SimCad: An Extensible and Faster Clone Detection Tool for Large Scale Software Systems

Md. Sharif Uddin Chanchal K. Roy Kevin A. Schneider
Deptartment of Computer Science, University of Saskatchewan, Saskatoon, Canada
{s.uddin, chanchal.roy, kevin.schneider}@usask.ca

***Abstract*—Code cloning is an inevitable phenomenon in evolution of software systems. To reduce the harmful effects of clones in software evolution, they need to be identified correctly as well in a time efficient way. There might be various types of clones in a software system. Earlier research shows detection of near-miss clones in large datasets appears to be costly in terms of time and memory. Among the clone detection tools available in practice, not very many of them are found effective in that regard. In this paper we present a standalone clone detection tool SimCad. It is based on a highly scalable and faster clone detection algorithm designed to detect both exact and near-miss clones in large-scale software systems. One of the potential aspects of SimCad is that its clone detection function is made more portable by packaging it into a library called SimLib. Thus, SimLib now can be used as an off-the-shelf clone detection library that can be easily integrated into other applications that are designed to work based on detected clones. For example, a standalone tool or an Integrated Development Environment (IDE) plugin can use SimLib for realtime clone detection while providing its own services like clone visualization and/or clone management functionalities. We hope that both researchers and developers would enjoy and utilize the benefit of using these tools in different aspects of detection and management of clones in software.**

I. Introduction

Are clones harmful in software development and evolution or they are not? Despite a decade of active research in software clone, arguably there is no such conclusive finding that significantly favors one argument over the other. However, researchers commonly agreed that in order to make best use of the good sides of code cloning and to reduce the possible harmful effects, we need an efficient and cost-effective way to mange clones. Although, a number of clone detection tools have been proposed in past studies [6], only a few of them can perform well against current diverse requirements such as fast detection of near-miss clones and adaptation/integration of a third party clone detection tool to a clone management system. Thus, the need for designing better clone detection techniques is still considered an important problem. Considering that as a motivation, we have designed SimCad as a fast and scalable clone detection tool that works well in detecting near-miss clones even for large-scale dataset, and SimLib to make the clone detection service more accessible to the applications designed for managing clones. The following two sections will cover both of the tools in short.

II. The SimCad Clone Detector

SimCad features a standalone clone detection tool evolved from our earlier research [8] where effectiveness of *simhash* (a similarity preserving data hashing technique) [1] has been studied for fast detection of clones in source code. The prototype implementation was found effective especially for quick detection of near-miss clones in large-scale software systems. Based on that prototype, a full-fledged clone detection tool has been engineered that we present here as SimCad. It employs a data clustering algorithm with a multi-level index based searching that enables fast detection of clones. Fig. 1 shows an overview of end-to-end clone detection process in simCad. The whole process includes three phases named as: Pre-processing, Detection and Output generation. The pre-processing phase again consists of the following four sub-process: Extraction, Normalization, SimHash generation and Indexing; once done altogether, the result (index) can be reused as many times as needed to perform clone detection operation. Technical details of all the steps can be found in an earlier study [8]. A variant of the mutation/injection framework [5] for evaluating clone detection tools was used to evaluate the correctness of simCad's detection result. Besides a comparative performance analysis of SimCad's clone detection algorithm with another state of the art tool, NiCad [2] was conducted.

SimCad is a structured clone detection tool. That is, it detects clones as code fragments (e.g., function or code block), the boundary of which are predefined during the source code pre-processing step. The tool provides both command-line and graphical user interface for its user. Detection outcome can be varied by providing input to a number of parameters exposed through the user interface. There is also an external configuration file named *simcad.cfg.xml* that provides more configurable parameters with their default values. User can modify these parameter values in order to fine tune the overall clone detection process. The clone detection result is exported as an XML file to a location specified by the user, otherwise to a system defined location relative to the given source location.

Listing 1. Command-line user intefacs of SimCad

```
Usage: simcad2 [-version] [-help] [-gx] [-v] [-i
     item_to_search] -s source_path -l language
    [-g granularity] [-t clone_type] [-c
    clone_grouping]  [-x source_transform]
    [-o output_path]

Usage: simcad2xml [-version] [-help] [-gx] [-v]
    [-n] [-i item_to_search] -s o_source_xml
    [-x t_source_xml] [-t clone_type] [-c
        clone_grouping] [-o output_path]
```

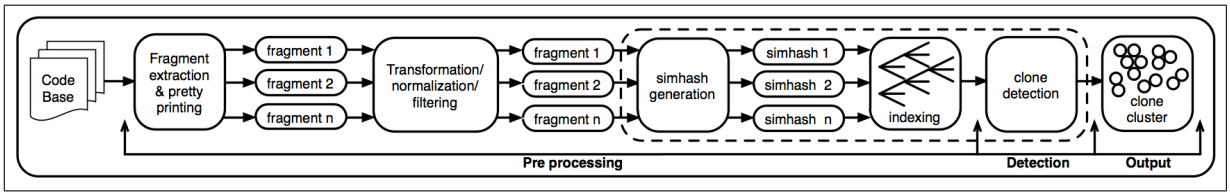

Fig. 1. Clone detection process in *SimCad* (taken from [8])

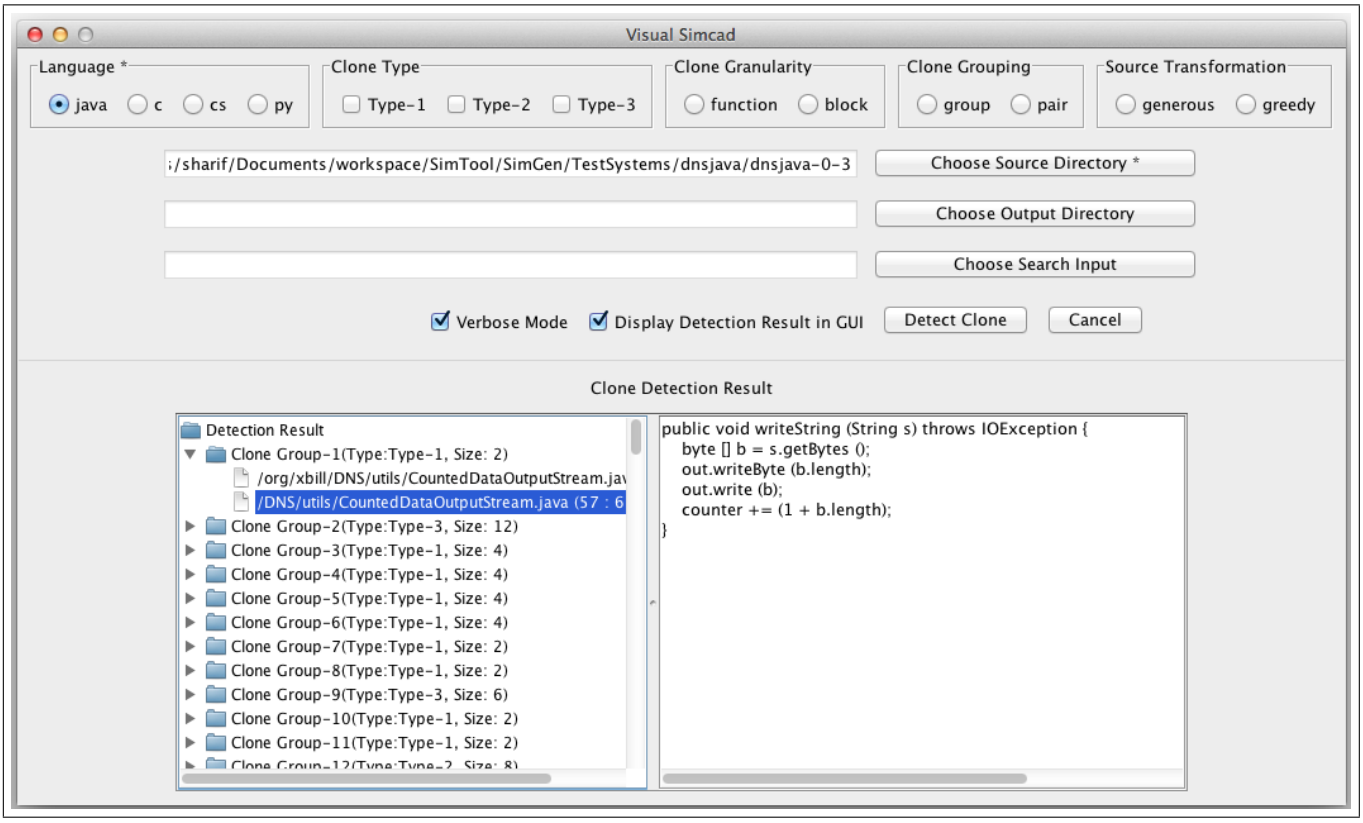

Fig. 2. *SimCad* graphical user interface

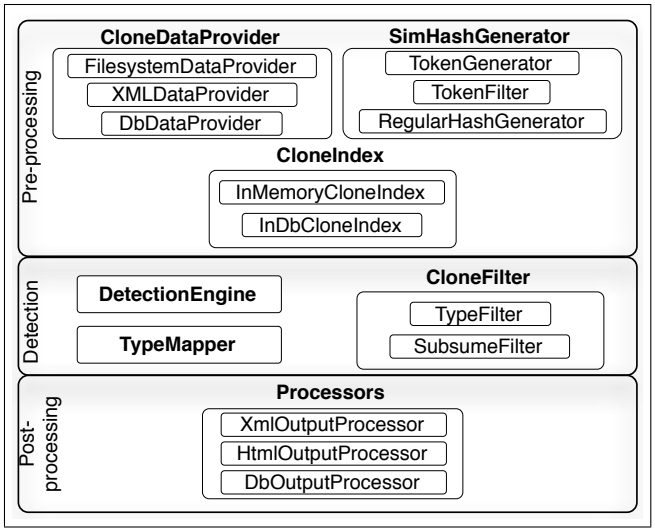

Fig. 3. SimLib architecture

A. Command-line User Interface (CUI) of SimCad

SimCad provides the following two commands for detecting clones in a target system: *simcad2* and *simcad2xml* (Listing 1). The command *simcad2* takes root folder of the subject system and source code language name of that system as two mandatory inputs. The language parameter is required by SimCad to use appropriate TXL [3] scripts for extraction and normalization of code fragments from original source during pre-processing step. Currently it supports four popular programming languages namely: C, Java, Python and CSharp. An example of detecting clones using the command *simcad2* is shown below for a project named 'dnsjava':

```
$ ./simcad2 -s /TestSystems/dnsjava -l java
```

Command *simcad2* has a special optional parameter *item_to_search*. This parameter takes a file or a folder that contains source code to be searched in a target project. That is, the parameter points to some source code as search candidate(s) and execution of the command goes for detecting codes similar to the search candidates in the target project.

This opens up a number of interesting possibilities in code and clone search. For example, user might want to see if some arbitrary code (or anything similar) exists in a target project. In such case, user needs to provide the arbitrary code location as input to the parameter *item_to_search* (i.e., the search candidates outside the target project location) and then execute the detection command after setting the target project. Thus, SimCad in such case essentially works as a source code search engine. An example command is given below.

```
$ ./simcad2 -i /Documents/Snippet.java -s
    /TestSystems/dnsjava -l java
```

Another use case scenario for this command would be *localized clone search*. That is, detection is performed into a specific region of a codebase to see if that region contains any clone code with respect to the whole project. Using this command, user can get the service by proving the subset codebase location as an input to the parameter *item_to_search* and choosing the whole project as the target project as follows:

```
$ ./simcad2 -i /TestSystems/dnsjava/org/xbill/
    DNS/utils -s /TestSystems/dnsjava -l java
```

In the same way, this command could also be used to detect inter-project clones in two difference projects. In case the user ignores the parameter, detection will go for searching all possible clones inside the target project.

The next command *simcad2xml* takes source input through the only required parameter *o_source_xml* as an XML file of a pre-defined format [7]. Optionally, user can provide a similar XML file (through parameter *t_source_xml*) that contains normalized/transformed version of the original source data if available, which would yield better detection result for near-miss clones. This interface opens a great opportunity for SimCad to perform clone detection on any kind of textual data since unlike the previous command it is language independent.

B. Graphical User Interface (GUI) for SimCad

SimCad also provides GUI as a convenient way of using the tool for clone detection. Fig. 2 shows the GUI corresponds to the command *simcad2* that can be made available using the following command:

```
$ ./simcad2 -gx
```

User can provide the other arguments here in the command line as well, but those will be propagated to the GUI as appropriate. Additionally, the GUI provides a simple clone explorer and clone code viewer (bottom part in Fig. 2) that user can choose to display clones (optionally) while performing a clone detection on a subject system.

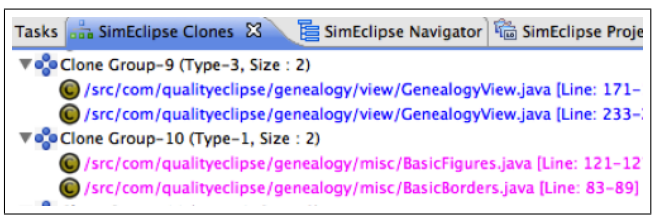

Fig. 4. *SimEclipse Clones View* for display clone detection result

III. SimLib: The Library for Clone Detection

SimLib is developed as a portable clone detection library based on SimCad to make it (SimCad) more usable in practical context. It provides a broader configuration and adaptation facility for identifying clones in data with diverse characteristics and providing off-the-shelf clone detection functionality for a host application. The modular architecture makes SimLib a highly configurable and extensible API that can be tailored with minimal effort to build a fully customized clone detection tool for target data, or can be integrated to IDE or third party source code analysis systems. The architecture contains three layers as shown in Fig. 3: Pre-processing, Detection and Post-processing. Each layer contains a number of configurable components where users can choose different options from within the library or can create customized components and link them with the core system using SimLib's external configuration file to address different detection requirements for the target data.

IV. Summary and Conclusion

Both SimCad and SimLib are potential tools to be used in research and industry. SimCad provides an easy to use command line interface as well as a convenient graphical user interface (GUI) where user can explore the detected clones in a subject system. Using it, clones can be searched locally or in the whole project. Besides, it can work as a code search engine as well. SimCad has been used in a recent study [4] where detection of clones in Java bytecode was experimented. It is also being used in our research lab for several other ongoing studies. On the other hand, SimLib makes the clone detection service portable to other applications that need one. It can be configured for detecting clones both in source and non-source code based documents, where most of the existing tools might not be a good fit. SimLib has been successfully integrated as an Eclipse plugin called SimEclipse that we are currently developing for managing clones in IDE (Fig. 4). We believe both these tools would add a great value not only to the software clone research community but also to the industry. Both the tools are available for download [7] including textual instruction and demonstration for setup and use.

References

[1] M. S. Charikar, "Similarity Estimation Techniques from Rounding Algorithms", Proc. STOC, 2002, pp. 380-388.

[2] J.R. Cordy and C.K. Roy, "The NiCad Clone Detector", Proc. ICPC, 2011, pp. 219-220.

[3] J. R. Cordy, "The TXL Source Transformation Language," Science of Computer Programming, 2006, 61(3), pp. 190-210.

[4] I. Keivanloo, C. K. Roy, and J. Rilling, "Java Bytecode Clone Detection via Relaxation on Code Fingerprint and Semantic Web Reasoning", Proc. IWSC, 2012, pp. 36-42.

[5] C. K. Roy and J. R. Cordy, "A Mutation / Injection-based Automatic Framework for Evaluating Code Clone Detection Tools", Proc. ICST Mutation Workshop, 2009, pp. 157-166.

[6] C. K. Roy, J. R. Cordy and R. Koschke. "Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach", Science of Computer Programming, 2009, 74(7), pp. 470-495.

[7] SimCad Clone Detector, URL: http://homepage.usask.ca/~mdu535/tools.html (Last accessed April 2013)

[8] S. Uddin, C. K. Roy, K. A. Schneider and A. Hindle, "On the Effectiveness of Simhash for Detecting Near-Miss Clones in Large Scale Software Systems", Proc. WCRE, 2011, pp. 13-22.