOf(CharSequence sequence) {...}
```

Total: 11K non-legitimate and **61K** legitimate comment clones





(non-legitimate clones by severity level)