CCFinder: A Multilinguistic Token-Based Code Clone Detection System for Large Scale Source Code
Paper in journals: this is the first page of a paper published in *IEEE Transactions on Software Engineering*.
[*IEEE Transactions on Software Engineering*] **28**, 654-670 (2002)

CCFinder: A Multilinguistic Token-Base Code Clone Detection System for Large Scale Source Code

Toshihiro Kamiya, *Member, IEEE*, Shinji Kusumoto, *Member, IEEE*, and Katsuro Inoue, *Member, IEEE*

Abstract—A code clone is a code portion in source files that is identical or similar to another. Since code clones are believed to reduce the maintainability of software, several code clone detection techniques and tools have been proposed. This paper proposes a new clone detection technique, which consists of the transformation of input source text and a token-by-token comparison. For its implementation with several useful optimization techniques, we have developed a tool, named CCFinder, which extracts code clones in C, C++, Java, COBOL, and other source files. As well, metrics for the code clones have been developed. In order to evaluate the usefulness of CCFinder and metrics, we conducted several case studies where we applied the new tool to the source code of JDK, FreeBSD, NetBSD, Linux, and many other systems. As a result, CCFinder has effectively found clones and the metrics have been able to effectively identify the characteristics of the systems. In addition, we have compared the proposed technique with other clone detection techniques.

Index Terms—Code clone, duplicated code, CASE tool, metrics, maintenance.

1 Introduction

A code clone is a code portion in source files that is identical or similar to another. Clones are introduced because of various reasons such as reusing code by "copy-and-paste," mental macro (definitional computations frequently coded by a programmer in a regular style, such as payroll tax, queue insertion, data structure access, etc.), or intentionally repeating a code portion for performance enhancement, etc. [5]. A conservative and protective approach for modification and enhancement of a legacy system would introduce clones. Also, systematic generation of a set of slightly different code portions from a single basis will bear clones. Clones make the source files very hard to modify consistently. For example, let's assume that a software system has several clone subsystems created by duplication with slight modification. When a fault is found in one subsystem, the engineer has to carefully modify all other subsystems [15]. For a large and complex system, there are many engineers who take care of each subsystem and then modification becomes very difficult. If the existence of clones has been documented and maintained properly, the modification would be relatively easy; however, keeping all clone information is generally a laborious and expensive process. Various clone detection tools have been proposed and implemented [1], [2], [3], [4], [5], [7], [11], [14], [15], [17] and a number of algorithms for finding clones have been used for them, such as line-by-line matching for an abstracted source program [1] and similarity detection for metrics values of function bodies [17].

We were interested in applying a clone detection technique to a huge software system for a division of government, which consists of one million lines of code in 2,000 modules written in both COBOL and PL/I-like language, which was developed more than 20 years ago and has been maintained continually by a large number of engineers [18], [21]. It was believed that there would be many clones in the system, but the documentation did not provide enough information regarding the clones. It was considered that these clones heavily reduce maintainability of the system; thus, an effective clone detection tool has been expected.

Based on such initial motivation for clone detection, we have devised a clone detection algorithm and implemented a tool named CCFinder (Code clone finder). The underlying concepts for designing the tool were as follows:

- The tool should be industrial strength and be applicable to a million-line size system within affordable computation time and memory usage.
- A clone detection system should have the ability to select clones or to report only helpful information for user to examine clones since large number of clones is expected to be found in large software systems. In other words, the code portions, such as short ones inside single lines and sequence of numbers for table initialization, may be clones, but they would not be useful for the users. A clone detection system that removes such clones with heuristic knowledge improves effectiveness of clone analysis process.

- *T. Kamiya is with Functions and Configuration Group, RPESTO, JST. Graduate School of Engineering Science, Osaka University, 1-3 Machikaneyama-cho, Toyonaka, Osaka 560-8531, Japan. E-mail: kamiya@ics.es.osaka-u.ac.jp.*
- *S. Kusumoto and K. Inoue are with the Graduate School of Engineering Science, Osaka University, 1-3 Machikaneyama-cho, Toyonaka, Osaka 560-8531, Japan. E-mail: {kusumoto, inoue}@ics.es.osaka-u.ac.jp.*

Manuscript received 19 July 2000; revised 28 Mar. 2001; accepted 17 Sept. 2001.
Recommended for acceptance by L. Briand.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number 112550.

The following is a comment on the published paper shown on the preceding page.

A New Code Clone Detection Tool for Large Scale Source Code

INOUE Katsuro
(Graduate School of Information Science and Technology)

Introduction

A code clone is a code portion in source files that is identical or similar to another. Clones are introduced because of various reasons such as reusing code by 'copy-and-paste', etc [4]. Clones make the source files very hard to modify consistently. For example, let's assume that a software system has several clone subsystems created by duplication with slight modification. When a fault is found in one subsystem, the engineer has to carefully modify all other subsystems[7]. Various clone detection tools have been proposed and implemented [1] [2] [4].

In this paper, we have devised a clone detection algorithm and implemented a tool named CCFinder (Code clone finder). The underlying concepts for designing the tool were as follows.
(1) The tool should be industrial strength, and be applicable to a million-line size system within affordable computation time and memory usage. (2) A clone detection system should have ability to select clones or to report only helpful information for user to examine clones, since large number of clones is expected to be found in large software systems. (3) Renaming variables or editing pasted code after copy-and-paste makes a slightly different pair of code portions. These code portions have to be effectively detected. (4) The language dependent parts of the tool should be limited to a small size, and the tool has to be easily adaptable to many other languages.

Proposed clone-code detection technique

A clone relation is defined as an equivalence relation (i.e., reflexive, transitive, and symmetric relation) on code portions. A clone relation holds between two code portions if (and only if) they are the same sequences. For a given clone relation, a pair of code portions is called clone pair if the clone relation holds between the portions. An equivalence class of clone relation is called clone class. That is, a clone class is a maximal set of code portions in which a clone relation holds between any pair of code portions.

Clone detection is a process in which the input is source files and the output is clone pairs. The entire process of our token-based clone detecting technique is shown in **Figure 1**. The process consists of four steps:

(1) Lexical analysis

Each line of source files is divided into tokens corresponding to a lexical rule of the programming language. The tokens of all source files are concatenated into a single token sequence, so that finding clones in multiple files is performed in the same way as single file analysis. At this step, the white spaces (including \n and \t and comments) between tokens are removed from the token sequence, but those characters are sent to the formatting step to reconstruct the original source files.

(2) Transformation

The token sequence is transformed with sub-processes (2-1) and (2-2) described below. At the same time, the mapping information from the transformed token sequence into the original token sequences is stored for the formatting step which comes later.

(2-1) Transformation by the transformation rules

The token sequence is transformed, i.e., tokens are added, removed, or changed based on the transformation rules.

(2-2) Parameter replacement

After step 2-1 each identifier related to types, variables, and constants is replaced with a special token. This replacement

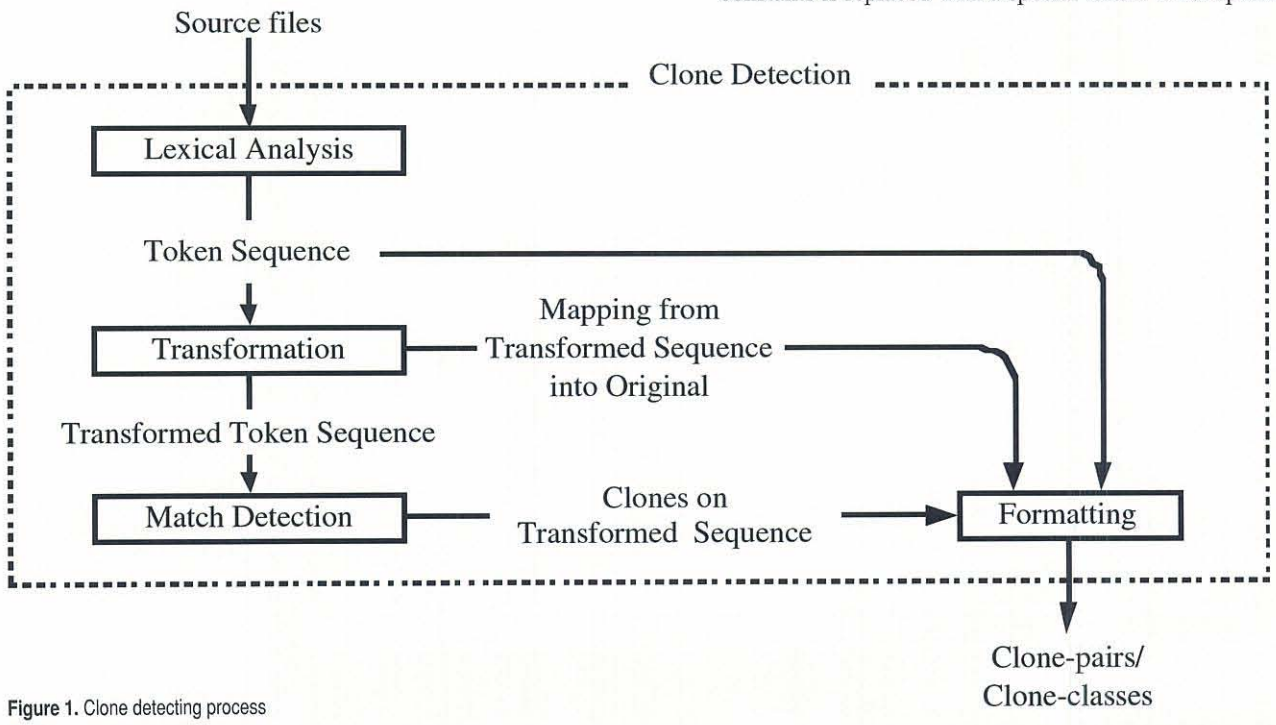

Figure 1. Clone detecting process

makes code-portions with different variable names to become clone pairs.

(3) Match Detection

From all the sub-strings on the transformed token sequence, equivalent pairs are detected as clone pairs. Each clone pair is represented as a quadruplet (LeftBegin, LeftEnd, RightBegin, RightEnd), where LeftBegin and LeftEnd are the beginning and termination positions (indices in the token sequence) of a leading clone, and RightBegin and RightEnd belong to another following clone for a clone pair.

(4) Formatting

Each location of clone pair is converted into line numbers on the original source files.

Tool CCFinder has been implemented in C++ and runs under Windows 95/NT 4.0 or later. CCFinder extracts clone classes from C, C++, Java, FORTRAN, LISP and COBOL source files. The tool receives the paths of source files, and writes the locations of the extracted clone classes to the standard output. CCFinder uses a suffix-tree algorithm [6] with both time and space complexities $O(m\,n)$, where m is the maximum length of involved clones and n is the total length of the source file. If we would naturally assume that m does not depend on n and it is bounded by some fixed length, the time and space complexities will practically be $O(n)$.

Case study

The purpose of the case studies was to evaluate our token-based clone-detecting technique and the metrics. The target source files were widely available files of 'industrial' size. In all the case studies, CCFinder was executed on a PC with Pentium III 650MHz and 640MB RAM, which seem to be moderate, non-special hardware specification for PC these days. In the following discussion we will use elapsed time on this PC.

JDK 1.3.0 [8] is a commonly used Java library, and the source files are publicly available. Tool CCFinder has been applied to all source files of JDK, about 570k lines in total, in 1877 files. It takes about 3 minutes for execution on the PC. **Figure 2** shows a scatter plot of the clone pairs having at least 30 same tokens (about 13 lines). Both the vertical and horizontal axes represent lines of source files. The files are sorted in alphabetical order of the file paths, so that files in the same directory are also located nearby on the axis. A clone pair is shown as a diagonal line segment. Only lines below the main diagonal are plotted. In **Figure 2**, each line segment looks like a dot since each clone pair is small (average 39, up to 628 lines) in comparison to the scale of the axis. Most line segments are located near the main diagonal line, and this means that most of the clones occur within a file or among source files at the near directories.

There are several crowded areas, marked *A*, *B*, *C*, *D*, and *E*.

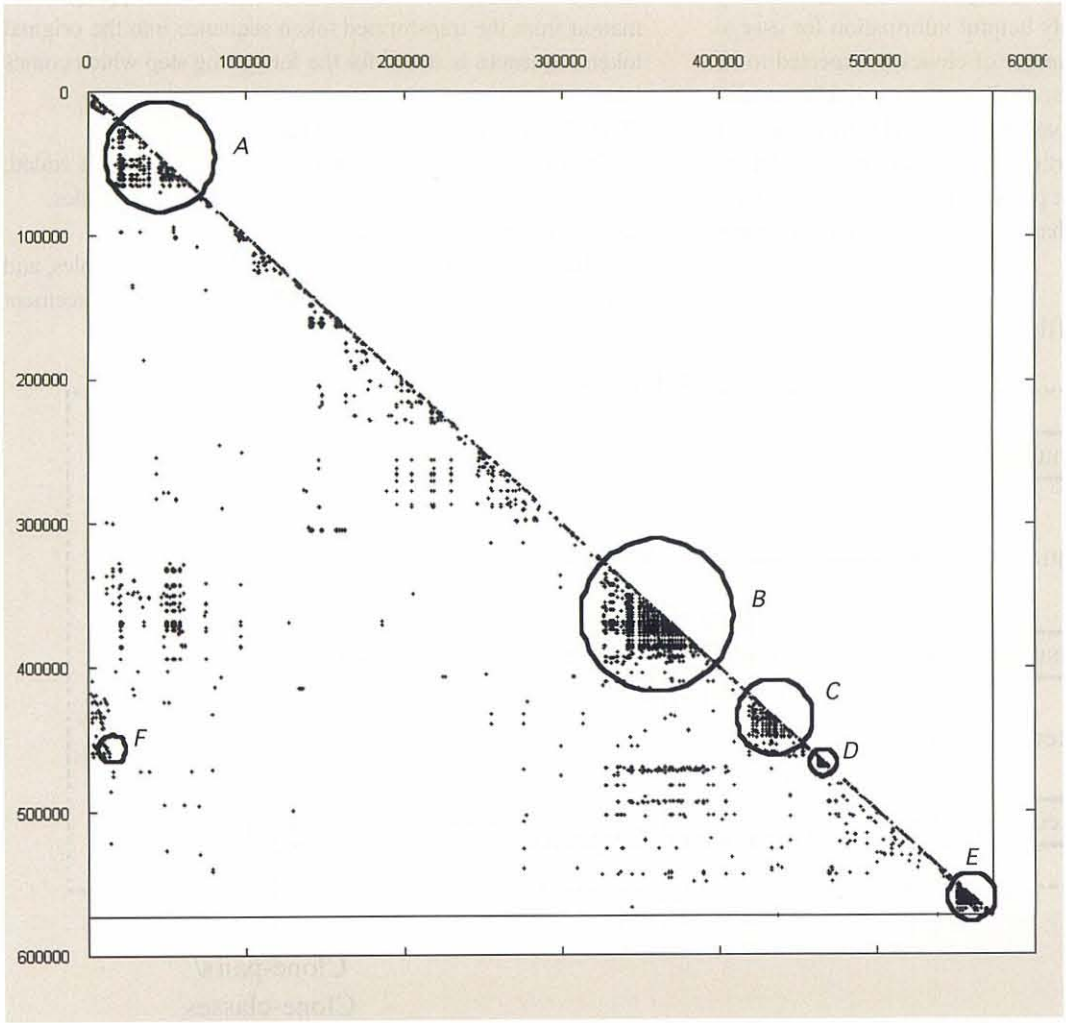

Figure 2.
Scatter plot for JDK 1.3.0

```
31|  */
 32| public class MultiButtonUI extends ButtonUI {
 33|

160|     public static ComponentUI createUI(JComponent a) {
161|         ComponentUI mui = new MultiButtonUI();
162|         return MultiLookAndFeel.createUIs(mui,
163|                                            ((MultiButtonUI) mui).uis,
164|                                            a);
165|     }
```

(a) MultiButtonUI.java

```
 31|  */
 32| public class MultiColorChooserUI extends ColorChooserUI {
 33|

160|     public static ComponentUI createUI(JComponent a) {
161|         ComponentUI mui = new MultiColorChooserUI();
162|         return MultiLookAndFeel.createUIs(mui,
163|                                            ((MultiColorChooserUI) mui).uis,
164|                                            a);
165|     }
```

(b) MultiColorChooserUI.java

These two files are identical except for three identifiers shown in bold style.

Figure 3. Example of clone found in JDK

Area *A* corresponds to source files of `java/awt/*.java`, *B*, *C*, and *D* to `javax/swing/*.java`, and *E* to `org/omg/CORBA/*.java`. *D* contains many 'clone files', that is, very similar source files. Some of them contain an identical class definition except for their different parent classes. **Figure 3** shows parts of the two files as examples, namely `MultiButtonUI.java` and `MultiColorChooserUI.java`. Differences are only in lines 32, 161, and 163. According to the comments of the source files, a code generator named `AutoMulti` has created these files. To modify them, the developer should obtain an automatic code generation tool called `AutoMulti` (it is not included in JDK), edit, and apply it correctly. If the developer does not have the tool, all the files have to be updated carefully by hand. In this case, these code portions have two different names : the base classes and the type of local variables named `mui`. Redesign techniques for Java are presented in [3] and might be applicable to this case. Also, using *generic type* for Java, as proposed in [5], would enable to rewrite them as a shared code.

The longest clone (1647 token, 627 lines) was found between src/com/sun/java/swing/plaf/windows/WindowsFileChooserUI.java and src/javax/swing/plaf/metal/MetalFileChooserUI.java (marked *F* in **Figure 2**). Each of the two classes WindowFileChooserUI and MetalFileChoosrUI has nine internal classes, one constructor, and 45 methods, all of which, except three of the methods, are clones.

Conclusions

In this paper, we have presented a clone detecting technique with transformation rules and a token-based comparison, as well as important optimization techniques to improve performance and efficiency. We have also proposed metrics to select interesting clones. They have been applied to several industrial-size software systems in the case studies. Our current clone detection tool does not accept source files written in two or more programming languages. However, today some software systems are implemented in multi-languages (e.g., C and C++, Java and HTML, etc). We are trying to extend the tool to accept source programs written in several programming languages at the same time.

References

[1] B.S. Baker, "A Program for Identifying Duplicated Code", *Proc. Computing Science and Statistics: 24th Symposium on the Interface*, **24**, pp. 49-57 Mar. 1992.

[2] M. Balazinska, E. Merlo, M. Dagenais, B. Lag_e, and K.A. Kontogiannis, "Measuring Clone Based Reengineering Opportunities", *Proc. 6th IEEE Int'l Symposium on Software Metrics (METRICS '99)*, pp. 292-303, Boca Raton, Florida, Nov. 1999.

[3] M. Balazinska, E. Merlo, M. Dagenais, B. Lagüe, and K.A. Kontogiannis, "Partial Redesign of Java Software Systems Based on Clone Analysis", *Proc. 6th IEEE Working Conf., on Reverse Eng. (WCRE '99)*, pp. 326-336, Atlanta, Georgia, Oct. 1999.

[4] I.D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier, "Clone Detection Using Abstract Syntax Trees", *Proc. IEEE Int'l Conf. on Software Maintenance (ICSM) '98*, pp. 368-377, Bethesda, Maryland, Nov. 1998.

[5] G. Bracha, M. Odersky, D. Stoutamire, and P. Wadler. "GJ Specification". http://cm.bell-labs.com/cm/cs/who/wadler/pizza/gj/

[6] D. Gusfield, *Algorithms on Strings, Trees, and Sequences*, pp. 89-180. Cambridge University Press 1997.

[7] B. Laguë, E.M. Merlo, J. Mayrand, and J. Hudepohl. "Assessing the Benefits of Incorporating Function Clone Detection in a Development Process", *Proc. IEEE Int'l Conf. on Software Maintenance (ICSM)'97*, pp. 314-321, Bari, Italy. Oct. 1997.

[8] The source for Java Technology. http://java.sun.com/