SemanticCloneBench: A Semantic Code Clone Benchmark Using Crowd-Source Knowledge

1st Farouq Al-omari
Department of Computer Science
University of Saskatchewan
Saskatoon, CA
faa634@mail.usask.ca

2nd Chanchal Roy
Department of Computer Science
University of Saskatchewan
Saskatoon, CA
chanchal.roy@usask.ca

3rd Tonghao Chen
Department of Computer Science
University of Saskatchewan
Saskatoon, CA
tonghao.chen@usask.ca

***Abstract*—Not only do newly proposed code clone detection techniques, but existing techniques and tools also need to be evaluated and compared. This evaluation process could be done by assessing the reported clones manually or by using benchmarks. The main limitations of available benchmarks include: they are restricted to one programming language; they have a limited number of clone pairs that are confined within the selected system(s); they require manual validation. To overcome these limitations, we proposed a methodology to generate a wide range of clone benchmark(s) for different programming languages with minimal human validation. Our technique is based on the knowledge provided by developers who participate in the crowd-sourced information website, Stack Overflow. We applied automatic filtering, selection and validation to the source code in Stack Overflow answers. Finally, we build a semantic code clone of 4000 clones pairs for the languages java, C, C# and Python.**

***Index Terms*—Semantic clone, Functional equivalent, Stack Overflow, Benchmark.**

I. Introduction

Software clones are defined as identical or similar (near-miss) code fragments in terms of syntax or semantic. Usually, syntax clones result from the practice of copying, pasting, and modifying by programmers. The case of having major modifications to these clones will result in their disappearance. On the other hand, some clones are unintentionally introduced to software systems. Sometimes, developers implement the same functionality using different programming approaches or constructs (different syntactic constructs), which results in what is called semantic clones.

Cloning is an essential practice by programmers for the benefits it offers in the development process [17]. Identifying clones is necessary for it's bad impact on software quality for later life cycle of the software. Therefore, there is a great many clone detectors have been proposed to detect both types of clones, syntax [2], [5], [18] and semantic [4], [10], [12], [20]. Practitioners need to know the accuracy of clone detection tool for each type of clones [2], [15], [24]. Thus, these tools need to be evaluated in term of recall and precision.

There are two primary measurements used to evaluate the accuracy of detection tools, precision, and recall. First, validate tool results as true positive and false positive manually. This evaluation can measure the actual precision (results accuracy) of the tool. However, it is time-consuming and dependent on the individual's understanding of code clone definition. Besides, it does not reflect the actual clones detected in the target system (Recall). Measuring Recall of tools is more challenging since there is a need to know all true clones in the target system, i.e., clone benchmark. Building a clone benchmark is a challenging task since it needs to be accurate, large enough, includes all types of clones and represents real clones that occur at the development process.

Available benchmarks have the following issues: First, they are not large enough or not enough references for all types of clones (Both syntax and semantic); for example, Bellon's benchmark [2] does not have type-4 clones, Krutz's benchmark [11] has only 66 clones without any type-1 clones, Yuki's benchmark [29] has 19 unclassified clones and Wanger's benchmark [26] has only 29 type-4 clones. Second, they are the resulted clones of one or more detection tools [2]. That means only a subset of real clones in the systems that are detected by selected tools. Third, they follow the definition of the creator [15], which do not represent real clones generated by developers. Fourth, existing benchmarks supports one or two languages only. For example, BigCloneBench [23] supports Java only and Krutz's benchmark [11] is for C language. Still there is no bench march specialized for semantic clones.

The two primary challenges in building a clone benchmark are candidate selection and manual validation. Candidates should be real clones that are occurred in the development process. Also, selection should be objective and includes all types of clones, i.e, not author's defined clones or some tool's results. Manual validation is a nontrivial process and time consuming [27]. For example, Jeffery et al. [23] spent 600 working hours to validate their benchmark.

In this work, we propose a methodology for building a semantic code clone oracle (functional clone database) with the minimal need for manual validation. Our clone selection process does not relate to any detection tool or clone definition. We used the knowledge in Stack Overflow, a question and answer website for computer programming [28]. We extracted all programming answers that have code snippets. We considered the answers for the same programming question as semantic clones to each other. To reduce the human efforts in validating the clones, our process gone through a number of steps of selection, syntax validation, functional validation, and syntactic clone filtering. Finally, we build a semantic code clone benchmark (SemanticCloneBench) that have 4,000 semantic code clone in four programming languages.

The remaining of the paper is organized as follows. Section II provides the definition of semantic clone. Section III presents our methodology of building semantic clone benchmark. Section IV describe the benchmark and it's usage. In section V we analyse the textual similarly holds by clones in the benchmark. Section VI discusses related works and section VII shows threats to validity, followed by conclusion and future work in section VII.

II. Semantic clone definition

A code fragment in the source code that identical or similar to another code fragment in the code base is considered code clone to the second and both called clone pair. This definition is based on the concept of similarity. In the literature, the following categorization of clone definition has been widely acceptable [16]:

- Type 1: Identical code fragments except for variations in whitespace (maybe also variations in layout) and comments.
- Type 2: Structurally/syntactically identical fragments except for variations in identifiers, literals, types, layout, and comments.
- Type 3: Copied fragments with further modifications. Statements can be changed, added or removed in addition to variations in identifiers, literals, types, layout, and comments.
- Type 4: Two or more code fragments that perform the same computation but are implemented through different syntactic variants.

Type 1, Type 2, and Type 3 clones are based on textual similarity. Code fragments are considered clones if they are textually similar even they are functionally different. Textual clones are more common in software code base because they are usually results of copy/paste practice. On the other hand, semantic clones are harder to detect since they could be implemented by different syntactics.

A number of definitions have been proposed in the literature for semantic clones. The terms relative clones [21], redundant code [14], [22], dependent clones [4], functional clones [9], functionally similar clones [6], [13], [25] and type 4 clones [17] are widely used to refer to semantic clones. Some of them narrow the definition of semantic clone to one type of semantic while others used a wider (non-specific) meaning of semantic. However, all mentioned definition agree that (1) semantic clones have the same functionality and (2) are implemented through different syntax.

III. Methodology

Stack Overflow is a website that enables users to ask and answer questions related to programming languages. It has over 19 million questions, 28 million answers, 73 million comments, and 11 million registered users[1]. For each question, participants submit different answers. We can be sure that correct answers should be functionally equivalent. On the other hand, some questions do not have multiple answers, answers could be incomplete or erroneous, answers could not represent a fully functional unit, or answers could be syntactically similar which do not comply with the definition of the semantic clone. Therefore, we design a selection process that includes automatic validation, filtering, and selection to overcome all the aforementioned challenges and reduce the number of candidate clones that need manual validation. The complete process construction of our semantic code clone benchmark is summarized in Figure 1.

[1]https://data.stackexchange.com/stackoverflow/queries

A. Data Extraction

Our process starts by collecting the Stack Overflow questions and answers for the selected programming languages (Java, C, C#, and Python) from ***SOTorrent*** [1] database. Our query retrieves all answers that solve programming language problem (Contain source code) and grouped by question. These answers create the initial clone class.

In the ***SOTorrent*** database, the ***Posts*** table contains all the question, answer, and comment posts, and each of them associates with an unique ***postType***. Listing 1 shows our query to retrieve all C answers. We limited our query for answers only by specifying the *PostTypeId* to 2. The condition *init.Body like'%<code>%´* specifies answers that contain code snippet, and the programming language is specified by *parent.Tags like'%<c>%'*. In the ***Posts*** table, only the question posts are associated with the programming language tags, and the answers are not. Therefore, we used left join method to find the programming language involved in the problem post.

Listing 1. An example of SQL Query

```
SELECT init.Body as Answer
,init.ID as Post_Link
,parent.ID as Id
,parent.Title as Title
FROM [sotorrent-org:2018_12_09.Posts] init
LEFT JOIN [sotorrent-org:2018_12_09.Posts]
parent on init.Parentid = parent.Id
Where init.Body like '%<code>%'
AND init.PostTypeID = 2
AND parent.Tags like '%<c>%'
```

We only need to apply different filtering keywords to the last line of the query, corresponding programming language related answer posts can be queried easily. Finally, approximately 4 million answers were queried from ***SOTorrent*** by above mentioned method. Table I column# 2 and 3 show the total number of questions and answers in Stack Overflow. The results of our query are shown in column# 4 and 5.

B. Syntax Validation

Answers in Stack Overflow contain both text and code snippet. Fortunately, the code snippet in each answer on Stack Overflow is surrounded by a $< code >$ tag. This greatly

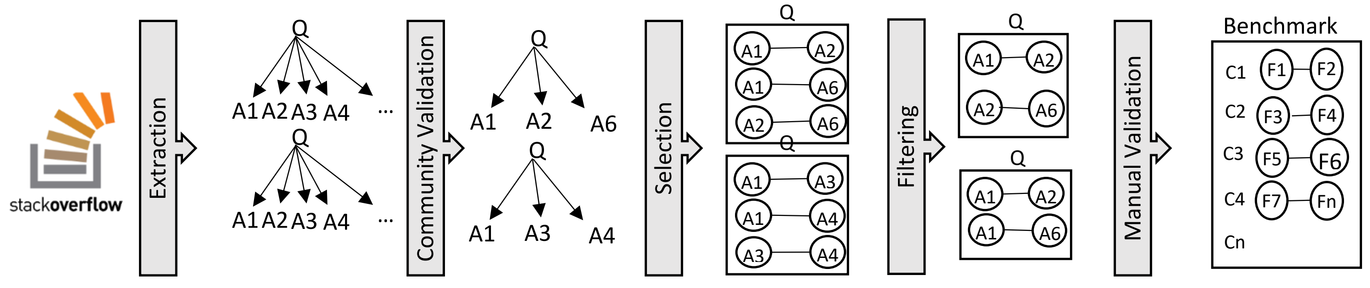


Fig. 1. A schematic diagram that shows the complete process for constructing a semantic code clone benchmark using data from Stack Overflow

TABLE I
NUMBER OF QUESTIONS AND ANSWERS IN STACK OVERFLOW AND EACH PROCESSING STEP

Filter	No Filter (Total Number)		>=2 answers contain source code		Include methods		>=10 lines		Votes>0	
Language	Questions	Answers	Questions	Answers	Questions	Answers	Questions	Answers	Questions	Answers
Java	1556189	2582119	432086	1183439	95837	435186	30153	92788	15320	60374
C	297405	602044	119990	338519	26597	90202	10435	30702	6972	21236
C#	1310182	2216152	375898	1025420	52827	176384	12180	33211	8615	26777
Python	1223511	1807837	359051	968235	57820	219632	23935	70485	15227	49886

reduces the difficulty of code extraction, as it only needs to apply a regular expression to match the content between $< code >$ and $< /code >$ and store them separately in the code file.

Code snippets in answers come in different granularities. Some are just a line of code or a few lines that show how an API is used. Others come as a function or a full program. Most of code snippets are not a complete program. Therefore, a solid functional unit should be extracted that is error-free.

For the purpose of building a semantic code clone benchmark we have chosen method/function level granularity, that is because the non-function code snippets are usually too short to express the meaning behind it. On the contrary, the method usually contains more information and the process of the solution. Therefore, the method will be more helpful in determining clones. TXL [3] is used in this step to extract methods from the code snippets. TXL parser extracts syntactically correct methods only. Also, we used TXL to normalize the selected methods. This includes the removal of comments, white spaces and a strict pretty printing. Table I column# 7 shows the number of answers that contain methods. In some cases, some answers contain more than one method so we saved all methods in the answers for further selection process.

We also performed a minimum-line-limit filter in this step to make sure all extracted functions have at least 10 lines of code [24]. This also helped us to filter out about 75% of the candidate functions, and will reduce the manpower for manual validation later. Look Table I column 9.

C. *Functionality Validation*

After all aforementioned selection processes, we still have a large number of methods to validate (92K java methods, 30K C methods, 10K C# methods, and 21K Python methods). In order to reduce manual validation, we applied another filter as per the quality of the answers by the Stack Overflow community.

Stack Overflow users participate in asking questions, answering, commenting, and voting in order to get rewards. Stack Overflow employs a voting mechanism for questions and answers to help users to identify trustworthy answers [7]. Answers are posts that have up-votes and down-votes with the best answers receiving more up votes. For the purpose of our benchmark we considered only up-voted methods and ignored non-voted and the down-voted methods. Up-voted methods represent 60% of total methods so this step reduces the total number of methods by 40%, see Table I (last two columns).

D. *Syntactic clone filtering*

The main goal of this study is to build a real semantic clone benchmark. Therefore, we reviewed all the definitions of semantic clone in the lecture and we found that most definitions agree that semantic clones are code fragments that perform the same computation but are implemented through different syntactic variants [17]. Therefore, we identified syntactically similar fragments using NiCad [18] and eliminate them. NiCad has a high precision and recall in detecting syntactic clones [24]. It is true that the eliminated answers are functionally equivalent. But, they are not considered semantic clones according to the definition of semantic clones.

Before using NiCad to detect syntactic clones we build all possible clone pairs. For each question, all answers combinations are candidate clones. However, some answers have more than one method. We used two heuristics to select between methods. First, we chose the biggest method in the answer. Table II, the second column shows the number of candidate clone pairs. Second, we selected methods in answers that have the same name. Column #5 shows the number of candidate clones that have the same name.

TABLE II
NUMBER OF CLONE PAIRS AFTER SYNTACTIC FILTERING

Language	Clone pairs (biggest method)	Detected By NiCad	**Candidate clones**	Clone pairs (same name)	Detected By NiCad	**Candidate clones**
Java	18146	3141	**15049**	7,627	2,019	**5,944**
C	6210	312	**5898**	3,383	372	**3,011**
C#	7404	488	**6916**	1,986	523	**1,507**
Python	15103	2773	**12343**	7,322	2,451	**5,320**

TABLE III
NUMBER OF VALIDATED CLONE PAIRS

Language	Number of validated clones	True semantic clones	Validation time(hours)
Java	1775	1000	26
C	1742	1000	32
C#	1894	1000	26
Python	2020	1000	30
Total	7431	4000	114

Table II shows the number of syntactic clone pairs detected by NiCad. Clones that are detected by NiCad represent syntactic clones (around 18%) are filtered out for the purpose of the semantic clone benchmark. It is evident that there is a good number of syntactic clones exist in Stack Overflow answers. These clones are perfect to construct a syntactic benchmark. To avoid the bias of using clone detector, we can select a random set of answers and validate them manually or we can select answers that are textually similar and validate them manually.

E. Manual Validation

Manual validation is the last task for building benchmarks. We hire two judges to mark all method pairs as true positive or false positive according to their functionality. We gave them a third option "Undecided" for confusing pairs. The judges are summer students who did not involve in the research. Rather, they introduced to the topic of semantic clones and read a number of related papers. Our final goal to have 1000 clone pairs for each programming language. Table III shows the number of candidate clones are needed to be validated to reach 1000 semantic clone pairs for each language. Also, it shows the time needed to finish validation. A total of 114 hours spent by two judges to validate 7432 clone pairs.

Manual validation proves the feasibility of our selection process in finding semantic clones candidates. 54% of selected candidates are true semantic clone. Unlike Krutz and Le [11], they examine 1536 candidate clones to identify 66 clone pairs, only 9 semantic semantic clones for one programming language. Also, the selection and filtering process helped to reduce the effort in building a benchmark of 4000 clone pairs. Only 114 hours are needed by two judges to finish the validation load. Finally, Figure 3 shows an example of a semantic clone that is tagged as true by the judges.

[2]https://stackoverflow.com/questions/1019793/how-can-i-convert-string-to-int

TABLE IV
SUBJECT SYSTEM TO INJECT CLONES

System	Language	Number of files	Line of code
JHotDraw7	Java	711	130k
PostgreSQL-12.0	C	1343	1368k
Mono1.1.4	C#	9822	5518k
django	Python	2031	240k

IV. SEMANTICCLONEBENCH IN USE

We designed our semantic code clone benchmark (SemanticCLoneBench) into two forms, injected in a system form and stand-alone clones form. SemanticCloneBench is available online for use [3].

Injected in a system. We designed SemanticCloneBench as a real software system that has clones. We selected 4 systems (Table IV) and we inject the clones into random locations. We considered medium size systems since we know not all detection tools are scalable for large systems. We inject clones into syntax-correct locations, where no clone pair are injected in the same file. Each subject system is associated with the clone references.

Stand alone clones. We also kept each clone pair in a single text file for another usage. Practitioners could use a subset of clones for other testing or could inject them into other systems. Some clone detectors cannot scale for large systems and others have certain limitations. For example, Oreo [19] detect clones only in the second directory of the file structure.

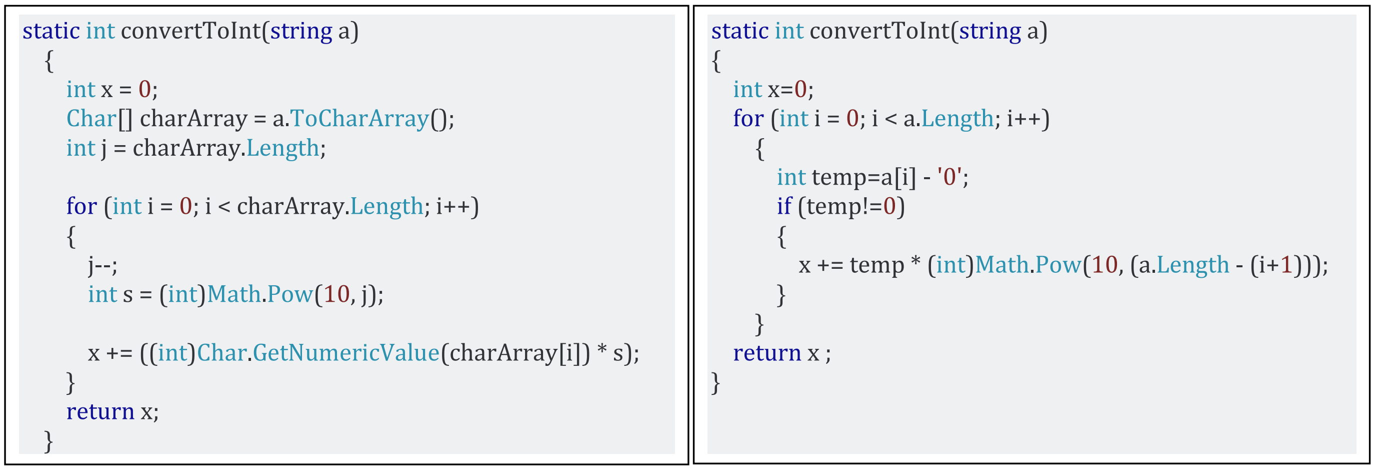
A. Evaluating clone detectors using SemanticCloneBench

SemanticCloneBench can be used to measure the semantic recall of clone detection tools. Recall is defined as the fraction of the true clone detected by a clone detection tool (1), where $Recall_{Sem}$ is the measured semantic recall, D_{clone} is the set of detected clones by the tool and R_{clone} is the set of reference clones in SemanticCLoneBench.

$$Recall_{Sem} = \frac{D_{clone} \cap R_{Clone}}{R_{Clone}} \qquad (1)$$

We used SeomticCLoneBinch to measure semantic reall for three clone detectors (See Table V). We used NiCad to filter out syntax clones in an earlier stage of building the benchmark. We are using it again with different configurations. We increased the dissimilarity threshold (UPI) of NiCad from 0.3 to 0.35 and used blind renaming of identifiers to enable NiCad to detect more gaped clones. We ran NiCad and SimCad

[3]https://drive.google.com/open?id=1KicfslV02p6GDPPBjZHNlmiXk-9IoGWl



```
static int convertToInt(string a)
  {
    int x = 0;
    Char[] charArray = a.ToCharArray();
    int j = charArray.Length;

    for (int i = 0; i < charArray.Length; i++)
    {
      j--;
      int s = (int)Math.Pow(10, j);

      x += ((int)Char.GetNumericValue(charArray[i]) * s);
    }
    return x;
  }
```

```
static int convertToInt(string a)
{
  int x=0;
  for (int i = 0; i < a.Length; i++)
    {
      int temp=a[i] - '0';
      if (temp!=0)
      {
        x += temp * (int)Math.Pow(10, (a.Length - (i+1)));
      }
    }
  return x ;
}
```

Fig. 2. Semantic clone pair form Stack Overflow post ”How can I convert String to Int? ” [2]

TABLE V
CLONE DETECTION TOOLS' RECALL

Tool	Granularity	Target language	Clone type	Recall
NiCad	Method	Java,C,C#, and Python	1,2,3	0.04
SimCad	Method	Java, C, C#, and Python	1,2,3	0.025
Oreo	Method	Java	1,2,3 and 4	0.097

on the injected JhotDraw7 with the 1000 clones. NiCad was able to detect 40 clones while SimCad detected 25 clones. Oreo runs on source code in the second level of the file structure only. Therefore, we had to inject the clones in certain locations for Oreo. Oreo is able to detect only 97 clones.

We have not expected a good recall for clone detection tools on SemanticCloneBench since most tools are based on measuring similarity to identify code clones. SmanticCloneBench represents the region where most detection tools are difficult to perform. That is because it follows the definition of semantic clones.

V. Textual similarity in SemanticCloneBench clones

Semantic clones represent a challenge for most detection tools where it is hard for most tools to reach a good recall since semantic clones are not textually similar. Most tools that are based on the similarity threshold, set up the threshold value at the point where false positive started to produce. These tools never attempt to detect clones with a lower threshold in order to keep high precision. However, semantic clones carry some textual similarity. But it is still not examined by detection tools. In this section, we measured the textual similarity of clones in SementicCloneBench.

We start by normalizing the code in the benchmark. This includes removal of comments and white-spaces, normalizing all literals and identifiers and pretty-printing of the code. Then we used the Longest Common Subsequence (LCS) [8] algorithm on the sequence of normalized lines to measure the textual similarity between clones. We scale the similarity in the range of 0 to 100, where 100 means textual identical and 0 means total-textual different. Fig. 3 shows the distribution of clones in SemanticCloneBench over the similarity scale.

The figure shows that semantic code clones hold a low textual similarity. The majority of Java, C, and C# semantic clones are less than 50 textual similar. This means that clone detection tools should set their threshold to a value less than the range in order to detect clones in that range. The figure shows that Python semantic clones have more textual similarity. Most Python clones reside in the range of 30 to 60. But it is still very low comparing to the threshold used by detection tools which justify the shortage of most detection tools to detect semantic clones.

VI. Related Work

The first clone benchmark was created by Bellon et al. [2] in 2007. They verified 2% of clones reported by six clone detection tools in four C systems and four Java systems. This verification was performed by one judge. The main issue in this benchmark is created by one author and all candidates are clone reports by tools. Therefore, not all types of clones are included in this benchmark.

Similarly, Krutz and Le [11], selected a random set of 1536 method pairs in three C open source programs, *Apache*, *Python* and *PostgreSQL*. They hired three experts judges and four students to evaluate these pairs manually. They found that only 66 clone pairs, out of the 1536 candidate clones, were real clones. Their benchmark contains 43 type-2 clones, 14 type-3 clones and 9 type-4 clones (semantic clones).

Svajlenko at al. [23] hired 9 judges to manually tag candidate clones as true positive or false positive. They identified 44 common functionalities that are used widely in *IJaDataset*. Unlike the aforementioned benchmarks, they used a search heuristic to identify candidate clones, for these functionalities, instead of using clone detection tools. They were able to build

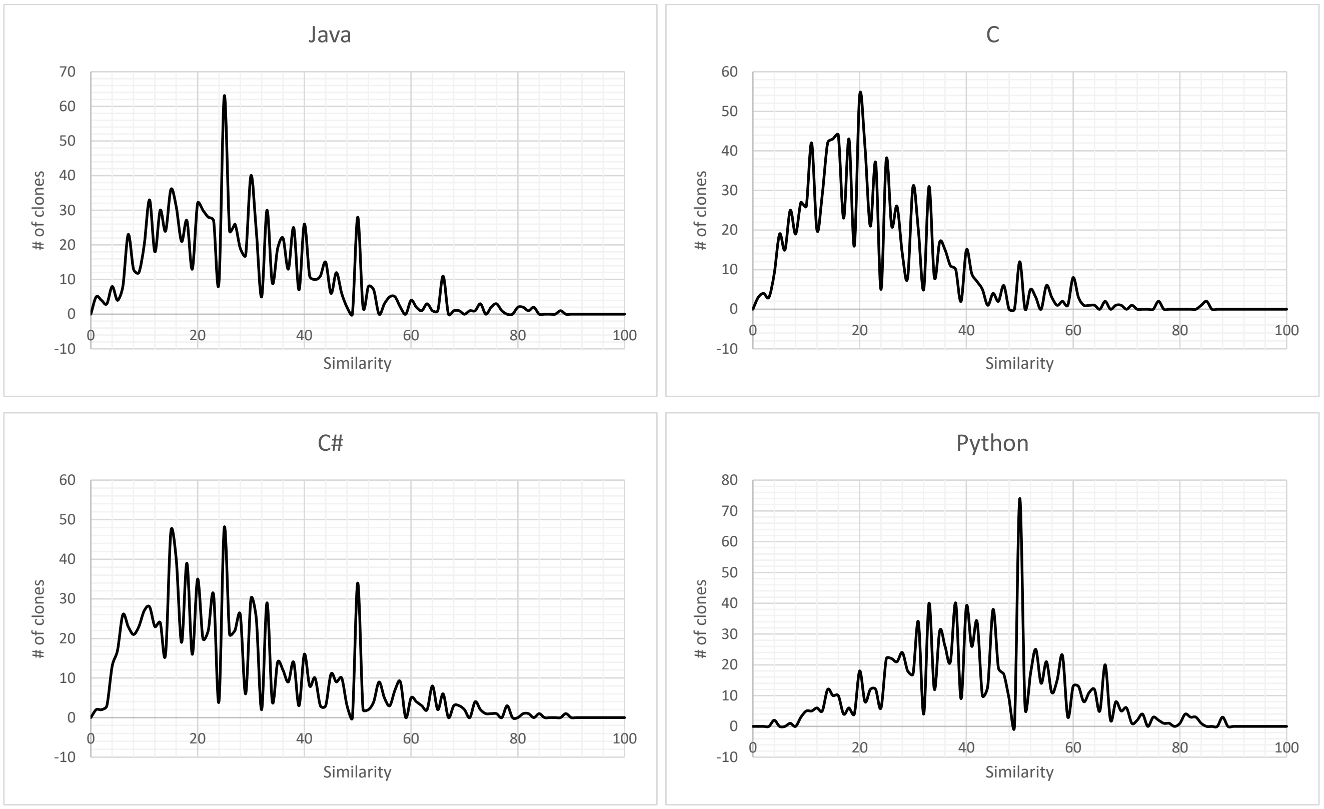


Fig. 3. Textual similarity between the clones of SemanticCloneBench

a large benchmark that contains all types of clones. However, their benchmark is only for Java language.

Roy and Cordy [15] design various scenarios to create different types of clones that are injected in the code base and used in the evaluation process to measure recall and precision. However, these scenarios have to be comprehensive (covering all types of clones that could appear in real source code) and not depend on any clone definition.

Recently, Yuki et al. [29] proposed a technique to build a benchmark through mining software versions and identify merged methods (merged cloned methods in the next version). Two merged methods are considered clones if they are textually similar and called by the same methods in the next version. Unfortunately, their technique is limited to only refactored clones, which are a small portion of code clones. In more than 15K versions they were able to identify 19 clones only.

Other studies [6], [21], [26] used Google Code Jam [4] as a semantic clone benchmark.They considered solutions for the same question semantic clones. The correct solutions that pass judgments of the contest are truly semantic programs. But the majority of clone detectors do not work on a file or a program granularity. Therefore, benchmarks should be built on a more fine grain granularities such as block or method. Wanger et al. [26] used the main functions of the solutions as semantic clones to be able to evaluate CCCD [12]. Then they build a semantic clone benchmark by considering the main method only and excluding syntactically similar methods. Their benchmark includes 29 clone pairs only (16 Java clones and 13 C clones).

[4]https://codingcompetitions.withgoogle.com/codejam

VII. Threats to validity

The clones in our benchmark may not be real clones. However, these clones are provided as example solutions to many problem by many real developers. Furthermore, we considered the method level clones and those received high votes by the stack Overflow users. Thus, the clones in our benchmark might be real semantic clones to some extent. In order to guarantee further accuracy, judges also validated the clones. Of course, there might be errors in the manual validation process and in future we plan to involve more judges for cross validation. In filtering out syntactic clones (type-1, type-2 and type-3 clones) we used the NiCad clone detector. NiCad might not have filtered out all such clones. However, NiCad has been widely used for detecting such types of clones. Furthermore, even if there are some syntactic clones in the semantic benchmark, they may not impact much in evaluating the semantic clone detection tools. Of course, the presence of more syntactic clones may wrongly promote some tools for their capability of detecting semantic clones. We plan to address this concern with a future release of the benchmark.

VIII. Conclusion an future work

Semantic code clones are the most challenging type of clones to detect. Also evaluating semantic clone results is fuzzy according to the definition and understanding of semantic clones. In this paper, we mine the knowledge of the Stack Overflow to find semantic code clones with minimal human effort. We considered the answers for the same programming question are a functionally equivalent code and methods in the answers are semantically similar answers. We extracted methods from Stack Overflow answers. Then we filter out small, incomplete and low-voted answers by Stack Overflow community. We validated 7431 candidate pairs to build a semantic clone benchmark of 4000 clone pairs in the languages Java, C, C# and Python. These clone pairs could be used as a piece of knowledge to understand how semantic clones are syntactically different and would guide the research to enhance the detection of semantic clones.

For the next release of semantic clone benchmark, we plan to extend the benchmark with other languages and double validate the clones with more judges. Also, we could extend the benchmark with syntactic clones, that are extracted from Stack Overflow by a similar procedure. In addition, there are some interesting future work that will be performed. For example, we will use the benchmark to evaluate existing semantic clone detectors. Or, we plan to use the benchmark to train a machine learning model to build an AI-based semantic clone detector.

References

[1] Sebastian Baltes, Christoph Treude, and Stephan Diehl. Sotorrent: Studying the origin, evolution, and usage of stack overflow code snippets. In *Proceedings of the 16th International Conference on Mining Software Repositories (MSR 2019)*, 2019.

[2] Stefan Bellon, Rainer Koschke, Giulio Antoniol, Jens Krinke, and Ettore Merlo. Comparison and Evaluation of Clone Detection Tools. *IEEE Transactions on Software Engineering*, 33(9):577–591, sep 2007.

[3] James R. Cordy. Source transformation, analysis and generation in txl. In *PEPM*, 2006.

[4] Tim A D. Henderson and Andy Podgurski. Rethinking dependence clones. In *2017 IEEE 11th International Workshop on Software Clones (IWSC)*, pages 1–7. IEEE, Feb 2017.

[5] Florian Deissenboeck, Elmar Juergens, Benjamin Hummel, Stefan Wagner, Benedikt Mas y Parareda, and Markus Pizka. Tool Support for Continuous Quality Control. *IEEE Software*, 25(5):60–67, sep 2008.

[6] Fang-Hsiang Su, J. Bell, G. Kaiser, and S. Sethumadhavan. Identifying functionally similar code in complex codebases. In *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*, pages 1–10, May 2016.

[7] Neelamadhav Gantayat, Pankaj Dhoolia, Rohan Padhye, Senthil Mani, and Vibha Singhal Sinha. The synergy between voting and acceptance of answers on stackoverflow, or the lack thereof. In *Proceedings of the 12th Working Conference on Mining Software Repositories*, MSR '15, pages 406–409, Piscataway, NJ, USA, 2015. IEEE Press.

[8] James W. Hunt and Thomas G. Szymanski. A fast algorithm for computing longest common subsequences. *Communications of the ACM*, 20(5):350–353, May 1977.

[9] Lingxiao Jiang and Zhendong Su. Automatic mining of functionally equivalent code fragments via random testing. In *Proceedings of the Eighteenth International Symposium on Software Testing and Analysis*, ISSTA '09, pages 81–92, New York, NY, USA, 2009. ACM.

[10] Heejung Kim, Yungbum Jung, Sunghun Kim, and Kwankeun Yi. MeCC. In *Proceeding of the 33rd international conference on Software engineering - ICSE '11*, page 301, New York, New York, USA, 2011. ACM Press.

[11] Daniel E. Krutz and Wei Le. A code clone oracle. In *Proceedings of the 11th Working Conference on Mining Software Repositories - MSR 2014*, pages 388–391, New York, New York, USA, 2014. ACM Press.

[12] Daniel E Krutz and Emad Shihab. CCCD: Concolic code clone detection. In *2013 20th Working Conference on Reverse Engineering (WCRE)*, pages 489–490. IEEE, oct 2013.

[13] V. Käfer, S. Wagner, and R. Koschke. Are there functionally similar code clones in practice? In *2018 IEEE 12th International Workshop on Software Clones (IWSC)*, pages 2–8, March 2018.

[14] António Menezes Leitão. Detection of redundant code using r2d2. *Software Quality Journal*, 12(4):361–382, December 2004.

[15] Chanchal K. Roy and James R. Cordy. A Mutation/Injection-Based Automatic Framework for Evaluating Code Clone Detection Tools. In *2009 International Conference on Software Testing, Verification, and Validation Workshops*, pages 157–166. IEEE, 2009.

[16] Chanchal K. Roy, James R. Cordy, and Rainer Koschke. Comparison and evaluation of code clone detection techniques and tools: A qualitative approach. *Science of Computer Programming*, 74(7):470–495, May 2009.

[17] Chanchal Kumar Roy and James R Cordy. A Survey on Software Clone Detection Research. *Computing*, 541:115, 2007.

[18] C.K. Roy and J.R. Cordy. NICAD: Accurate Detection of Near-Miss Intentional Clones Using Flexible Pretty-Printing and Code Normalization. In *2008 16th IEEE International Conference on Program Comprehension*, pages 172–181. IEEE, jun 2008.

[19] Vaibhav Saini, Farima Farmahinifarahani, Yadong Lu, Pierre Baldi, and Cristina V. Lopes. Oreo: Detection of clones in the twilight zone. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2018, pages 354–365, New York, NY, USA, 2018. ACM.

[20] Abdullah Sheneamer and Jugal Kalita. Semantic Clone Detection Using Machine Learning. In *2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 1024–1028. IEEE, dec 2016.

[21] Fang-Hsiang Su, Jonathan Bell, Kenneth Harvey, Simha Sethumadhavan, Gail Kaiser, and Tony Jebara. Code relatives: Detecting similarly behaving software. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2016, pages 702–714, New York, NY, USA, 2016. ACM.

[22] M. Suzuki, A. C. d. Paula, E. Guerra, C. V. Lopes, and O. A. L. Lemos. An exploratory study of functional redundancy in code repositories. In *2017 IEEE 17th International Working Conference on Source Code Analysis and Manipulation (SCAM)*, pages 31–40, Sep. 2017.

[23] J Svajlenko, JF Islam, and I Keivanloo. Towards a big data curated benchmark of inter-project code clones. *and Evolution (ICSME ...*, 2014.

[24] Jeffrey Svajlenko and Chanchal K. Roy. Evaluating Modern Clone Detection Tools. In *2014 IEEE International Conference on Software Maintenance and Evolution*, pages 321–330. IEEE, sep 2014.

[25] Ryo Tajima, Masataka Nagura, and Shingo Takada. Detecting functionally similar code within the same project. In *2018 IEEE 12th International Workshop on Software Clones (IWSC)*, pages 51–57. IEEE, Mar 2018.

[26] Stefan Wagner, Asim Abdulkhaleq, Ivan Bogicevic, Jan-Peter Ostberg, and Jasmin Ramadani. How are functionally similar code clones syntactically different? An empirical study and a benchmark. *PeerJ Computer Science*, 2:e49, mar 2016.

[27] A. Walenstein, N. Jyoti, Junwei Li, Yun Yang, and A. Lakhotia. Problems creating task-relevant clone detection reference data. In *10th Working Conference on Reverse Engineering, 2003. WCRE 2003. Proceedings.*, pages 285–294. IEEE, 2003.

[28] S. Wang, T. P. Chen, and A. E. Hassan. How do users revise answers on technical q a websites? a case study on stack overflow. *IEEE Transactions on Software Engineering*, pages 1–1, 2018.

[29] Yusuke Yuki, Yoshiki Higo, Keisuke Hotta, and Shinji Kusumoto. Generating clone references with less human subjectivity. In *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*, pages 1–4. IEEE, may 2016.