Clone Notifier: Developing and Improving the System to Notify Changes of Code Clones

Shogo Tokui*, Norihiro Yoshida†, Eunjong Choi‡, and Katsuro Inoue*
*Osaka University, Japan, {s-tokui, inoue}@ist.osaka-u.ac.jp
†Nagoya University, Japan, yoshida@ertl.jp
‡Kyoto Institute of Technology, Japan, echoi@kit.ac.jp

***Abstract*—A code clone is a code fragment that is identical or similar to it in the source code. It has been identified as one of the main problems in software maintenance. When a developer fixes a defect, they need to find the code clones corresponding to the code fragments. In this paper, we present Clone Notifier, a system that alerts on creations and changes of code clones to software developers. First, Clone Notifier identifies creations and changes of code clones. Subsequently, it groups them into four categories (new, deleted, changed, stable) and assigns labels (e.g., consistent, inconsistent) to them. Finally, it notifies on creations and changes of code clones along with the corresponding categories and labels. Clone Notifier and its video are available at: https://github.com/s-tokui/CloneNotifier.**

***Index Terms*—code clone, tracking clone, software maintenance**

I. Introduction

A code clone is a code fragment that is identical or similar to it in the source code. It has been identified as one of the main problems in software maintenance [1]. When a developer fixes a defect, they have to find the code clones corresponding to the code fragment.

Simultaneous modification is needed in maintenance for clone sets (i.e., sets of code clones in which all pairs of the code clones are identical or like each other) [2]. However, consistent modification of all code clones is an error-prone task. Therefore, tool-based management is required for simultaneous modification of code clones.

Our research team has developed a Clone Notifier, which enables consistent modification of code clones by automatically alerting developers of changes in the information of code clones [3]. Clone Notifier detects code clones using a token-based code clone detector named CCFinderX [4]. Furthermore, it provides the change information (i.e., changed, deleted, new, and stable) of clone sets based on the changes between two revisions. Clone Notifier continuously detects the change information of code clones and sends an e-mail about the change to the developers.

However, it is difficult for developers to maintain many code clones suggested by Clone Notifier consistently. Specifically, there are many inconsistent changes to code clones, and some of them contain bugs [5], [6], [7]. To mitigate this problem, we improved the Clone Notifier in the following four ways to show developers the areas where they should focus (e.g., inconsistent changes).

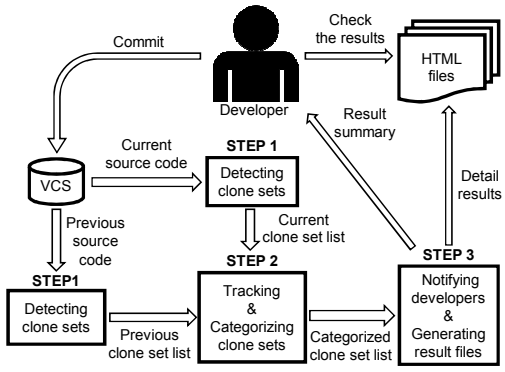

Fig. 1. An overview of Clone Notifier

- Classification of changed clone sets into refined categories
- Addition of two code clone detectors SourcererCC [8] and CCVolti [9]
- Change of the definition of a clone set
- Improving the method of tracking of code clones [10]

These improvements make it possible to notify developers of inconsistent changes.

In this paper, we demonstrate how Clone Notifier supports developers, such that they can fix defects in clone sets consistently. In this paper, we illustrate the use of Clone Notifier using open source software PostgreSQL[1] from 23rd June 2018 to 22nd June 2019.

II. Clone Notifier: Overview

Clone Notifier is a system used to generate HTML files with the change information of code clones from the source code of two revisions as shown in Figure 1. Moreover, Clone Notifier notifies developers of the summary of the change information. It performs three steps: 1) detecting the clone sets in each revision, 2) tracking and categorizing clone sets between two revisions, and 3) notifying developers of the change information of code clones.

A. Detect Code Clones

Clone Notifier defines a clone set as a set of code clones in which all pairs of code clones are identical or similar to

[1]https://github.com/postgres/postgres

TABLE I
CATEGORIES OF CLONE SETS

Stable clone sets that were not changed.	**New clone sets** that did not exist in the previous version but the current version.
Deleted clone sets that existed in the previous version and removed.	**Changed clone sets** that include edited, added, or deleted code clones.

TABLE II
LABELS OF CHANGED CLONE SETS

Label	Definition
Add	An added code clone exists.
Subtract	A deleted code clone exists.
Shift	A code clone which is moved from another clone sets exists.
Consistent	All code clones were done same edit.
Inconsistent	At the same time, an edited code clone and a stable code clone exist.

each other. Code clone detectors output clone pairs (pairs of code clones) from each revision of source code. Clone Notifier detects the clone pairs in the source code using the three code clone detectors SourcererCC [8], CCFinderX [4] and CCVolti [9]. The supported languages of Clone Notifier depend on the code clone detectors.

After detecting the clone pairs, Clone Notifier constructs clone sets from the clone pairs. When the syntax code clone detector (e.g., CCFinderX), detects two clone pairs (c_1, c_2) and (c_2, c_3), the clone c_1 is exactly similar to the clone c_3 because the relationship preserves the transitive property. When the near-miss code clone detector (e.g. SourcererCC and CCVolti), detects two clone pairs (c_1, c_2) and (c_2, c_3), clone c_1 may be different from clone c_3.

When developers use the near-miss code clone detector, Clone Notifier constructs clone sets from clone pairs of JGraphT[2] for solving the maximal clique problem. A clique is a subset of vertices of an undirected graph, such that edges exist between two different vertices in the clique. A maximal clique is a clique that does not exist exclusively within the vertex set of a larger clique. For example, there are four code fragments c_1, c_2, c_3, c_4. When detecting the four clone pairs (c_1, c_2), (c_2, c_3), (c_3, c_1) and (c_3, c_4), Clone Notifier detects (c_1, c_2, c_3) and (c_3, c_4) as the clone sets. As clone sets are maximal cliques of JGraphT, clone sets can be included in code clones that may need simultaneous modification without a surplus or deficiency.

B. Track and Categorize Clone Sets

After detecting clone sets, Clone Notifier tracks from the clone sets in the old revision to the clone sets in the new revision with the position of the code fragments in each source code. To track code clones between two revisions, Clone Notifier calculates the overlapping location of code clones,

[2]https://jgrapht.org/

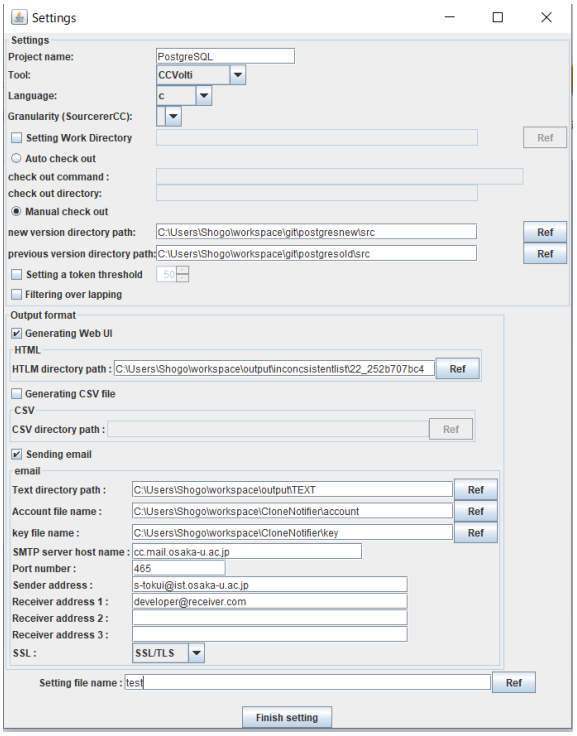

Fig. 2. Configuration GUI

based on the location overlapping function of Kim et al. [10]. Location overlapping measures how much two locations $l1$ and $l2$ overlap each other $(0 \leq LO(l1, l2) \leq 1)$. Clone Notifier uses the difference between files of the same name in each revision, without the added and deleted lines. It computes the relative proportion of an overlapped region between $l1$ and the calibrated $l2$.

$$LO(l1, l2) = \frac{min(n_e, o_e) - max(n_s, o_s)}{n_e - n_s} \quad (1)$$

where $l1$ in the old revision spans from the line o_s to the line o_e, and the calibrated location of $l2$ in the new revision spans from the line n_s to the line n_e. If the location overlapping between the two code clones is 70% or more, Clone Notifier tracks from the code clone at the old revision to the code clone at the new revision. If a code clone exists only in either revision, it is defined as a deleted/added code clone or is included as a subtract/add clone set, as shown in Table II. The condition to track a clone set in the new revision is that the clone set must contain the largest number of code clones tracked from the code clones of the clone set in the old revision.

After tracking the clone sets, Clone Notifier classifies the clone sets into four categories (stable, changed, new, and deleted) as shown in Table I [3]. According to the type of editing of the code clones included in the changed clone set, Clone Notifier gives the changed clone sets any of five labels as shown in Table II. The labels are *add*, *subtract*, *shift*,

Project information	
File information	
Total files	1086
Add files	0
Deleted files	0
files contaning clones	595
Clone set categories information	
Total clone sets	2745
Stable clone sets	2744
Changed clone sets	1
New clone sets	0
Deleted clone sets	0
Code clone information	
Total code clones	9470
Stable code clones	9469
Modified code clones	1
Moved code clones	0
Added code clones	0
Deleted code clones	0
Deleted and modified code clones	0

Fig. 3. Home page

Changed Clone Set

Clone set 0 (inconsistent)			
ID	Category	File name	Line
0.0	STABLE	backend¥utils¥cache¥lsyscache.c	775.0-795.0
0.1	MODIFIED	backend¥utils¥cache¥lsyscache.c	836.0-852.0
0.2	STABLE	backend¥utils¥cache¥lsyscache.c	861.0-879.0

Fig. 4. Clone set information

consistent, and *inconsistent* [10]. Clone Notifier can notify the developers of the clone set information in more detail.

C. Notify Developers and Show the Change Information

After categorizing the clone sets, Clone Notifier outputs the result file about the change information of the clone sets and sends an e-mail to developers. The extension of the result file is HTML. If developers set up their e-mail information, they can receive an e-mail consistently with a summary of the results and a link to the result file. The e-mail messages are a summary of the results, which have four pieces of information: number of clone sets, number of code clones, number of files containing code clones, and a link to the home page for the result in a browser. Developers can analyze the change information of the code clones.

Figure 3 shows the home page of the result files in a browser. Developers click the 'Changed clone set' and check the clone setlist page to examine the details of the changed clone sets. In Figure 4, the clone setlist page shows the details information of the clone sets. The several pieces of information are the labels, categories of the code clones shown in section II-B, file containing the code clone, and line number of the code clone in the new revision. Notably, the clone sets are sorted by the same label, and the inconsistent changes in the clone sets are at the top of the list. Developers click the ID of the code clone 0.0 on the clone setlist page and check the source code page. As shown in Figures 5, 6 and 7, the source code pages are written based on the difference between two revisions, and developers can easily check how the code clone changed.

III. Illustrative Usage Scenario

In this section, we demonstrate a usage instance of Clone Notifier to detect the inconsistent change clone sets. Moreover, we examine the results to uncover the code clone which should be fixed.

A. Usage Scenario

First, the developer sets the configurations of the Clone Notifier. After downloading Clone Notifier, the developer executes setting.jar and write (e.g., the code clone detection tool, two versions of the directory path, email address, and configuration file name) to generate the configuration file. Figure 2 shows the input form of the configuration settings. Next, the developer executes Clone Notifier with the configuration file as an argument. Finally, after completing the execution, the developer receives an email from Clone Notifier with a summary of the results. In this use case, one inconsistent changed clone set has been detected. To investigate whether any defect exists in the code clone, the developer accesses the URL written in the email and checks the results.

When accessing the URL, the home page is displayed, as first shown in Figure 3. If the developer clicks a clone set category name, he can check the detailed information about that category. On the Clone Set page, as shown in Figure 4, the developer can confirm the change information of the code clone for each clone set. The information includes whether the code clone has been modified, is containing a file, and the line location in the source code. When the developer clicks the code clone ID, he can see more detailed information about changes. In the source code page, as shown in Figures 5, 6, and 7, the code clone in the source code is displayed. If changes have been made, the changes are colored.

B. Findings

In this section, we illustrate three instances of inconsistent changes detected by the Clone Notifier.

The first instance of inconsistent change on 29th Dec 2018 is the clone set of three code fragments, as shown in Figure 5. These code fragments are in the same file. Although, it was necessary to refactor the other two same code clones, the modified one was refactored. The commit message includes that *the modified coding is too convoluted and hard to follow*. Approximately two weeks before this commit, the committer discussed with PostgreSQL developers and improved the readability of this code by e-mail[3]. When a code clone in a clone set was very convoluted, the other code clones are considered to be convoluted. Thus, they should be identified as refactoring candidates.

The second inconsistent change on 5th Apr 2019 is the clone set of three code fragments, as shown in Figure 6. These code

[3]https://www.postgresql.org/message-id/20181206222221.g5witbsklvqthjll@alvherre.pgsql

Stable Code Clone — File: src/backend/executor/execMain.c

```
2097    {
2098            TupleDesc       old_tupdesc = RelationGetDescr(rel);
2099            AttrNumber *map;
2100
2101            rel = resultRelInfo->ri_PartitionRoot;
2102            tupdesc = RelationGetDescr(rel);
2103            /* a reverse map */
2104            map = convert_tuples_by_name_map_if_req(old_tupdesc,
2105                                                    tupdesc,
2106                                                    gettext_noop("could not convert row type"));
2107
2108            /*
2109             * Partition-specific slot's tupdesc can't be changed,
2110             * so allocate a new one.
2111             */
2112            if (map != NULL)
2113                    slot = execute_attr_map_slot(map, slot,
2114                                                 MakeTupleTableSlot(tupdesc, &TTSOpsVirtual));
2115    }
```

Modified Code Clone — File: src/backend/executor/execMain.c

```
1852        {
1853    +       TupleDesc       old_tupdesc;
        -       TupleDesc       old_tupdesc = RelationGetDescr(rel);
1854            AttrNumber *map;
1855
1856    +       root_relid = RelationGetRelid(resultRelInfo->ri_PartitionRoot);
1857    +       tupdesc = RelationGetDescr(resultRelInfo->ri_PartitionRoot);
1858    +
1859    +       old_tupdesc = RelationGetDescr(resultRelInfo->ri_RelationDesc);
        -       rel = resultRelInfo->ri_PartitionRoot;
        -       tupdesc = RelationGetDescr(rel);
1860            /* a reverse map */
1861            map = convert_tuples_by_name_map_if_req(old_tupdesc, tupdesc,
1862                                                    gettext_noop("could not convert row type"));
1863
1864            /*
1865             * Partition-specific slot's tupdesc can't be changed, so allocate a
1866             * new one.
1867             */
1868            if (map != NULL)
1869                    slot = execute_attr_map_slot(map, slot,
1870                                                 MakeTupleTableSlot(tupdesc, &TTSOpsVirtual));
1871        }
```

Fig. 5. Inconsistent change (previoius commit ID: f7ea1a4233, current commit ID: e8b0e6b82d)

Stable Code Clone — File: src/backend/utils/cache/lsyscache.c

```
775     get_attname(Oid relid, AttrNumber attnum, bool missing_ok)
776     {
777             HeapTuple       tp;
778
779             tp = SearchSysCache2(ATTNUM,
780                                  ObjectIdGetDatum(relid), Int16GetDatum(attnum));
781             if (HeapTupleIsValid(tp))
782             {
783                     Form_pg_attribute att_tup = (Form_pg_attribute) GETSTRUCT(tp);
784                     char       *result;
785
786                     result = pstrdup(NameStr(att_tup->attname));
787                     ReleaseSysCache(tp);
788                     return result;
789             }
790
791             if (!missing_ok)
792                     elog(ERROR, "cache lookup failed for attribute %d of relation %
793                          attnum, relid);
794             return NULL;
795     }
```

Modified Code Clone — File: src/backend/utils/cache/lsyscache.c

```
836     get_attgenerated(Oid relid, AttrNumber attnum)
837     {
838             HeapTuple       tp;
839     +       Form_pg_attribute att_tup;
840     +       char            result;
841
842             tp = SearchSysCache2(ATTNUM,
843                                  ObjectIdGetDatum(relid),
844                                  Int16GetDatum(attnum));
845     +       if (!HeapTupleIsValid(tp))
        -       if (HeapTupleIsValid(tp))
        -       {
        -               Form_pg_attribute att_tup = (Form_pg_attribute) GETSTRUCT(tp);
        -               char            result;
        -
        -               result = att_tup->attgenerated;
        -               ReleaseSysCache(tp);
        -               return result;
        -       }
        -       else
846                     elog(ERROR, "cache lookup failed for attribute %d of relation %u",
847                          attnum, relid);
848     +       att_tup = (Form_pg_attribute) GETSTRUCT(tp);
849     +       result = att_tup->attgenerated;
850     +       ReleaseSysCache(tp);
851     +       return result;
852     }
```

Fig. 6. Inconsistent change (previoius commit ID: 82150a05be, current commit ID: edda32ee25)

fragments are in the same file. Although, it was necessary to refactor the other two same code clones, the modified one of these code fragments was refactored, such that the condition statement changes to the condition negative form. The commit message indicates that *"the developer rewrites get_attgenerated() to avoid compiler warning if the compiler does not recognize that error log does not return"*. Therefore, to avoid the compiler warning with the other code clones in the clone set, these code clones should be modified with the same change. If Clone Notifier is constantly used, the developer can consistently refactor.

The third inconsistent change on 17th Apr 2019 is the clone set of two code fragments, as shown in Figure 7. These code fragments are in the same file. One of these code fragments was refactored, such that it added a NULL return if *DataChecksumsEnabled* is false. Although, it was necessary to refactor the other code clone in the same way as the refactored code fragment. The commit message includes that *'returning 0 could falsely indicate that there is no problem, but returning NULL correctly indicates that there is no information about potential problems'*.

IV. Related Work

Our research team has developed CCEvovis [11], which is a system that visualizes the evolution of code clones [3]. It highlights and visualizes the clone change for developers to understand. Saha et al. described the design and implementation of a near-miss clone genealogy extractor, gCad

Stable Code Clone

File: src/backend/utils/adt/pgstatfuncs.c

```
1365    pg_stat_get_db_stat_reset_time(PG_FUNCTION_ARGS)
1366    {
1367            Oid                     dbid = PG_GETARG_OID(0);
1368            TimestampTz result;
1369            PgStat_StatDBEntry *dbentry;
1370
1371            if ((dbentry = pgstat_fetch_stat_dbentry(dbid)) == NULL)
1372                    result = 0;
1373            else
1374                    result = dbentry->stat_reset_timestamp;
1375
1376            if (result == 0)
1377                    PG_RETURN_NULL();
1378            else
1379                    PG_RETURN_TIMESTAMPTZ(result);
1380    }
```

Modified Code Clone

File: src/backend/utils/adt/pgstatfuncs.c

```
1542    pg_stat_get_db_checksum_last_failure(PG_FUNCTION_ARGS)
1543    {
1544            Oid                     dbid = PG_GETARG_OID(0);
1545            TimestampTz result;
1546            PgStat_StatDBEntry *dbentry;
1547  +
1548  +         if (!DataChecksumsEnabled())
1549  +                 PG_RETURN_NULL();
1550
1551            if ((dbentry = pgstat_fetch_stat_dbentry(dbid)) == NULL)
1552                    result = 0;
1553            else
1554                    result = dbentry->last_checksum_failure;
1555
1556            if (result == 0)
1557                    PG_RETURN_NULL();
1558            else
1559                    PG_RETURN_TIMESTAMPTZ(result);
1560    }
```

Fig. 7. Inconsistent change (previoius commit ID: 9010156445, current commit ID: 252b707bc4)

[12], which can extract and classify both exact and near-miss clone genealogies. In this paper, we described the Clone Notifier that constantly notifies about the inconsistent changes, including the changes in the information of the code clones.

Other studies have existed on detection code clones [13], [14], [15], and code clones evolution [16], [17], [18]. Also, Mondai et al. have found that the percentage of severe bugs is significantly higher in micro-clones than regular clones [19], [20].

V. Summary

In this paper, we presented a Clone Notifier that automatically alerts about changes in the information of code clones. We demonstrated the use of Clone Notifier to detect such changes. We found using improved Clone Notifier that there were code clones that did not change consistently in the changed clone set. And, we have future work to implement incremental clone detection because the detection tools used by Clone Notifier detect code clones for each revision, and this treatment costs a lot for large software projects.

Acknowledgements

This work was supported by JSPS KAKENHI Grant Numbers JP18H04094, JP19K20240. We would like to thank Editage (www.editage.com) for English language editing.

References

[1] C. K. Roy, J. R. Cordy, and R. Koschke, "Comparison and evaluation of code clone detection techniques and tools," *Science of Computer Programming*, vol. 74, no. 7, pp. 470–495, 2009.

[2] M. Fowler, *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 2018.

[3] Y. Yamanaka, E. Choi, N. Yoshida, K. Inoue, and T. Sano, "Applying clone change notification system into an industrial development process," in *Proc. ICPC*, 2013, pp. 199–206.

[4] 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*, vol. 18, no. 7, pp. 654–670, 2002.

[5] K. Inoue, Y. Higo, N. Yoshida, E. Choi, S. Kusumoto, K. Kim, W. Park, and E. Lee, "Experience of Finding Inconsistently-Changed Bugs in Code Clones of Mobile Software," in *Proc. IWSC*. IEEE, 2012, pp. 94–95.

[6] L. Jiang, Z. Su, and E. Chiu, "Context-based detection of clone-related bugs," in *Proc. ESEC*. ACM, 2007, pp. 55–64.

[7] E. Juergens, F. Deissenboeck, B. Hummel, and S. Wagner, "Do Code Clones Matter?" in *Proc. ICSE*. IEEE, 2009, pp. 485–495.

[8] H. Sajnani, V. Saini, J. Svajlenko, C. K. Roy, and C. V. Lopes, "SourcererCC: Scaling code clone detection to big-code," in *Proc. ICSE*, 2016, pp. 1157–1168.

[9] K. Yokoi, E. Choi, N. Yoshida, and K. Inoue, "Investigating vector-based detection of code clones using bigclonebench," in *Proc. APSEC*, 2018, pp. 699–700.

[10] M. Kim, V. Sazawal, D. Notkin, and G. Murphy, "An empirical study of code clone genealogies," in *Proc. ESEC/FSE*, 2005, pp. 187–195.

[11] H. Honda, S. Tokui, K. Yokoi, E. Choi, N. Yoshida, and K. Inoue, "CCEvovis: A Clone Evolution Visualization System for Software Maintenance," in *Proc. ICPC*, 2019, pp. 122–125.

[12] R. K. Saha, C. K. Roy, and K. A. Schneider, "gCad: A Near-Miss Clone Genealogy Extractor to Support Clone Evolution Analysis," in *Proc. ICSM*, 2013, pp. 488–491.

[13] F. Farmahinifarahani, V. Saini, D. Yang, H. Sajnani, and C. V. Lopes, "On precision of code clone detection tools," in *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2019, pp. 84–94.

[14] L. Büch and A. Andrzejak, "Learning-based recursive aggregation of abstract syntax trees for code clone detection," in *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2019, pp. 95–104.

[15] C. K. Roy and J. R. Cordy, "Benchmarks for software clone detection: A ten-year retrospective," in *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2018, pp. 26–37.

[16] E. Duala-Ekoko and M. P. Robillard, "Clonetracker: tool support for code clone management," in *Proceedings of the 30th international conference on Software engineering*. ACM, 2008, pp. 843–846.

[17] G. Zhang, X. Peng, Z. Xing, S. Jiang, H. Wang, and W. Zhao, "Towards contextual and on-demand code clone management by continuous monitoring," in *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2013, pp. 497–507.

[18] E. Duala-Ekoko and M. P. Robillard, "Clone region descriptors: Representing and tracking duplication in source code," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 20, no. 1, p. 3, 2010.

[19] J. F. Islam, M. Mondal, and C. K. Roy, "A comparative study of software bugs in micro-clones and regular code clones," in *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2019, pp. 73–83.

[20] M. Mondai, C. K. Roy, and K. A. Schneider, "Micro-clones in evolving software," in *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2018, pp. 50–60.