Measuring Code Similarity in Large-scaled Code Corpora

Chaiyong Ragkhitwetsagul
CREST, Department of Computer Science
University College London, UK

***Abstract*—Source code similarity measurement is a fundamental technique in software engineering research. Techniques to measure code similarity have been invented and applied to various research areas such as code clone detection, finding bug fixes, and software plagiarism detection. We perform an evaluation of 30 similarity analysers for source code. The results show that specialised tools including clone and plagiarism detectors, with proper parameter tuning, outperform general techniques such as string matching. Although these specialised tools can handle code similarity in local code bases, they fail to locate similar code artefacts from large-scaled corpora. This is increasingly important considering the rising amount of online code artefacts. We propose a scalable search system specifically designed for source code. It lays a foundation to discovering online code reuse, large-scale code clone detection, finding usage examples, detecting software plagiarism, and finding software licensing conflicts. Our proposed code search framework is a hybrid of information retrieval and code clone detection techniques. This framework will be able to locate similar code artefacts instantly. The search is not only based on textual similarity, but also syntactic and structural similarity. It is resilient to incomplete code fragments that are normally found on the Internet.**

I. Introduction

Source code similarity measurement is frequently used in software engineering research. Techniques to measure code similarity are invented and applied to various research areas such as code clone detection [6], finding bug fixes [23], and software plagiarism detection [35]. These techniques are successfully put to use in analysing software projects and local code bases in companies or open source systems [10], [15], [20], [29], [32]. Nowadays there are plethora of source code files and incomplete code fragments proliferating on the Internet. For example, online code artefacts from GitHub or Stackoverflow are invaluable since they can be mined in order to gain insights of several emerging challenges in software development, e.g. online code reuse, software licensing conflicts [10], [12], and usage examples [24]. Computer science students are also benefited from online Q&A websites by having a vast amount of examples to study from [33]. On the other hand, the reuse of online code examples can lead students to the problem of plagiarism [18], [19].

Most of similar online code artefacts are not only textually similar but they also can share similarity at syntactic or structural level. This syntactic or structural similarity commonly occur due to modifications made to the code such as variable renaming, statements insertions, deletions, and replacements. Current text-based search engines such as Bing and Google are not specifically designed for source code. Code search engines such as BlackDuck OpenHub [1] also do not handle modifications made to source code very well [24]. Unfortunately, specific tools like classical code clone [16], [21], [37] and source code plagiarism detectors [17], [35] cannot handle this large amount of code artefacts. They cannot be scaled to process millions of files with billion lines of code in a reasonable amount of time. This prompts new techniques for large-scale clone detection to emerge [25], [34], [38]. However, these techniques mainly target on complete code blocks or functions while galore of code snippets on the Internet, e.g. from Stackoverflow, are incomplete code fragments.

II. Research Question and Contributions

This research is motivated by a question **"how to effectively locate similar source code with pervasive modifications from large-scaled code corpora?"**. To answer this question, we propose the following contributions:

A broad, thorough study of the performance of tools and techniques against pervasively modified source code: We compared 30 source code similarity measurement tools and found that specialised tool such as clone detection and plagiarism detection tools outperform compression-based and general document similarity tools. We also evaluated a normalisation technique using decompilation and found that it is effective against pervasive modifications [36].

Internet-scaled similar code search: We are creating and evaluating a large-scaled similar code search technique that is resilient to incomplete code. The preliminary results show that our information retrieval framework enhanced by clone detection techniques can locate similar code fragments with high precision.

Applications of the code search framework: We plan to apply our large-scale code search framework to empirical software engineering research. Specifically, we will perform a study of finding online code reuse between online code corpora (e.g. Stackoverflow, GitHub) and open source projects.

III. Related Work

Cloned code is created by duplication of source code due to reusing of well-written code, hardware/platform variations, templating, or customisations [22]. Similarly, software plagiarism is a copying of source code with an intention to hide its origin and found in both computer science education [8], [9]

To be published in the Proceedings 32nd International Conference on Software Maintenance and Evolution 2016, in Raleigh, NC, USA.

and software industry [3]. To locate these similar pieces of cloned and copied code, several clone and plagiarism detectors have been invented in the past decades. They are based on various code similarity measurement techniques such as string matching [37], [41], token matching [7], [11], [13], [14], [17], [21], [35], [38]–[40], [42], tree similarity [5], [16], or sub-graph similarity [27], [28].

Code clone and plagiarism detection tools usually apply normalisation to source code. Code normalisation is a process of transforming source code into an intermediate representation. The similarity detection is then performed on the intermediate representation instead of the original source code text. Normalisation can improve the tools' precision and recall by canonicalising textual and structural changes that have been made to the source code. Examples of code normalisation found in clone and plagiarism detection include a usage of pretty printing [37], blind or consistent identifier renaming [21], [35], [37], token stream [21], [35], and abstract representation using abstract syntax tree (AST) [5], [16] or program dependence graph (PDG) [28].

Information retrieval (IR) technique is an established field of study underpinning modern search engines. Information retrieval techniques provide nearly instant searching time over massive datasets. There are numbers of indexing, querying, and ranking methods introduced by IR research community [4], [26], [31]. It has been mainly used for text search in very large databases or over the web. Recently, source code similarity detection using information retrieval technique is emerging in modern clone and plagiarism detection due to its scalability over large code bases [7], [38].

IV. Methodology and Results

This research is divided into two parts. The first part covers a study of existing similarity measurement tools and techniques. The second part introduces a novel method to search for similar code in a large-scaled code database and its applications.

A. Code Similarity in the Presence of Pervasive Modifications

We performed an empirical study of 30 similarity analysers on 50 Java source code with pervasive modifications. The pervasive modifications are code changes that affect the whole source file and are mostly found in code cloning, code plagiarism, and code refactoring. We created a ground truth dataset of pervasively modified code using ARTIFICE (source code obfuscator), ProGuard (byte code obfuscator), Java compiler (*javac*), and two decompilers: Krakatau and Procyon. The overall process is depicted in Figure 1. We applied obfuscators and decompilers to each original Java file to generate several of its pervasively modified counterparts. The 30 analysers are executed against this dataset and report pairwise similarity values. Only pairs of Java programs that are from the same original program are true positives. Thus, we thoroughly searched and selected the tools' settings and similarity thresholds that give the best classifications. Finally we compare the tools using their F-scores.

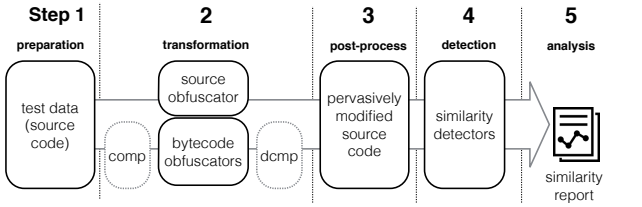

Fig. 1. The experimental framework for evaluation of code analysers against pervasively modified source code [36]

Furthermore, it is known that normalisation strongly affects the accuracy of similarity detection. We included compilation and decompilation as a pre-processing step of source code with pervasive modifications. The evaluation of the tools was repeated with a compilation and decompilation applied to the dataset before performing the detection. We analysed the results and compared them using F-scores.

Results: Table I lists the selected tools, including clone detectors; plagiarism detectors; compression tools; general similarity measurement tools, and their performance against pervasive modifications. The best performing tool in terms of F-score is the widely used token-based clone detector, *CCFinderX (ccfx)*, followed by *simjava*, a token-based plagiarism detector, and the string-based clone detector, *simian*. Normalised compression distance (NCD) which is a compression-based algorithm were observed to give comparable F-scores over all of its variants with different compression techniques. Table I only reports the tools' optimised parameter settings due to space limitations. The complete optimised settings and results can be found from the study website[1].

Furthermore, we observed that normalisation by decompilation reduces the numbers of false positive and false negative of all the tools. The two decompilers, Krakatau and Procyon, provide comparable performance with Krakatau having slightly better F-measure scores for most of the tools. Six tools including *ccfx, deckard, jplag-java, plaggie, sherlock*, and *simjava* did not report any false classification at all after compiled and decompiled by Krakatau.

B. Internet-scaled Similar Code Search: ISiCS

Classical code clone and plagiarism detectors do not scale well to large-scaled datasets [38]. Although there are scalable approaches to overcome this limitation [25], [34], [38], they still fail to handle incomplete code fragments. This is crucial since there are many of these fragments residing on the Internet especially on Q&A websites. Our "Internet-scaled Similar Code Search (ISiCS)" framework is a code search framework that is scalable and resistant to code incompleteness.

In ISiCS, information retrieval (IR) techniques are exploited as the underlying infrastructure of code retrieval and similarity measurement. Code clone detection techniques are adopted for code abstraction and normalisation. The search indexes are designed to store online code artefacts collected from

[1] http://crest.cs.ucl.ac.uk/resources/cloplag/

TABLE I
TOOL PERFORMANCE COMPARISON ON PERVASIVELY MODIFIED JAVA SOURCE CODE WITH AND WITHOUT DECOMPILATION (KRAKATAU) USING THE TOOLS' OPTIMAL CONFIGURATIONS.

Tool/Technique	Optimised Settings (Original)	F-score	
		Original	Decomp.
Clone det.			
ccfx	b=20,21,24,t=1..7 b=22,23,t=7	0.9095	1.0000
deckard	MINTOKEN=30 STRIDE=2 SIMILARITY=0.95	0.8595	1.0000
iclones	minblock=10 minclone=50	0.6033	0.9407
nicad	abstract=expressions	0.7080	0.9370
simian	minline=5,ignoreIdf	0.8719	0.9980
Plagiarism det.			
jplag-java	t=3	0.8045	1.0000
jplag-text	t=8	0.8582	0.9843
plaggie	M=7	0.8210	1.0000
sherlock	N=6,Z=3	0.8284	1.0000
simjava	r=22	0.8941	1.0000
simtext	r=4	0.5622	0.9754
Compression			
7zncd-BZip2	mx=1,3,5	0.8301	0.9494
7zncd-LZMA	mx=7,9	0.8160	0.9501
7zncd-LZMA2	mx=7,9	0.8189	0.9511
7zncd-Deflate	mx=9	0.8157	0.9500
7zncd-Deflate64	mx=9	0.8142	0.9500
7zncd-PPMd	mx=9	0.8078	0.9513
bzip2ncd	C=1..9	0.8219	0.9453
gzipncd	C=9	0.8153	0.9535
icd	ma=Deflate,Deflate64 mx=9	0.7404	0.8605
ncd-bzlib	N/A	0.8163	0.9419
ncd-zlib	N/A	0.8282	0.9474
xz-ncd	-e	0.8228	0.9560
Others			
bsdiff	N/A	0.5797	0.9075
diff	N/A	0.6996	0.8815
py-difflib	SM_noautojunk	0.8393	0.9056
py-fuzzywuzzy	token_set_ratio	0.8167	0.9712
py-jellyfish	jaro_distance	0.6169	0.7937
py-ngram	N/A	0.7925	0.9098
py-sklearn	N/A	0.6802	0.9107

famous programming websites. These online code artefacts can be extracted from archives provided for downloads from Stackoverflow and GitHub, or crawled by web crawlers from websites that do not provide such download options. One of ISiCS goals is resiliency to incomplete code artefacts. Thus, its token-based similarity measurement method do not rely on code block or function completeness. Incomplete code snippets are processed as-is while complete source code files are parsed and processed at method level.

We envisage the complete ISiCS framework to contain more than one search index. Multiple indexes with different code abstract representations can enhance the recall of the search. Using this method, code artefacts are stored in one index only if they can be transformed into that index's required representation. Figure 2 represents the "multi-representation" search framework containing three search indexes. The first

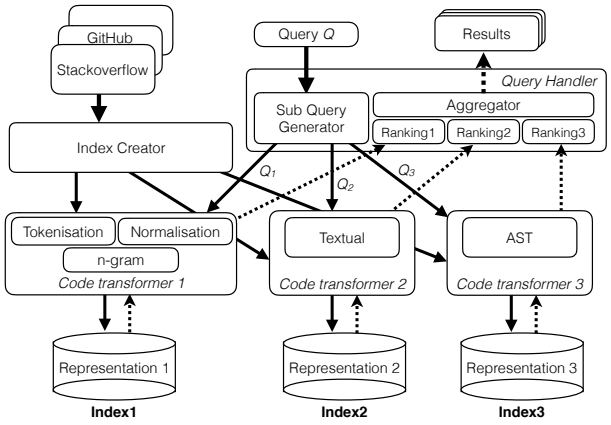

Fig. 2. The framework of Internet-scaled Similar Code Search (ISiCS)

index exploits token-based representation. It stores source code in a form of *n*-grams of normalised tokens [6], [21]. The normalisation handles variable name, data type, and identifier modifications while *n*-grams [30] is used to capture code sequences. The second and the third index store source code in its textual and structural representation (e.g. AST [5], [16] or PDG [28]) respectively. For each index, we choose a similarity measure from the choices of six state-of-the-art IR similarity ranking functions: TF-IDF, BM25, DFR, IB, LMDirichlet, and LMJelinekMercer [2], [4], [31].

At query time, a code query Q is processed to create three sub queries Q_1, Q_2, Q_3 as depicted in Figure 2. Each of the sub queries contains suitable representation of the code according to its respective index. The sub queries are then use to search from their designated index. The search results are aggregated and the ranked results of similar code is reported.

Preliminary Results: We implemented the proposed ISiCS framework based on Elasticsearch [2], an open-source high performance distributed search engine. The current implementation consists of single search index of token-based normalised *n*-grams. The ISiCS search index has four parameters: size of grams in *n*-gram, Java normalisation options, information retrieval similarity functions, and parameters of the IR similarity functions. The complete set of parameters and their possible values are shown in Table II. To obtain the highest precision for similar code search, we tuned the system by searching for the optimal set of configuration for these four parameters. The tuning dataset was extended from the experiment of pervasively modified source code [36].

The results of optimised configuration for each of the six IR similarity functions are listed in Table III. We selected the precision-at-*n* as the error measure [31] since the widely used precision and recall are difficult to measure over large datasets without a ground truth. We found that the best configuration with the highest precision-at-10 of 0.938 is by using Divergence From Randomness (DFR) similarity function [4] with inverse term frequency (*if*) as the basic model of information content, Laplace's law of succession (*l*) as the first normalisation of information gain, and uniform distribution of term frequency

TABLE II
PARAMETERS OF ISICS

Parameter	Values
Size of grams	2,3,4,5
Java normalisation	d (data type), j (Java class), k (Java keyword), p (Java package), s (string), w (word)
IR similarity functions	TFIDF, BM25, DFR, IB, LMDirichlet, LMJelinekMercer
IR sim fnc. parameters	Varied based on each specific sim func.

TABLE III
THE BEST SETTINGS OF ISICS AGAINST PERVASIVELY MODIFIED CODE BASED ON PRECISION-AT-10.

IR sim fnc.	Parameter	Value	Norm.	*n*-gram	Prec@10
DFR	basic_model after_effect normalization	if l h1	kw	2-gram	0.938
LMJelinekMercer	lambda	0.9	p	3-gram	0.927
TF-IDF	disc_overlap	true,false	k,dkp	3-gram	0.908
BM25	k1 b disc_overlap	0.0 .. 2.4 0.0 .. 1.0 true,false	kp,dkp	3-gram	0.908
IB	distribution lambda normalization	ll ttf h1	kp,dkp	3-gram	0.907
	distribution lambda normalization	spl df h1	none	3-gram	
LMDirichlet	mu	500,750	d, none	3-gram	0.898

(*h1*) for the second normalisation. The best code abstraction for this DFR similarity is *n*-grams of normalised Java keyword and word tokens (*kw*) with gram size of 2.

V. FUTURE WORK

A. Evaluation and Improvements of ISiCS

ISiCS system with the derived optimised settings will be evaluated by benchmarking with state-of-the-art clone detection tools such as CCFinderX [21], NiCad [37], and scalable code clone/search tools [25], [34], [38]. The comparison will be done both in terms of scalability, i.e. time to create index and search on large-scale code corpus, and precision of finding similar code artefacts.

Furthermore, we plan to incorporate the variable-length gram (VGRAM) technique [30] as an improvement to the existing fixed-length gram. Source code statements are usually varied in lengths. Thus, flexibility of gram sizes in VGRAM may capture code sequence better than traditional fixed-size *n*-gram. The framework will be more generalised since finding the proper size of *n* is no longer needed. We also plan to add more indexes to capture code similarity at different code abstraction levels. The VGRAM-based framework with multiple indexes will be evaluated again against its original implementation and related tools.

B. Applications of the Code Search Framework

We plan to demonstrate applications of ISiCS framework with an empirical study of online code reuse between online code corpora and open source projects. Nowadays, programmers search and use code examples from Q&A websites (e.g. Stackoverflow [33]) in their software development. They also integrate libraries from online repositories (e.g. GitHub) into their projects. The study aim to (1) detect code propagation between online code corpora and open source projects, and (2) investigate software licensing issues caused by reuse of online source code.

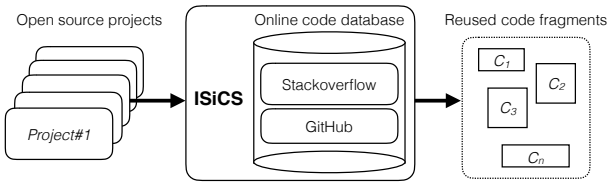

Fig. 3. Searching for online code reuse in open source projects using ISiCS

The corpora can contain incomplete code fragments and these fragments may also be pervasively modified to fit in the new environments. ISiCS framework is a suitable tool due to both its scalability and its flexibility of code similarity measurement. Online code artefacts will be stored in the search index and queried by source code from open source projects. The framework then reports pairs of reused code fragments for further analysis. The process is illustrated in Figure 3.

VI. CONCLUSION

Source code similarity is a fundamental technique in software engineering underlying code cloning, software plagiarism, similar code search, and software licensing conflicts. This research focuses on an empirical evaluation of current state-of-the-art code similarity tools and techniques, and framework for searching similar code over large-scaled corpora.

With presence of pervasive modifications, we found that specialised code similarity analysers such as clone detectors and plagiarism detectors outperform general similarity measurements provided by string similarity or compression-based tools. We also discovered the optimised parameter settings of the 30 selected tools from our study. Normalisation by compilation and decompilation is observed to be effective against pervasive modifications with six tools reporting no false classification.

The first version of Internet-scaled similar code search (ISiCS) framework has been implemented. We searched for its optimal configurations against pervasively modified source code based on precision-at-10 error measure. We plan to enhance the system by incorporating VGRAM and multi-representation techniques, then evaluate the system with state-of-the-art tools. The framework will finally be put to use in an empirical software engineering study of online code reuse.

VII. ACKNOWLEDGEMENTS

The author's PhD study is fully supported by scholarships from the Faculty of ICT, Mahidol University and Mahidol University Academic Development program. The author is grateful to be a member of Centre for Research on Evolution, Search and Testing (CREST), Computer Science department, University College London and supervised by Dr. Jens Krinke and Dr. David Clark. The ISiCS project is awarded Microsoft Azure Research Award (CRM:0518332).

REFERENCES

[1] BlackDuck OpenHub. http://code.openhub.net, 2016. Online; access 18-May-2016.

[2] Elasticsearch. https://www.elastic.co/products/elasticsearch, 2016. Online; access 25-Jun-2016.

[3] Oracle America, Inc. v. Google Inc., No. 3:2010cv03561 - Document 642 (N.D. Cal. 2011). http://law.justia.com/cases/federal/district-courts/FSupp2/259/597/2362960, 2016. Online; access 23-Apr-2016.

[4] G. Amati and C. J. Van Rijsbergen. Probabilistic models of information retrieval based on measuring the divergence from randomness. *ACM Transactions on Information Systems*, 20(4):357–389, Oct 2002.

[5] I. D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier. Clone detection using abstract syntax trees. In *International Conference on Software Maintenance (ICSM'98)*, pages 368–377, 1998.

[6] 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), 2007.

[7] S. Burrows, S. M. M. Tahaghoghi, and J. Zobel. Efficient plagiarism detection for large code repositories. *Software: Practice and Experience*, 37(2):151–175, Feb. 2007.

[8] G. Cosma and M. Joy. Towards a definition of source-code plagiarism. *IEEE Transactions on Education*, 51(2):195–200, May 2008.

[9] C. Daniela, P. Navrat, B. Kovacova, and P. Humay. The issue of (software) plagiarism: A student view. *IEEE Transactions on Education*, 55(1):22–28, Feb. 2012.

[10] J. Davies, D. M. German, M. W. Godfrey, and A. Hindle. Software Bertillonage: Determining the provenance of software development artifacts. *Empirical Software Engineering*, 18:1195–1237, 2013.

[11] Z. Duric and D. Gasevic. A source code similarity system for plagiarism detection. *The Computer Journal*, 56(1):70–86, Mar. 2012.

[12] D. M. German, M. Di Penta, Y. G. Gueheneuc, and G. Antoniol. Code siblings: Technical and legal implications of copying code between applications. In *Working Conference on Mining Software Repositories (MSR'09)*, pages 81–90, 2009.

[13] D. Gitchell and N. Tran. Sim: a utility for detecting similarity in computer programs. In *Technical symposium on Computer Science Education (SIGCSE '99)*, pages 266–270, 1999.

[14] N. Göde and R. Koschke. Incremental clone detection. In *Software Maintenance and Reengineering (CSMR'09)*, pages 219–228, 2009.

[15] J. Harder and N. Göde. Cloned code: stable code. *Journal of Software: Evolution and Process*, 25(10):1063–1088, 2013.

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

[17] M. Joy, N. Griffiths, and R. Boyatt. The BOSS online submission and assessment system. *ACM Journal on Educational Resources in Computing*, 5(3), 2005.

[18] M. Joy and J. E. Sinclair. Student perspectives on source-code plagiarism. *International Journal for Educational Integrity*, 9:3–19, 2012.

[19] M. Joy, J. E. Sinclair, R. C. Boyatt, J. Y. Yau, and G. Cosma. Student Perspectives on Plagiarism in Computing. In *International Plagiarism Conference*, pages 16–18, 2012.

[20] E. Juergens, F. Deissenboeck, B. Hummel, and S. Wagner. Do code clones matter? In *International Conference on Software Engineering (ICSE'09)*, pages 485–495, 2009.

[21] 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.

[22] C. Kapser and M. Godfrey. "Cloning Considered Harmful" Considered Harmful. In *Working Conference on Reverse Engineering (WCRE'06)*, pages 19–28, 2006.

[23] Y. Ke, K. T. Stolee, C. L. Goues, and Y. Brun. Repairing Programs with Semantic Code Search. In *International Conference onAutomated Software Engineering (ASE'15)*, pages 295–306, 2015.

[24] I. Keivanloo, J. Rilling, and Y. Zou. Spotting working code examples. In *International Conference on Software Engineering (ICSE'14)*, pages 664–675, 2014.

[25] I. Keivanloo, L. Roostapour, P. Schugerl, and J. Rilling. SE-CodeSearch: A scalable Semantic Web-based source code search infrastructure. In *International Conference on Software Maintenance (ICSM'10)*, pages 1–5, 2010.

[26] M. Kim, K. Whang, J. Lee, and M. Lee. n-Gram / 2L : A Space and Time Efficient Two-Level n-Gram Inverted Index Structure. In *International Conference on Very Large Data Bases (VLDB '05)*, pages 325–336, 2005.

[27] R. Komondoor and S. Horwitz. Using slicing to identify duplication in source code. In *International Symposium of Static Analysis (SAS'01)*, pages 40–56, 2001.

[28] J. Krinke. Identifying similar code with program dependence graphs. In *Working Conference on Reverse Engineering (WCRE'01)*, pages 301–309, 2001.

[29] J. Krinke, N. Gold, Y. Jia, and D. Binkley. Cloning and copying between GNOME projects. In *Working Conference on Mining Software Repositories (MSR'10)*, pages 98–101, 2010.

[30] C. Li, B. Wang, and X. Yang. VGRAM : Improving Performance of Approximate Queries on String Collections Using Variable-Length Grams. In *International Conference on Very Large Data Bases (VLDB'07)*, pages 303–314, 2007.

[31] C. D. Manning, P. Raghavan, and H. Schutze. *An Introduction to Information Retrieval*. 2009.

[32] 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 *International Conference on Program Comprehension (ICPC'11)*, pages 242–245, 2011.

[33] 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 *International Conference on Software Maintenance (ICSM'12)*, 2012.

[34] J. Park, M. Lee, J. Roh, S. Hwang, and S. Kim. Surfacing code in the dark: an instant clone search approach. *Knowledge and Information Systems*, 41:727–759, 2014.

[35] L. Prechelt, G. Malpohl, and M. Philippsen. Finding Plagiarisms among a Set of Programs with JPlag. *Journal Of Universal Computer Science*, 8(11):1016–1038, 2002.

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

[37] C. K. Roy and J. R. Cordy. NICAD: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization. In *International Conference on Program Comprehension (ICPC'08)*, pages 172–181, 2008.

[38] H. Sajnani, V. Saini, J. Svajlenko, C. K. Roy, and C. V. Lopes. SourcererCC: Scaling Code Clone Detection to Big Code. In *International Conference on Software Engineering (ICSE'16)*, pages 1157–1168. ACM Press, 2016.

[39] S. Schleimer, D. S. Wilkerson, and A. Aiken. Winnowing: Local algorithms for document fingerprinting. In *International Conference on Management of Data/Principles of Database Systems (SIGMOD'03)*, 2003.

[40] G. Whale. Identification of program similarity in large populations. *The Computer Journal*, 33(2):140–146, Feb. 1990.

[41] M. J. Wise. Detection of similarities in student programs. In *Technical symposium on Computer science education (SIGCSE'92)*, 1992.

[42] M. J. Wise. YAP3: Improved detection of similarities in computer program and other texts. In *Technical symposium on Computer science education (SIGCSE'96)*, pages 130–134, 1996.