Are We Reusing Outdated Code from Stack Overflow?

[1]Chaiyong Ragkhitwetsagul, [1]Jens Krinke, [2]Giuseppe Bianco
[1]University College London, London, UK
[2]Università degli Studi del Molise, Campobasso, Italy

ABSTRACT

This paper provides a sample of a LaTeX document which conforms, somewhat loosely, to the formatting guidelines for ACM SIG Proceedings. It is an *alternate* style which produces a *tighter-looking* paper and was designed in response to concerns expressed, by authors, over page-budgets. It complements the document *Author's (Alternate) Guide to Preparing ACM SIG Proceedings Using LaTeX2$_\epsilon$ and BibTeX*. This source file has been written with the intention of being compiled under LaTeX2$_\epsilon$ and BibTeX.

The developers have tried to include every imaginable sort of "bells and whistles", such as a subtitle, footnotes on title, subtitle and authors, as well as in the text, and every optional component (e.g. Acknowledgments, Additional Authors, Appendices), not to mention examples of equations, theorems, tables and figures.

To make best use of this sample document, run it through LaTeX and BibTeX, and compare this source code with the printed output produced by the dvi file. A compiled PDF version is available on the web page to help you with the 'look and feel'.

1. INTRODUCTION

Stack Overflow is a popular online programming community with 6.3 million users. It allows programmers to ask questions and give answers to programming problems. The website has found to be useful for software development and also valuable for educational purposes [?]. On Stack Overflow, each conversation contains a question and answer(s). The answers normally contain at least one code snippet as a solution to the question asked. The code snippet is usually not written directly on Stack Overflow website but copied from another location. It can be copied and modified from the problematic code snippet in the question, copied from an answerer's own code, or borrowed from other locations including open source software (OSS) systems. As a result, the process of posting and answering questions on Stack Overflow which involves copying and pasting source code can be considered as code cloning.

MSR '17 May 20–21, 2017, Buenos Aires, Argentina
© 2016 ACM. ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

Code cloning is an activity of reusing source code by copying and pasting. It normally occurs in software development and account from 7% to 23% in typical software systems [3]. The benefits and drawbacks of clones are still controversial. Several authors state that clones lead to bug propagations and software maintenance issues [13], while some others have proofs that in some cases clones are not harmful than normal code or even beneficial [25, 14]. Code cloning can also have side effects of violating software licenses. Carelessly cloning code from one project and reusing it in another project with different license may cause software licensing violation [9].

In this study, we treat code snippets that are copied from software systems to Stack Overflow, and vice versa, as code clones. We call them **online code clones**. There are three ways to create online code clones: 1) code is cloned from a software project to Stack Overflow as an example; 2) code is cloned from Stack Overflow to a software project to obtain a functionality, perform a particular task, or fixing a bug; and 3) code is implicitly cloned from one software project to another by having Stack Overflow as a medium. Online code clones can similarly lead to a problem of bug propagation as classical code clones. Unfortunately, they are more difficult to locate and fix since the search space from online corpora is larger and no longer confined in a local repository.

A motivating example of problems caused by online code clones can be found in a Stack Overflow post regarding how to implement RawComparator in Hadoop[1]. In Figure 1, the left hand side shows a code snippet embedded as a part of accepted answer to the question. The snippet shows how Hadoop implements *compare* method in its *WritableComparator* class. The code snippet on the right hand side shows another version of the same *compare* method in *WritableComparator* class but it is extracted from the latest version of Hadoop. We can obviously see that they are highly similar except one line, `buffer.reset(null,0,0);`, added in the latest version after `key2.readFields(buffer);`. The added line is intended for cleaning up the reference in `buffer` variable. While this change has already been introduced into *compare* method in the latest version of Hadoop, the code example in Stack Overflow post is still unchanged and outdated. This example shows that there can be inconsistencies between online code clones and its original. This is an emerging and challenging problem. Since studies in this area are still limited, we aim to gain more insight of the problem in this study.

[1]http://stackoverflow.com/questions/22262310

```
/* Code in Stack Overflow #22315734 */
public int compare (byte[] b1,int s1,int l1,
                    byte[] b2,int s2,int l2) {
  try {
    buffer.reset(b1,s1,l1); /* parse key1 */
    key1.readFields(buffer);
    buffer.reset(b2,s2,l2); /* parse key2 */
    key2.readFields(buffer);
  } catch (IOException e) {
    throw new RuntimeException(e);
  }
  return compare(key1,key2); /* compare them */
}
```

```
/* WritableComparator.java (2016-09-26) */
public int compare(byte[] b1,int s1,int l1,
                   byte[] b2,int s2,int l2) {
  try {
    buffer.reset(b1,s1,l1); /* parse key1 */
    key1.readFields(buffer);
    buffer.reset(b2,s2,l2); /* parse key2 */
    key2.readFields(buffer);
    buffer.reset(null,0,0); /* clean up reference */
  } catch (IOException e) {
    throw new RuntimeException(e);
  }
  return compare(key1, key2); /* compare them */
}
```

Figure 1: The same code fragments, WritableComparator.java, on Stack Overflow post 22315734 and latest version in hadoop code base

This paper makes the following primary contributions:

1. A manual study of online code clones: We used two clone detection tools to discover 266,837,480 similar code fragment pairs and manually investigated 7,840 candidate clone pairs between Java code fragments obtained from Stack Overflow accepted answers and 109 Java open source projects.

2. Addressing the problems of reusing source code between open source projects and Stack Overflow: Our study shows that there are at least 238 clones that have been obviously copied from open source projects or external online sources to Stack Overflow as code examples which potentially violate their software licenses. Furthermore, 50 out of the 84 clones are outdated and questionable for being reused.

2. EMPIRICAL STUDY

We perform an empirical study of online code clones between Stack Overflow and 109 Java open source projects to answer the following research questions:

RQ1 (online code clones): *To what extent source code is cloned between Stack Overflow and open source projects?* We would like to quantitatively measure the number of online code clones between Stack Overflow and open source projects to understand the scale of the problem.

RQ2 (classification of online code clones): *What are the main characteristics among the set of online code clones?* We group them into seven groups according to our pre-defined classification scheme so we can differentiate and understand the motivation of cloning. Some of the clones are copy from open source projects to Stack Overflow, while some are copied from a third-party location, and some are accidental clones containing boiler-plate code, and code stubs generated by IDE.

RQ3 (effects of online code clones): *what are the effects derived from online code clones? can they be harmful to software development?* Is there observable problems caused by clones between Stack Overflow and open source projects?

2.1 Experimental Framework

To answer the three research questions, an experimental framework is designed as depicted in Figure 2. We process two datasets, Stack Overflow and open source projects from Qualitas corpus. Java code fragments are extracted from Stack Overflow posts using regular expressions. We preprocess Java code in both datasets by removing comments and pretty-printing to increase accuracy of clone detection. Then, we deploy two clone detection tools, Simian [1] and NiCad [24, 5], to locate clones between the two datasets. Due to a technical limit of Simian and NiCad to scale to large datasets, we partition the input and run the tools multiple times. Each run is composed of the whole Stack Overflow data and a single Qualitas project. We repeat the process until we cover 109 projects.

We then convert the clone reports to General Clone Format (GCF) [29] and combine them into a single file. GCF provides a common format for clones which enable us to reuse scripts that analyse clone reports from Simian and NiCad. Moreover, using GCF, other additional clone detectors can be adopted, if needed, without any changes in the analysis. Simian do not provide an option to detect inter clones between two locations. Hence the Simian GCF clone report is pruned to contain only inter clone pairs between Stack Overflow and Qualitas project. In this step, all intra clone pairs within Stack Overflow and open source projects are removed. NiCad provides an option to detect inter clones so no pruning is needed. Next, clone pairs reported from the two clone detectors are pair-wise matched to find agreements using Bellon's clone overlapping criteria [3]. This step generates **agreement-based clone pairs**. They are clones with highest confidence since they receive agreement from both tools. Then, clone pairs reported by Simian and NiCad that do not find agreement are filtered by size of minimum 10 lines. This step generates **non agreement-based clone pairs**. The non agreement-based clone pairs are clones with less confidence than agreement-based ones. Finally, agreement-based and non agreement-based clone pairs are looked at and classified manually by the first author.

In the manual inspection process, we classify clones into categories according to their properties. This process takes approximately a months until we successfully classified 7,840 clone pairs into categories. Some of the clone candidates are false clones due to being boiler-plate code or IDE-generated and are discarded for further analysis. By ignoring the false clones, we compare licensing information of 477 remaining clone pairs for possibility of software licensing violations. Moreover, we look forward through history of the clones from the projects' git versioning systems. This is to see if there is any changes made to the clones after it has been copied, hence resulting in outdated clones on Stack Overflow.

2.2 Experimental Setup

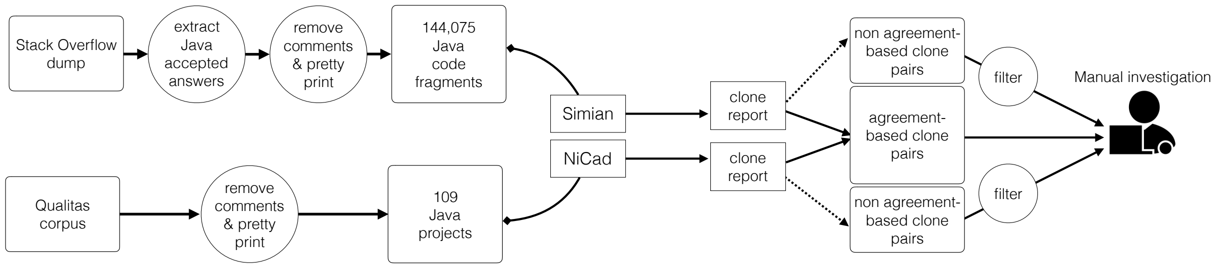
Stack Overflow dump
extract Java accepted answers
remove comments & pretty print
144,075 Java code fragments
Qualitas corpus
remove comments & pretty print
109 Java projects
Simian
NiCad
clone report
clone report
non agreement-based clone pairs
agreement-based clone pairs
non agreement-based clone pairs
filter
filter
Manual investigation

Figure 2: Experimental framework

Table 1: Stack Overflow and Qualitas datasets

Dataset	No. of files	SLOC
Stack Overflow	144,075	2,347,269
Qualitas	138,238	16,081,899

2.2.1 Datasets

Stack Overflow: we extracted Java code snippets from accepted answers in a snapshot of Stack Overflow dump [2] in January 2016. The archived dump has a size of 9 gigabytes. The data dump is in XML format containing information of *Posts* (questions and answers) and supporting data such as user accounts and timestamps of the posts. We are interested in code snippets embedded in posts which are pieces of code located between `<code></code>` tags. We filtered the snippets with two filtering criteria. First, we ignore snippets that are less than 6 lines since they are usually not considered as clones by clone detection research [**?**]. Second, we are only interested in code snippets from posts that are marked accepted answer since they have high chances to be reused than snippets in questions and other answers. Each snippet is extracted from the dump using regular expressions and saved to a file using its post ID as the file's name. We use `.java` extension so that the clone detectors can recognise them. If a Stack Overflow conversation has more than one code snippet in the accepted answer, we append an indexing number starting from zero after the post ID (e.g. 45051109_0.java, and 45051109_1.java). With the two filters, we finally obtained 144,075 Java code snippets which contain xxx lines of Java source code excluding comments and blank lines[3].

Open source systems: we selected an established corpus for empirical software engineering studies called **Qualitas** [28]. **FIXME: more details about Qualitas and why we chose it**. We selected the 20130901r version of Qualitas corpus containing 112 Java open source projects with releases no later than 1st September 2013. We chose the snapshot late back in 2013 since we are interested in online code clones in the direction from open source projects to Stack Overflow. The 20130901r snapshot provides Java code that is at least 3 years old which is sufficient for a number of code snippets to be copied onto Stack Overflow. Out of 112 Qualitas projects, there are 3 projects, *eclipse*, *jre*, and *squirrel-sql*, that do not contain any Java source code file so they are removed from the study. This results in totally 109 projects analysed in the study. As shown in Table 1, the 109 Qualitas project have 138,238 Java files containing 16,081,899 lines of code.

2.2.2 Clone Detectors

There are a number of restrictions in terms of choosing clone detection tools for this study. Firstly, they have to compatible with Java. Secondly, due to nature of code snippets posted on Stack Overflow, some of them are not complete Java classes or methods. Hence, the tool must be flexible enough to process code snippets which are not in a complete block or not compilable. Thirdly, since the amount of code that have to be processed are in a scale of millions line of code (as shown in Table 1), a clone detector must be scalable enough to successfully complete the execution and report clones in a reasonable amount of time. We have tried running 5 state-of-the-arts clone detectors including Simian [1], NiCad [5, 24], CCFinder [13], iClones [11], and DECKARD [12] against the Stack Overflow and Qualitas datasets. CCFinder, iClones, and DECKARD failed to successfully detect clones between 144,075 Stack Overflow code snippets and 109 Qualitas projects. All of them reported execution errors after running for couple of hours. Thus, we removed them from the study. Simian and NiCad completed the detection with success. We found that both of them are flexible enough to handle million-SLOC code corpus with method or class incompleteness. So, we decided to use both of them.

Simian is a text-based clone detector which locate clones at line-level granularity and has been used extensively in several clone studies [22, 29, 18, 4, 17]. It is a command-line tool which enables us to automate the detection. It also offers normalisation of variable names and literals (strings, and numbers) which enable Simian to detect clones of type 1 and type 2. **NiCad** is also a text-based clone detector which detects clones at either method- or block-level granularity. It can detect clone up to type 3 and is used in several clone studies [24, 22, 27, 29, 18, 26]. It utilises TXL for parsing and pretty-printing source code. It also provide code normalisation by variable renaming and code abstraction. We use a variant of NiCad called *nicadcross*. It offers the same functionalities as the original NiCad but is specialised for detecting code clones between two systems. NiCad is also a command-line tool which makes it suitable for automation.

2.2.3 Agreement-based Clone Detection

The number of detected clones between the two datasets can be very large and infeasible for human to manually check all of them. Therefore, we adopted an idea of clone agreement which has been used in clone research studies [29, 8, 23] in a situation that clone oracle is missing or impossible

[2]https://archive.org/details/stackexchange
[3]measured by cloc: https://github.com/AlDanial/cloc

to establish. Clone pairs agreed by multiple clone detection tools have higher confident to be real clones [23]. By using this agreement-based clone detection approach, we can reduce the number of clone candidates for manual investigation by paying more attention to the agreed clone pairs. To find agreement of two clone pairs, we resort to an approach proposed by a study by Bellon et al. [3]. Two clone pairs which have overlapping clone lines can be considered as a good-match or an ok-match pair. A good-match clone pair has stronger agreement than an ok-match pair. We follow the same following definitions introduced in the original paper:

A clone pair CP is formed by two clone fragments CF_1 and CF_2 with a pre-defined similarity threshold t: $CP = (CF_1, CF_2, t)$. We can define *overlap* and *contained* value of two clone pairs as

$$overlap(CP_1, CP_2) = \frac{|lines(CF_1) \cap lines(CF_2)|}{|lines(CF_1) \cup lines(CF_2)|} \quad (1)$$

$$contained(CP_1, CP_2) = \frac{|lines(CF_1) \cap lines(CF_2)|}{|lines(CF_1)|}. \quad (2)$$

good-value of two clone pairs is then defined as

$$\begin{aligned} good(CP_1, CP_2) = min(&overlap(CP_1.CF_1, CP_2.CF_1), \\ &overlap(CP_1.CF_2, CP_2.CF_2)). \end{aligned}$$

Furthermore, *ok-value* is defined as

$$\begin{aligned} ok(CP_1, CP_2) = min(&max(contained(CP_1.CF_1, CP_2.CF_1), \\ &contained(CP_2.CF_1, CP_1.CF_1)), \\ &max(contained(CP_1.CF_2, CP_2.CF_2), \\ &contained(CP_2.CF_2, CP_1.CF_2))). \end{aligned}$$

Two clone pairs CP_1 and CP_2 are called a *good-match(p)* iff, for $p \in [0, 1]$ holds

$$good(CP_1, CP_2) \geq p. \quad (3)$$

Similarly for an *ok-match(p)* pair

$$ok(CP_1, CP_2) \geq p. \quad (4)$$

Using this good-match and ok-match criteria with a pre-defined threshold p, we prune the detected clone pairs for manual investigation. good-match pairs are the ones with the highest confident to be true clones, followed by ok-match pairs, and followed by clone pairs without agreement.

2.3 Clone detectors' parameter tuning

We are aware of effects of configurations to clone detection results and needs for searching for optimised configurations in empirical clone studies [29, 23, 22, 27]. However, considering the size of the two datasets and search space of at least 15 of Simian's and 5 of NiCad's parameters, we are hindered from finding the best configurations of the tools. Thus, we decided to configure Simian and NiCad using two established settings: 1) the tools' default settings chosen by the tools' creators (denoted as *df*), and 2) the general settings for Java sets from *EvaClone*, a study of optimising clone detectors' configurations based on clone agreement, by Wang et al. [29] (denoted by *EvCl*). The details of the two configurations are described in Table 2. With our two tools and these two sets of configurations, we look for agreements in four possible pairwise combinations: Simian$_{df}$-NiCad$_{df}$, Simian$_{df}$-NiCad$_{EvCl}$, Simian$_{EvCl}$-NiCad$_{df}$, and Simian$_{EvCl}$-NiCad$_{EvCl}$.

Table 2: Configurations of Simian and NiCad

Tool	Parameters
Simian$_{df}$	threshold=6, ignoreStringCase, ignoreCharacterCase, ignoreModifiers
Simian$_{EvCl}$	threshold=5, ignoreIdentifiers, ignoreIdentifierCase, ignoreStrings, ignoreCharacters, ignoreSubtypeNames, balanceSquareBrackets
NiCad$_{df}$	MinLine=10, MaxLine=1000, UPI=0.30
NiCad$_{EvCl}$	MinLine=5, MaxLine=604, UPI=0.20, blind renaming, literal abstraction

Table 3: Statistics of clones found between Stack Overflow and Qualitas projects using Simian and NiCad

Stats	Simian$_{df}$	Simian$_{EvCl}$	NiCad$_{df}$	NiCad$_{EvCl}$
Snippets	1,406	1,360	1,197	12,884
Total C$_{pairs}$	54,790	59,936,722	127,305	206,718,663
Avg. C$_{pairs}$	52	44,104	106	16,046
Avg. C$_{size}$	7.45	4.81	9.54	5.31
Avg. C$_{\%}$	28%	29%	25%	21%

3. RESULTS

The clone statistics obtained from running Simian and NiCad with *df* and *EvCl* settings are presented in Table 3. Preliminary manual investigation of Simian's clone report showed that there were problematic 11 fragments. These 11 fragments trigger Simian to generate large clone clusters containing a huge number of false clones of array initialisation. Hence, they were removed from Simian's clone reports before the analysis. From Table 3, Simian clones are from approximately 10% of the 144,075 Stack Overflow snippets, 1,406 reported by Simain$_{df}$ and 1,360 from Simain$_{EvCl}$ respectively. NiCad$_{df}$ reports clones from 1,197 Stack Overflow code snippets while NiCad$_{EvCl}$ reports clones from a large amount of 12,884 snippets due to its relaxed settings. In terms of number of clone pairs, Simian$_{EvCl}$ and NiCad$_{EvCl}$ report a large number of clone pairs of 59,936,722 and 206,718,663 respectively. This is expected since EvaClone configurations prefer recall [29]. The average clone size of Simian$_{df}$ is 7.45 which is bigger than its Simian$_{EvCl}$ counterpart of 4.81. Similarly, NiCad$_{df}$ has an average clone size of 9.54 which is bigger than 5.31 reported by NiCad$_{EvCl}$. We can see from the statistics that EvaClone tunes the tools in the way that they report smaller clones but result in higher agreements. The average percentage of code in Stack Overflow snippets that are clones reported by Simian$_{df}$, Simian$_{EvCl}$, NiCad$_{df}$, and NiCad$_{EvCl}$ is 28%, 29%, 25% and 21% accordingly.

3.1 Agreement based clone pairs

We use a threshold p of 0.7 for both good and ok-match. A visualisation of good-match clone pairs between four combinations of *df* and *EvCl* settings can be seen from Figure 3. We found that NiCad throws renaming and clustering errors while processing some Qualitas projects as listed in

Table 4: No. of projects in Qualitas successfully analysed by Simian and NiCad

	Simian$_{df}$	Simian$_{EvCl}$	NiCad$_{df}$	NiCad$_{EvCl}$
Successful	109	109	97	82
Clust. fail	–	–	6	16
Renm. fail	–	–	–	11

Table 5: 13 Qualitas projects that have been cloned as examples on Stack Overflow with their respective licenses

Project	Version	Licenses (no. of files)
aspectj	1.6.9	Apache-1.1 (182), CPLv1 (3), EPLv1 (2011), None (23), SeeFile (6), Unknown (286)
hadoop	1.0.0	Apache-2 (1,935), spxdBSD/Apache-2 (8) None (33), Unknown (14)
hibernate	4.2.2	Apache-2 (20), GLGPLv2.1+ (31) PublicDomain (1), None (1,850), SeeFile (4), Unknown (4,324)
jasperreports	3.7.4	GLGPLv2.1+ (3), GLGPLv3+ (1,581), None (1), Unknown (4)
jfreechart	1.0.13	GLGPLv2.1+ (989)
jgraph	5.13.0.0	GLGPLv2.1+ (4), spdxBSD (2), Unknown (151), None (24), SeeFile (9)
jstock	1.0.7c	GLGPLv2.1+ (1), GPLv2+ (239), Apache-2 (1), BSD3 (1), None (23), SeeFile (1), spdxMIT (3), Unknown (5)
jung2	2_0_1	N/A
junit	4.11	None (160), Unknown (4)
poi	3.6	Apache-2 (2,002), None (5)
spring	3.0.5	Apache-2 (2,982)
struts2	2.2.1-all	Apache-2 (1,717), spdxBSD3 (6), None (118), SeeFile (1), Unknown (2)
tomcat	7.0.2	Apache-2 (1,313), None (11)

Table 4. NiCad$_{df}$, produced clustering errors for 6 projects. NiCad$_{EvCl}$ produced renaming error for 4 projects, and clustering error for 13 projects. As a result, we did not have NiCad clone reports for these projects and hence they are missing from agreement-based clone pair calculation. **FIXME: Report the errors to NiCad creator.** There are 2,261 unique good-match pairs consisting of 10 pairs from Simian$_{df}$-NiCad$_{df}$, 26 pairs from Simian$_{df}$-NiCad$_{EvCl}$, 10 pairs from Simian$_{EvCl}$-NiCad$_{df}$, and 2,228 pairs from Simian$_{EvCl}$-NiCad$_{EvCl}$. Figure 4 show the distribution of 23,868 unique ok-match pairs, which subsume the good-match pairs. There are 3,800 pairs from Simian$_{df}$-NiCad$_{df}$, 1,017 pairs from Simian$_{df}$-NiCad$_{EvCl}$, 88 pairs from Simian$_{EvCl}$-NiCad$_{df}$, and 19,777 pairs from Simian$_{EvCl}$-NiCad$_{EvCl}$. Between the four sets, there are a few clone pairs that are common between two adjacent sets, but there is no clone pair that is agreed by all four combinations.

Table 6: Distribution of agreement-based clone pairs using Bellon's criteria for *df* and *EvCl* settings

Tool		No. of clone pairs	
Simian	NiCad	good-match	ok-match
df	*df*	10	3800
df	*EvCl*	26	1017
EvCl	*df*	10	88
EvCl	*EvCl*	2228	19777
Total		2274	24682
Total (unique)		2261	23868

3.1.1 *Manual investigation of agreement-based clone pairs*

The classification scheme is described in Table 7 and the classification results are shown in Table 8. We have manually investigated all of the 2,261 good-match ones reported by agreement of four different Simian and NiCad settings. However, for the ok-match, we could not investigate all of the 23,868 pairs manually. According to the manual investigation of good-match results, we found that Simian$_{EvCl}$–NiCad$_{EvCl}$ produces a large number, 2225 (99.87%), of false positive results (category D, E, and F). Thus, we decided to leave them out of the manual investigation of ok-match pairs. There are totally 608 ok-match pairs that were investigated. The 39 true positive pairs found are combinations of 8 unique Stack Overflow fragments, and 9 unique Qualitas Java files from 6 different projects.

Since we are not certain about the direction of copying in the B-classified pairs, we checked the modification time of each Java file in Qualitas project and compare it to the timestamp of Stack Overflow answers. We found that all Stack Overflow code fragments were posted after their respectively similar Java files in Qualitas project. This means that the copying can only be either (1) from Qualitas to Stack Overflow or (2) from a third source to both Stack Overflow and Qualitas independently.

3.2 Non-agreement based clone pairs

In the preliminary stage of our experiment, we found that there are 41 Stack Overflow fragments reported by Simian with default configurations. However, only 10 of them appear in the new results using tool's agreement. Thus, we further investigated the clone pairs reported by Simian and NiCad but *without* an agreement.

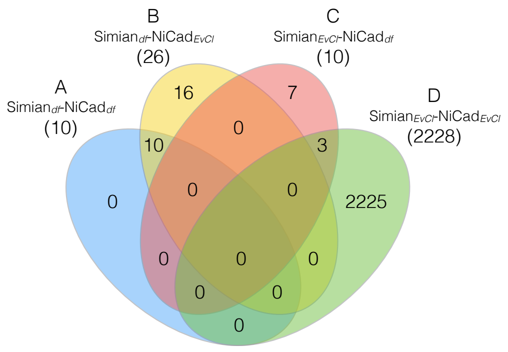

Figure 3: Distributions of good-match(0.7) pairs

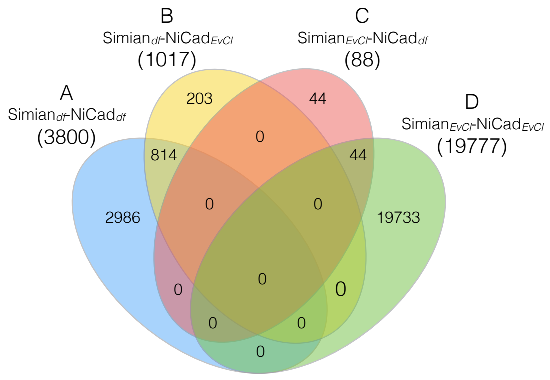

Figure 4: Distributions of ok-match(0.7) pairs

Table 7: Classifications of clone creation

Category	Descriptions
A	Code in Stack Overflow is copied from Qualitas (Q → S).
A'	Code in Qualitas is copied from Stack Overflow (S → Q).
B	Clone pair is exactly identical or highly similar and may be copied either from each other or a third source (unknown) (S ↔ Q ∨ (T → S ∧ T → Q)).
C	Code in both places are copied from a third source T (known) (T → S ∧ T → Q).
D	Code is a boiler-plate or IDE auto-generated.
E	Code in both places initialise a similar/the same object; extend the same class/its subclass; implement the same interface.
F	Accidental similarity, false clone

Table 8: Classification results of agreement-based and non agreement-based clone pairs)

Classificaiton	A	A'	B	C	**Sum**	S_u	Q_u	Q_{up}	D	E	F	**Sum**	S_u	Q_u	Q_{up}	**Total**	S_u	Q_u	Q_{up}
good-match(0.7)	1	0	4	3	**8**	7	6	6	58	6	2189	**2253**	81	693	58	**2261**	87	699	59
ok-match(0.7)	8	0	33	8	**49**	12	14	10	4459	35	82	**4576**	99	136	29	**4625**	110	150	33
Simian$_{df}$	71	0	318	11	**401**	138	210	37	182	64	148	**394**	120	191	45	**795**	246	386	55
NiCad$_{df}$	4	0	15	0	**19**	15	13	9	30	6	104	**140**	51	55	20	**159**	66	68	23

Table 9: Statistics of non agreement-based clone pairs (Simian$_{df}$ and NiCad$_{df}$).

Tool	Clone pairs	ok-pairs	filtered	remaining
Simian$_{df}$	18422	140	17490	792
NiCad$_{df}$	29227	378	28690	159

With our 4 settings, we decided to investigate only 2 settings, Simian$_{df}$, and NiCad$_{df}$, and drop Simian$_{EvCl}$ and NiCad$_{EvCl}$ due to their large number of false positives. With the 2 selected settings, we investigated clone pairs having the minimum clone size of 10 SLOC as they are meaningful and tend to be real clone in modern clone detection [26].

For Simian$_{df}$, there were 9,383 clone pairs reported by the tool. Out of 9,383 pairs, 140 of them are the ones found in *ok*-pairs using agreement-based detection. We filtered the results further by removing false positives such as similar *equals()*, *hashCode()* methods, getters and setters out by using regular expression. We managed to remove 8,956 pairs using this method. Eventually, there were 287 clone pairs remaining for manual investigation. For NiCad$_{df}$, we obtained 7,040 clone pairs to look at which is infeasible for manual investigation. Hence, result filtering was also needed. However, regular expressions could not be used effectively as in Simian's case since NiCad allowed clones that are different at keywords/variable names or even added/deleted lines. So we decided to filter the results by selecting pairs that pass stricter clone criteria with UPI = 0.2. By reducing the UPI to 0.2, there were totally 166 pairs left. Out of 166, 52 are ok-pairs and 114 are remaining pairs for manual check (18 pairs are from *cayenne* and *iReport* that could not be analysed using UPI = 0.3). The statistics of the clones and classification results are reported in Table 9.

3.2.1 *Manual investigation of non agreement-based clone pairs*

We performed manual investigation and classified the clone pairs reported by Simian$_{df}$ and NiCad$_{df}$ in the same way as the agreement-based clone pairs. The results of the manual investigation is reported in Table 8.

Table 10: Numbers of true positive online clone pairs found by manual investigation

Tool	A	A'	B	C	Total
good-pairs	1	0	4	3	8
ok-pairs	8	0	33	8	49
Simian$_{df}$ pairs	71	0	318	10	399
NiCad$_{df}$ pairs	4	0	15	0	19
Total	84	0	370	21	476

4. EFFECTS OF ONLINE CODE CLONE

In this study, we are interested in the effects of online code clones to software development. From the manual investigation of 184 true online clone pairs, we found that there are two potential issues: stale online code, and software licensing violation.

4.1 Issue 1 – stale online code clones

Stale online code occurs when a piece of code has been copied from a software project to Stack Overflow, and later it has been changed in the original project. However, in this

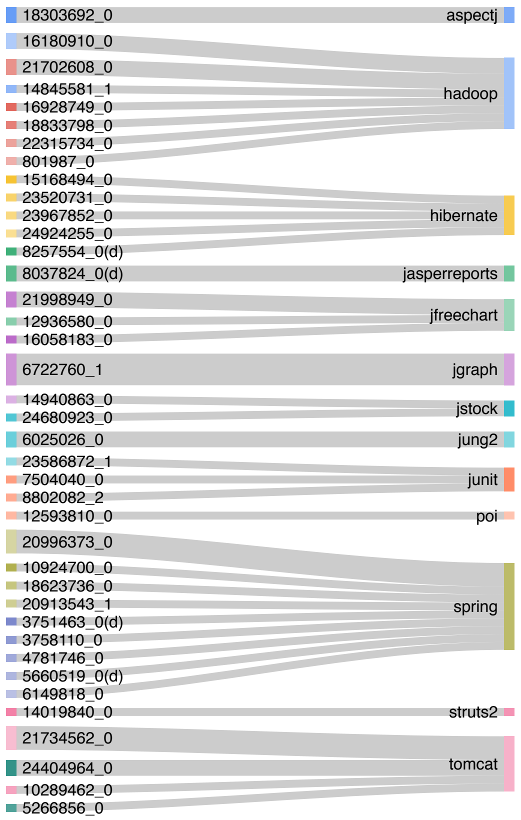

Figure 5: Relationships of 53 Stack Overflow clone pairs to their original projects. 49 are outdated and 4 are deleted (shown using (d) suffix).

Table 11: Results from a manual investigation of 84 category-A online clone pairs

Project	Pairs	Stale	Fresh	Del.	Others
ant	1	0	1	0	0
log4j	4	0	4	0	0
tomcat	7	7	0	0	0
aspectj	2	2	0	0	0
hadoop	14	9	5	0	0
hibernate	16	4	11	1	0
jasperreports	2	2	0	0	0
jfreechart	4	4	0	0	0
jgraph	5	4	0	0	1
jgrapht	1	0	1	0	0
jstock	2	2	0	0	0
jung	2	2	0	0	0
junit	3	3	0	0	0
poi	3	1	2	0	0
spring	14	9	3	2	0
struts	1	1	0	0	0
weka	3	0	3	0	0
Total	84	50	30	3	1

situation, the copy is still unchanged. Since the code were updated due to various possible reasons including bug fixing, this can cause a problem if developers reuse stale online code from Stack Overflow. They might also introduce the same unfixed bug(s) into the software. To discover stale online code, we focus on the true online clone pairs that are copied in the direction from Qualitas to Stack Overflow (category-*A* online clone pairs) in Table 10 which results in 84 pairs selected.

Table 11 shows the results of manual investigation of 84 category-A online clone pairs. The investigation reveals that there are 50 clone pairs that are outdate (i.e. "stale clone"). They are clone pairs that were copied from Qualitas projects to Stack Overflow and marked as accepted answers, and later have been changed during the development.

4.2 Issue 2 – software licensing violations

Software licensing is a paramount factor in software development. Violation of software license can cause a major impact to the delivery of the software and also lead to legal issues. It is an emerging area that software engineering research community is paying attention to. For example, there are studies of automatic technique to identify software licensing from source code files [10] and the evolution of licenses in open source projects [6].

In our study, we reveal another possible situation of software licensing issue caused by code cloning to Stack Overflow. We found that there are at least 84 pieces of code have been copied from 13 open source projects in Qualitas dataset to Stack Overflow as examples. They are also marked as accepted answers which increase their probability of being reused. These 13 open source projects come with their respective software licenses. However, the licensing information are mostly missing from these clones when they are posted on Stack Overflow. Mostly one or a few methods from the full source file are cloned. This makes the license information at the top of the file mostly left uncopied. If developers copy and reuse these pieces of code in their projects, a licensing conflict can quietly happen without realisation of the developers.

5. THREATS TO VALIDITY

6. RELATED WORK

- Code clones
  - Definition: Baxter et al. [2]
  - Comparison of clone detectors: [24, 22, 27]
  - NiCad [24, 5]
  - Simian [1]
  - Clone taxonomy [15]
  - Clone evolution [21, 18]
  - Comparing Quality Metrics for Cloned and non cloned Java Methods : A Large Scale Empirical Study [25].
- Agreement-based Clone Detection
  - Bellon's framework [3].
  - EvaClone [29]
  - Hybrid [8]
- Software licensing
  - Code siblings [9], Ninka – Automatic indification of SW license [10], Evolution of SW licensing [6]
- Stack Overflow
  - Code example [19]
  - Search for code in Stack Overflow [7, 16, 20]

7. CONCLUSIONS

This paragraph will end the body of this sample document. Remember that you might still have Acknowledgments or Appendices; brief samples of these follow. There is still the Bibliography to deal with; and we will make a disclaimer about that here: with the exception of the reference to the LaTeX book, the citations in this paper are to articles which have nothing to do with the present subject and are used as examples only.

8. ACKNOWLEDGMENTS

This section is optional; it is a location for you to acknowledge grants, funding, editing assistance and what have you. In the present case, for example, the authors would like to thank Gerald Murray of ACM for his help in codifying this *Author's Guide* and the **.cls** and **.tex** files that it describes.

9. REFERENCES

[1] Simian. http://www.harukizaemon.com/simian. Accessed: 07.04.2016.

[2] I. D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier. Clone detection using abstract syntax trees. In *ICSM'98*, pages 368–377, 1998.

[3] S. Bellon, R. Koschke, G. Antoniol, J. Krinke, and E. Merlo. Comparison and evaluation of clone detection tools. *IEEE Transactions on Software Engineering*, 33(9):577–591, 2007.

[4] W. T. Cheung, S. Ryu, and S. Kim. Development nature matters: An empirical study of code clones in JavaScript applications. *Empirical Software Engineering*, pages 517–564, 2015.

Table 12: 53 code clones in Stack Overflow that were altered, rewritten, or removed from the project after posted and their respective licenses

No.	Project	File	License	SO Post	Changes	Date
1	aspectj-1.6.9	Agent.java	–	18303692	alteration	2015-09-08
2	aspectj-1.6.9	Agent.java	–	18303692	alteration	2015-09-08
3	hadoop-1.0.0	DBCountPageView.java	Apache-2	21702608	alteration	2011-06-12
4	hadoop-1.0.0	DBCountPageView.java	Apache-2	21702608	alteration	2011-06-12
5	hadoop-1.0.0	JobSubmissionFiles.java	Apache-2	14845581	alteration	2012-06-25
6	hadoop-1.0.0	LineRecordReader.java	Apache-2	16180910	alteration	2011-07-25
7	hadoop-1.0.0	LineRecordReader.java	Apache-2	16180910	alteration	2011-07-25
8	hadoop-1.0.0	StringUtils.java	Apache-2	801987	alteration	2013-02-04
9	hadoop-1.0.0	TestJobCounters.java	Apache-2	18833798	alteration	2011-06-12
10	hadoop-1.0.0	TextOutputFormat.java	Apache-2	16928749	alteration	2011-06-12
11	hadoop-1.0.0	WritableComparator.java	Apache-2	22315734	alteration	2014-11-20
12	hibernate-4.2.2	ConnectionProviderInitiator.java	–	15168494	alteration	2012-06-24
13	hibernate-4.2.2	Example.java	–	24924255	alteration	2013-04-23
14	hibernate-4.2.2	SchemaUpdate.java	–	23520731	alteration	2016-02-05
15	hibernate-4.2.2	SettingsFactory.java	–	8257554	removal	2011-03-11
16	hibernate-4.2.2	SQLServer2005LimitHandler.java	–	23967852	alteration	2015-03-12
17	jasperreports-3.7.4	JRVerifier.java	GLGPLv3+	8037824	alteration	2008-04-17
18	jasperreports-3.7.4	JRVerifier.java	GLGPLv3+	8037824	alteration	2011-05-20
19	jfreechart-1.0.13	AbstractXYItemRenderer.java	GLGPLv2.1+	12936580	alteration	2016-02-19
20	jfreechart-1.0.13	KeyToGroupMap.java	GLGPLv2.1+	16058183	alteration	2013-07-03
21	jfreechart-1.0.13	SpiderWebPlot.java	GLGPLv2.1+	21998949	alteration	2008-06-02
22	jfreechart-1.0.13	SpiderWebPlot.java	GLGPLv2.1+	21998949	alteration	2008-06-02
23	jgraph-5.13.0.0	HelloWorld.java	GLGPLv2.1+	6722760	rewriting	2014-04-13
24	jgraph-5.13.0.0	HelloWorld.java	GLGPLv2.1+	6722760	rewriting	2014-04-13
25	jgraph-5.13.0.0	HelloWorld.java	GLGPLv2.1+	6722760	rewriting	2014-04-13
26	jgraph-5.13.0.0	HelloWorld.java	GLGPLv2.1+	6722760	rewriting	2014-04-13
27	jstock-1.0.7c	GoogleMail.java	GPLv2+	14940863	alteration	2015-12-13
28	jstock-1.0.7c	GoogleMail.java	GPLv2+	24680923	alteration	2015-12-13
29	jung2-2_0_1	ShortestPathDemo.java	–	6025026	alteration	2010-04-13
30	jung2-2_0_1	ShortestPathDemo.java	–	6025026	alteration	2010-04-13
31	junit-4	Assert.java	–	23586872	alteration	2015-05-12
32	junit-4	ExternalResource.java	–	7504040	alteration	2016-06-25
33	junit-4.11	ExpectException.java	–	8802082	alteration	2014-05-26
34	poi-3.6-20091214	WorkbookFactory.java	Apache-2	12593810	alteration	2015-04-29
35	spring-framework-3.0.5	AnnotationMethodHandlerExceptionResolver.java	Apache-2	5660519	removal	2012-01-20
36	spring-framework-3.0.5	AutowireUtils.java	Apache-2	20913543	alteration	2014-10-28
37	spring-framework-3.0.5	CustomCollectionEditor.java	Apache-2	18623736	alteration	2013-11-21
38	spring-framework-3.0.5	DefaultAnnotationHandlerMapping.java	Apache-2	3758110	removal	2012-01-20
39	spring-framework-3.0.5	DefaultPropertiesPersister.java	Apache-2	6149818	alteration	2013-03-19
40	spring-framework-3.0.5	DelegatingServletInputStream.java	Apache-2	20996373	alteration	2016-07-15
41	spring-framework-3.0.5	DelegatingServletInputStream.java	Apache-2	20996373	alteration	2008-12-18
42	spring-framework-3.0.5	DelegatingServletInputStream.java	Apache-2	20996373	alteration	2008-12-18
43	spring-framework-3.0.5	DispatcherServlet.java	Apache-2	4781746	alteration	2011-08-08
44	spring-framework-3.0.5	Jaxb2Marshaller.java	Apache-2	10924700	alteration	2012-08-28
45	spring-framework-3.0.5	ScheduledTasksBeanDefinitionParser.java	Apache-2	3751463	alteration	2016-07-05
46	struts2-2.2.1	DefaultActionMapper.java	Apache-2	14019840	alteration	2013-10-18
47	tomcat-7.0.2	BasicAuthenticator.java	Apache-2	21734562	alteration	2016-08-04
48	tomcat-7.0.2	BasicAuthenticator.java	Apache-2	21734562	alteration	2016-08-04
49	tomcat-7.0.2	CoyoteAdapter.java	Apache-2	24404964	alteration	2012-11-18
50	tomcat-7.0.2	CoyoteAdapter.java	Apache-2	24404964	alteration	2012-11-18
51	tomcat-7.0.2	FormAuthenticator.java	Apache-2	21734562	alteration	2016-08-04
52	tomcat-7.0.2	HttpServlet.java	Apache-2	5266856	alteration	2011-10-22
53	tomcat-7.0.2	JspRuntimeLibrary.java	Apache-2	10289462	alteration	2012-09-12

[5] J. R. Cordy and C. K. Roy. The NiCad Clone Detector. In *ICPC '11 Proceedings of the 2011 IEEE 19th International Conference on Program Comprehension*, pages 3–4, 2008.

[6] M. Di Penta, D. M. German, Y.-G. Guéhéneuc, and G. Antoniol. An exploratory study of the evolution of software licensing. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - ICSE '10*, volume 1, page 145, 2010.

[7] T. Diamantopoulos and A. L. Symeonidis. Employing source code information to improve question-answering in stack overflow. In *MSR '15 Proceedings of the 12th Working Conference on Mining Software Repositories*, pages 454–457, 2015.

[8] M. Funaro, D. Braga, A. Campi, and C. Ghezzi. A hybrid approach (syntactic and textual) to clone detection. In *Proceedings of the 4th International Workshop on Software Clones - IWSC '10*, pages 79–80. ACM Press, 2010.

[9] D. M. German, M. Di Penta, Y.-G. Gueheneuc, and G. Antoniol. Code siblings: Technical and legal implications of copying code between applications. In *2009 6th IEEE International Working Conference on Mining Software Repositories*, pages 81–90, 2009.

[10] D. M. German, Y. Manabe, and K. Inoue. A sentence-matching method for automatic license identification of source code files. In *Proceedings of the IEEE/ACM international conference on Automated software engineering - ASE '10*, page 437, 2010.

[11] N. Göde and R. Koschke. Incremental Clone Detection. In *2009 13th European Conference on Software Maintenance and Reengineering*, pages 219–228, 2009.

[12] L. Jiang, G. Misherghi, Z. Su, and S. Glondu. DECKARD: Scalable and Accurate Tree-Based Detection of Code Clones. In *Proceeding of the 29th International Conference on Software Engineering (ICSE'07)*, pages 96–105, 2007.

[13] T. Kamiya, S. Kusumoto, and K. Inoue. CCFinder: a multilinguistic token-based code clone detection system for large scale source code. *IEEE Transactions on Software Engineering*, 28(7):654–670, 2002.

[14] C. Kapser and M. Godfrey. "Cloning Considered Harmful" Considered Harmful. In *Proceeding of the 13th Working Conference on Reverse Engineering*, pages 19–28, 2006.

[15] C. Kapser and M. W. Godfrey. Toward a taxonomy of clones in source code: A case study. In *Proceedings of the ELISA workshop – Evolution of Large-scale Industrial Software Evolution*, pages 67–78, 2003.

[16] I. Keivanloo, J. Rilling, and Y. Zou. Spotting working code examples. In *Proceedings of the 36th International Conference on Software Engineering - ICSE 2014*, pages 664–675, 2014.

[17] J. Krinke, N. Gold, Y. Jia, and D. Binkley. Cloning and copying between GNOME projects. In *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*, pages 98–101, 2010.

[18] M. Mondal, M. S. Rahman, R. K. Saha, C. K. Roy, J. Krinke, and K. A. Schneider. An Empirical Study of the Impacts of Clones in Software Maintenance. In *2011 IEEE 19th International Conference on Program Comprehension*, pages 242–245, 2011.

[19] S. M. Nasehi, J. Sillito, F. Maurer, and C. Burns. What makes a good code example?: A study of programming Q&A in StackOverflow. In *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, pages 25–34. IEEE, 2012.

[20] J.-w. Park, M.-W. Lee, J.-W. Roh, S.-w. Hwang, and S. Kim. Surfacing code in the dark: an instant clone search approach. *Knowledge and Information Systems*, 41(3):727–759, dec 2014.

[21] J. R. Pate, R. Tairas, and N. A. Kraft. Clone evolution: A systematic review. *Journal of software: Evolution and Process*, 25:261–283, 2013.

[22] C. Ragkhitwetsagul, J. Krinke, and D. Clark. Similarity of Source Code in the Presence of Pervasive Modifications. In *16th IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM'16)*. IEEE, 2016.

[23] C. Ragkhitwetsagul, M. Paixao, M. Adham, S. Busari, J. Krinke, and J. H. Drake. Searching for Configurations in Clone Evaluation A Replication Study. In *8th Symposium on Search-Based Software Engineering (SSBSE'16)*, 2016.

[24] C. K. Roy and J. R. Cordy. An Empirical Study of Function Clones in Open Source Software. In *2008 15th Working Conference on Reverse Engineering*, 2008.

[25] V. Saini, H. Sajnani, and C. Lopes. Comparing Quality Metrics for Cloned and non cloned Java Methods : A Large Scale Empirical Study. In *ICSE '16 Proceedings of the 32th International Conference on Software Maintenance and Evolution*, pages 256–266, 2016.

[26] H. Sajnani, V. Saini, J. Svajlenko, C. K. Roy, and C. V. Lopes. SourcererCC: Scaling Code Clone Detection to Big-Code. In *Proceedings of the 38th International Conference on Software Engineering - ICSE '16*, pages 1157–1168, 2016.

[27] J. Svajlenko, I. Keivanloo, and C. K. Roy. Big data clone detection using classical detectors: an exploratory study. *Journal of Software: Evolution and Process*, 2014.

[28] E. Tempero, C. Anslow, J. Dietrich, T. Han, J. Li, M. Lumpe, H. Melton, and J. Noble. Qualitas corpus: A curated collection of java code for empirical studies. In *2010 Asia Pacific Software Engineering Conference (APSEC2010)*, pages 336–345, Dec. 2010.

[29] T. Wang, M. Harman, Y. Jia, and J. Krinke. Searching for Better Configurations: A Rigorous Approach to Clone Evaluation. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, pages 455–465, 2013.