SourcererCC and SourcererCC-I: Tools to Detect Clones in Batch mode and During Software Development

Vaibhav Saini, Hitesh Sajjani, Jaewoo Kim, and Cristina Lopes

Dept. of Information and Computer Science, University of California
Irvine, CA, USA

vpsaini@uci.edu, hsajjani@uci.edu, jaewoo@uci.edu, lopes@uci.edu

ABSTRACT

Given the availability of large source-code repositories, there has been a large number of applications for large-scale clone detection. Unfortunately, despite a decade of active research, there is a marked lack in clone detectors that scale to big software systems or large repositories, specifically for detecting near-miss (Type 3) clones where significant editing activities may take place in the cloned code.

This paper demonstrates: (i) SourcererCC, a token-based clone detector that targets the first three clone types, and exploits an index to achieve scalability to large inter-project repositories using a standard workstation. It uses an optimized inverted-index to quickly query the potential clones of a given code block. Filtering heuristics based on token ordering are used to significantly reduce the size of the index, the number of code-block comparisons needed to detect the clones, as well as the number of required token-comparisons needed to judge a potential clone; and (ii) SourcererCC-I, an Eclipse plug-in, that uses SourcererCC's core engine to identify and navigate clones (both inter and intra project) in real-time during software development.

In our experiments, comparing SourcererCC with the state-of-the-art tools¹, we found that it is the only clone detection tool to successfully scale to 250 MLOC on a standard workstation with 12 GB RAM and efficiently detect the first three types of clones (precision 86% and recall 86-100%). Link to the demo: https://youtu.be/17F_9Qp-ks4

1. INTRODUCTION

Clone detection locates exact or similar pieces of code, known as clones, within or between software systems. Clones are created when developers reuse code by copy, paste and modify, although clones may be created by a number of other means [1]. Developers need to detect and manage their clones in order to maintain software quality, detect and prevent new bugs, reduce development risks, and costs [1]. Clone management and clone research studies depend on quality tools.

With the amount of source code increasing steadily, large-scale clone detection has become a necessity. Large-scale clone detection can be used for mining library candidates [2],

detecting similar mobile applications [3], license violation detection [4], reverse engineering product lines [5], finding the provenance of a component [6], and code search [7, 8]. Large-scale clone detection allows researchers to study cloning in large software ecosystems (e.g., Debian), or study cloning in open-source development communities like GitHub. Developers often clone modules or fork projects to meet the needs of different clients, and need the help of large-scale clone detectors to merge these cloned systems towards a product-line style of development. These applications require tools that scale to hundreds of millions of lines of code. However, very few tools/techniques can scale to the demands of clone detection in very large code bases [9]. They achieve scalability by using deterministic [10] or non-deterministic [9] input partitioning and distributed execution of an existing non-scalable detector, using large distributed code indexes [11], or by comparing hashes after Type-1/2 normalization [2]. These existing techniques have a number of limitations. The novel scalable algorithms [11, 2] do not support Type-3 near-miss clones, where minor to significant editing activities might have taken place in the copy/pasted fragments, and therefore miss a large portion of the clones, since there are more Type-3 in the repositories than other types [12]. Type-3 clones can be the most needed in large-scale clone detection applications [7, 3]. While input partitioning can scale existing non-scalable Type-3 detectors, this significantly increases the cumulative runtime, and requires distribution over a large cluster of machines to achieve scalability in absolute runtime [9, 10]. Distributable tools [10] are costly and difficult to setup and maintain.

To address this state of affairs, we developed a clone detection technique [13, 14], which satisfies the following requirements: (1) accurate detection of near-miss clones, where minor to significant editing changes occur in the copy/pasted fragments; (2) programming language agnostic; (3) simple, non-distributed operation; and (4) scalability to hundreds of millions of lines of code using a standard workstation. This paper demonstrates the accompanying tool-suite (SourcererCC and SourcererCC-I) and several notable features that rely on the introduced approach in our earlier work.

2. SOURCERERCC'S OVERVIEW

SourcererCC's general procedure is summarized in Figure 1. It operates in two primary stages: (i) partial index creation; and (ii) clone detection.

In the index creation phase, it parses the code blocks from the source files, and tokenizes them with a simple scanner that is aware of token and block semantics of a given language². From the code blocks it builds an inverted index mapping tokens to the blocks that contain them. Unlike previous approaches, it does not create an index of all tokens in the code blocks, instead it uses a filtering heuristic, named

¹Deckard, CCFinder, NiCad, and iClones

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ICSE'16 Companion, May 14-22, 2016, Austin, TX, USA

© 2016 ACM. ISBN 978-1-4503-4205-6/16/05...\$15.00

DOI: <http://dx.doi.org/10.1145/2889160.2889165>

²Currently, we have implemented this for Java, C and C#, but can be easily extended to other languages

Figure 1: SourcererCC's clone detection process

Sub-block Overlap Filtering to construct a partial index of only a subset of the tokens in each block [14].

In the detection phase, SourcererCC iterates through all of the code blocks, retrieves their candidate clone blocks from the index. As per the filtering heuristic, only the tokens within the sub-block are used to query the index, which reduces the number of candidate blocks. After candidates are retrieved, SourcererCC uses another filtering heuristic named *Token Position Filtering*, which exploits ordering of the tokens in a code block to measure a live upper-bound and lower-bound of similarity scores between the query and candidate blocks [14]. Candidates whose upper-bound falls below the similarity threshold are eliminated immediately without further processing. Similarly, candidates are accepted as soon as their lower-bound exceeds the similarity threshold. This is repeated until the clones of every code block are located.

3. FEATURES

3.1 Scalable

In our experiments, comparing SourcererCC with the state-of-the-art tools (Deckard, CCFinder, NiCad, and iClones), we found that it is the only clone detection tool to successfully scale to 250 MLOC on a standard workstation with 12 GB RAM³ and detect first three types of clones [14].

3.2 Accurate

In our experiments, SourcererCC gave 100% recall for Type 1 clones, 97-100% recall for Type 2 clones and 86 - 99% recall for Type 3 clones [14]. It gave 86% precision, a strong precision as per the literature [1, 15].

3.3 Language Independent

SourcererCC decouples the parsing logic from the indexing and clone detection logic, making it easier to support any number of languages as long as there is a parser for it. Currently, we have parsers to support Java, C, C++, C#, and python.

3.4 Fast Index Creation and Clone Detection

SourcererCC uses filtering heuristics to index a small subset (for 70% threshold, only 30% of the block gets indexed) of a code block, making the size of the index small. Thus, indexing large software repositories takes considerably less time. Moreover, searching on smaller indexes is faster. For example, it took around 1.5 days to report clones on 100 MLOC, 3 times faster than CCFinderX, the second fastest tool [14].

³SourcererCC successfully executed even on a machine with 8GB RAM

3.5 Configurable

The threshold is configurable and users can increase or decrease it to find more stricter or liberal clones, respectively. Moreover, it is possible to detect clones at different granularities such as block level (statements between {}), method level, and file level. We note that we found the precision and recall to be optimum at 70% threshold.

4. SOURCERERCC-I: ECLIPSE PLUG-IN

Many batch processing tools are designed for detecting clones in large software repositories. These tools, however, do not integrate well with the development process. To this end, Lague et al. conducted a case study to assess the impact of integrating clone detection with the development process as a preventive control for maintenance issues [16]. They analyzed 89 million lines of code and found several opportunities where the integration of automated clone detection with the development process could have helped.

Although many tools have been developed to automatically detect clones, adoption of these tools during software development still faces three main challenges: (i) Difficult to setup and configure; (ii) Not designed to seamlessly integrate with the development process, as these are mostly stand alone software, designed for batch processing; and (iii) The output files of almost all clone detection tools is impossible to read by a human at any large-scale.

4.1 SourcererCC-I Overview

SourcererCC-I addresses the above limitations. It is an Eclipse plug-in that instantaneously reports intra & inter project method level clones in Java software. SourcererCC-I is built on top of SourcererCC, thus exploiting its powerful indexing and searching capabilities. SourcererCC-I detects the method a developer is currently working on, then proactively reports the clones of the method in a non-obtrusive manner. SourcererCC-I reports the clones inside the IDE, rather than just creating an output file that is a report. This bypasses the need to look at the text of a clone detection report file offering a great advantage to the user.

4.2 SourcererCC-I's Architecture

SourcererCC-I has five modules as shown in Figure 2.

Figure 2: SourcererCC-I Architecture

4.2.1 Manager: controls interaction between different modules

The *Manager* module acts as a controller, delegating jobs like *create indexes*, *update indexes*, *search clones*, and *report clones* to other modules. It mediates the data flow among them and listens to the *change* and *selection* events gener-

ated by the editor. On detection of such events, it performs action such as *update indexes* or *search clones*.

4.2.2 Parser: parses projects and creates input for the indexer

The *Parser* generates input files for the *Indexer* Module. On activation of the plug-in, the *Manager* delegates the job of parsing the source files to the *Parser* Module, which creates parsed files using Eclipse's JDT (Java Development Toolkit).

4.2.3 Indexer: creates inverted and forward indexes of code blocks

The *Indexer* uses the parsed files to create a partial inverted index and a forward index for each of the open projects in the Eclipse workspace. The partial inverted index is used to search the candidate clones whereas the forward index is used to verify if the candidates are clones or not. The more details on the creation and working of these indexes can be found in our other study [14].

4.2.4 Searcher: searches clones in the indexes

The *Manager* detects the current method and then creates its query block. The *Searcher* uses this query block as an input and detect its clones using the indexes. It produces a clone output file containing meta information of clones.

4.2.5 Reporter: reports detected clones

After the *Searcher* detects the clones of the current method, the *Manager* asks the *Reporter* module to perform two tasks i) create a marker on the editor (where the line numbers are written). This marker signifies the presence of clones of the current method in the project; and ii) display the list of clone methods using a view part in Eclipse. A user can quickly navigate to the clone methods by clicking any item in this list of clone methods.

4.3 SourcererCC-I's Features

1. Non-obtrusive user interface

SourcererCC-I has a minimalistic design in order to minimize the cognitive burden on the user.

displaying clones. The tool uses colored markers to notify the clones detected. A marker is either *red*, *blue*, or *green* in color where *red* signifies there are more than 10 clones of the current method in the project; *yellow* signifies there are five to 10 clones; and *green* signifies the existence of less than five clones

Figure 3 shows a red colored marker (annotated inside the top blue rectangular region in the eclipse's editor) signifying that SourcererCC-I has found more than 10 clones of this method.

navigating clones. The output of most of the detection tools is impossible to read by a human at any large-scale. This is because of an extra step to link the output produce by the tools to the actual source of code fragments. To overcome this limitation, SourcererCC-I displays all the clones found for a method using their fully qualified name in the result view pane of Eclipse's console. This not only facilitates easy navigation to cloned methods (in the editor) using a mouse click but also displays the clones in an hierarchical fashion using a tree structure. The detected clones are grouped by projects and also by files, making it easier for a developer to navigate.

2. Instant clone detection

SourcererCC's optimized index and filtering heuristics enable SourcererCC-I to detect clones in real time.

3. Incremental and fast index creation

SourcererCC-I index creation step is very fast i.e., few milliseconds for even project of 100,000 LOC. Thus, even for a

project consisting of several thousands of lines of code, clone detection process can begin as soon as the project is loaded in the workspace. Moreover, the index creation is incremental, meaning, the index is not re-created for the whole project whenever a file or a method is changed. This is a very important feature for clone detection tools to be useful during ongoing development activities.

4. Inter-project and intra-project clone detection

SourcererCC-I has not only the ability to detect clones in a given project, but it can also easily detect clones across projects or repositories.

5. Quick and easy set-up

SourcererCC-I can be installed easily by following these 3 steps in the given order: (i) Open Eclipse and click *Install New Software* in *Help* menu; (ii) Click *Add*. Then, in the *Add Repository* window, enter the name SourcererCC-I, and in the location field enter the url <http://tinyurl.com/pm3xl2g>; and finally, (iii) follow the steps given in eclipse wizard to install the plug-in.

Figure 3: Screenshot of Eclipse, displaying that the plug-in has detected clones.

5. RELATED TOOLS

We have extensively covered the work related to SourcererCC in our other paper [14]. In this section we focus on the tools which are related to SourcererCC-I.

Tools integrated with Development Environment.

Patricia Jablonski's proposed CnP, a tool to detect copy-and-paste clones in the IDE [17]. CnP establishes links between the original and the pasted clones and uses their content information later on for error detection and other purposes. Unlike our tool, CnP doesn't use a clone detector to detect clones and hence is not capable of finding legacy clones. Moreover, CnP cannot find accidental clones.

CP-Miner detects copy-paste errors in the context of traditional clone detection [18]. It uses a token-based approach with data mining and a gap constraint. We are using an index-based token comparison technique, which falls under the information retrieval approach. Also, unlike our tool, CP-Miner is not available freely and one needs to buy a license to use it.

CloneTracker, an Eclipse plug-in is a tool that keeps track of the evolution of clones [19]. For its input, it relies on the output of a clone detector tool, that needs to be run before

hand to generate all the clones in a system. CloneTracker, unlike SourcererCC-I, does not detect the clones in real time.

Stand Alone Tools. There are some stand alone tools, namely Dup [20], iClones [21], CCFinder [22], and Nicad [23] that do batch processing to detect clones, but these tools are not designed to be integrated with the development process. Nicad, however, can be integrated to an IDE but being a batch processing tool it lacks the capability to detect clones on the fly. CloneDR, a commercial AST based clone detection tool [24] detects clones by finding identical subtrees. CloneDR claims to have Eclipse integration for IBM Enterprise languages.

6. CONCLUSION AND FUTURE WORK

In this paper we presented: (i) SourcererCC, a token-based clone detector that targets the first three clone types, and exploits an index to achieve scalability to large inter-project repositories using a standard workstation. SourcererCC uses an optimized inverted-index to quickly query the potential clones of a given code block. Filtering heuristics based on token ordering are used to significantly reduce the size of the index, the number of code-block comparisons needed to detect the clones, as well as the number of required token-comparisons needed to judge a potential clone. (ii) SourcererCC-I, an easy-to-use, free, and well packaged clone detection plug-in for the Eclipse IDE that can detect clones on the fly. The tool pro-actively finds method level clones (inter and intra-project) and reports them in a non-obtrusive manner. In future we plan to exploit the architecture of both Sourcerer-CC and SourcererCC-I, which decouples the parsing logic from the clone detection logic, to add support for more languages and different granularities of code blocks.

Tool Artifacts. Link to install the Eclipse plug-in: <http://mondego.ics.uci.edu/projects/clonedetection/tool/latest> & link to the source of SourcererCC: <http://mondego.ics.uci.edu/projects/clonedetection/tool/source>

7. REFERENCES

- [1] C. K. Roy and J. R. Cordy, "A survey on software clone detection research," no. TR 2007-541, 2007. 115 pp.
- [2] T. Ishihara, K. Hotta, Y. Higo, H. Igaki, and S. Kusumoto, "Inter-project functional clone detection toward building libraries - an empirical study on 13,000 projects," in *Reverse Engineering (WCRE), 2012 19th Working Conference on*, pp. 387–391, Oct 2012.
- [3] K. Chen, P. Liu, and Y. Zhang, "Achieving accuracy and scalability simultaneously in detecting application clones on android markets," in *Proceedings of the 36th International Conference on Software Engineering, ICSE 2014, (New York, NY, USA)*, pp. 175–186, ACM, 2014.
- [4] R. Koschke, "Large-scale inter-system clone detection using suffix trees," in *Proceedings of CSMR*, pp. 309–318, 2012.
- [5] A. Hemel and R. Koschke, "Reverse engineering variability in source code using clone detection: A case study for linux variants of consumer electronic devices," in *Proceedings of Working Conference on Reverse Engineering*, pp. 357–366, 2012.
- [6] J. Davies, D. German, M. Godfrey, and A. Hindle, "Software Bertillonage: finding the provenance of an entity," in *Proceedings of MSR*, 2011.
- [7] I. Keivanloo, J. Rilling, and P. Charland, "Internet-scale real-time code clone search via multi-level indexing," in *Proceedings of WCRE*, 2011.
- [8] S. Kawaguchi, T. Yamashina, H. Uwano, K. Fushida, Y. Kamei, M. Nagura, and H. Iida, "Shinobi: A tool for automatic code clone detection in the ide," vol. 0, (Los Alamitos, CA, USA), pp. 313–314, IEEE Computer Society, 2009.
- [9] J. Svajlenko, I. Keivanloo, and C. Roy, "Scaling classical clone detection tools for ultra-large datasets: An exploratory study," in *Software Clones (IWSC), 2013 7th International Workshop on*, pp. 16–22, May 2013.
- [10] S. Livieri, Y. Higo, M. Matsushita, and K. Inoue, "Very-large scale code clone analysis and visualization of open source programs using distributed ccfinder: D-ccfinder," in *Proceedings of ICSE*, 2007.
- [11] B. Hummel, E. Juergens, L. Heinemann, and M. Conradt, "Index-based code clone detection: incremental, distributed, scalable," in *Proceedings of ICSM*, 2010.
- [12] C. K. Roy and J. R. Cordy, "Near-miss function clones in open source software: An empirical study," *J. Softw. Maint. Evol.*, vol. 22, pp. 165–189, Apr. 2010.
- [13] H. Sajjani, V. Saini, and C. Lopes, "A parallel and efficient approach to large scale clone detection," *Journal of Software: Evolution and Process*, vol. 27, no. 6, pp. 402–429, 2015.
- [14] H. Sajjani, V. Saini, J. Svajlenko, C. K. Roy, and C. V. Lopes, "Sourcerercc: scaling code clone detection to big-code," in *Proceedings of the 38th International Conference on Software Engineering*, 2016.
- [15] C. K. Roy, J. R. Cordy, and R. Koschke, "Comparison and evaluation of code clone detection techniques and tools: A qualitative approach," *Sci. of Comput. Program.*, pp. 577–591, 2009.
- [16] B. Lague, D. Proulx, J. Mayrand, E. M. Merlo, and J. Hudspohl, "Assessing the benefits of incorporating function clone detection in a development process," in *Software Maintenance, 1997. Proceedings., International Conference on*, pp. 314–321, IEEE, 1997.
- [17] D. Hou, P. Jablonski, and F. Jacob, "Cnp: Towards an environment for the proactive management of copy-and-paste programming," in *Program Comprehension, 2009. ICPC'09. IEEE 17th International Conference on*, pp. 238–242, IEEE, 2009.
- [18] Z. Li, S. Lu, S. Myagmar, and Y. Zhou, "Cp-miner: A tool for finding copy-paste and related bugs in operating system code," in *OSDI*, vol. 4, pp. 289–302, 2004.
- [19] E. Duala-Ekoko and M. P. Robillard, "Clonetraacker: tool support for code clone management," in *Proceedings of the 30th international conference on Software engineering*, pp. 843–846, ACM, 2008.
- [20] B. S. Baker, "On finding duplication and near-duplication in large software systems," in *Reverse Engineering, 1995., Proceedings of 2nd Working Conference on*, pp. 86–95, IEEE, 1995.
- [21] N. Gode and R. Koschke, "Incremental clone detection," in *Software Maintenance and Reengineering, 2009. CSMR'09. 13th European Conference on*, pp. 219–228, IEEE, 2009.
- [22] T. Kamiya, S. Kusumoto, and K. Inoue, "CCFinder: a multilingualistic token-based code clone detection system for large scale source code," *IEEE Trans. Softw. Eng.*, vol. 28, no. 7, pp. 654–670, 2002.
- [23] C. K. Roy and J. R. Cordy, "Nicad: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization," in *Program Comprehension, 2008. ICPC 2008. The 16th IEEE International Conference on*, pp. 172–181, IEEE, 2008.
- [24] I. D. Baxter, A. Yahin, L. Moura, M. Sant'Anna, and L. Bier, "Clone detection using abstract syntax trees," in *Software Maintenance, 1998. Proceedings., International Conference on*, pp. 368–377, IEEE, 1998.