Instant Clone Finder: Detecting Clones During Software Development

Vaibhav Saini, Hitesh Sajnani, Jaewoo Kim, Aftab Hussain and Cristina Lopes
Dept. of Information and Computer Science, University of California
Irvine, CA, USA
vpsaini@uci.edu, hsajnani@uci.edu, jaewoo@uci.edu, aftabh@uci.edu, lopes@uci.edu

ABSTRACT

Code clone detection has been an active area of research over the past few years. Due to various maintainance implications, it is perceived to be equally important in practice. Although many advanced techniques are proposed to detect clones, there are very few tools that are readily available for use. Moreover, most of the tools function in a batch mode, and are not integrated with the development environment, making them less suitable for the use of developers.

In this paper, we describe InstaCF, an index-based code clone detection tool that detects inter and intra project clones in real time during the software development process. The tool is also packaged as an Eclipse plugin that integrates seamlessly with it. The tool is token based and has the capability to detect Type 1, Type 2 and Type 3 clones.

We posit that the tool has a potential to increase developers' awareness about the presence of clones in their projects, enabling them to take appropriate clone management actions.

1. INTRODUCTION

Software developers create duplicate code, often referred to as code clones, both deliberately (intentional clones) and accidentally (unintentional clones) [3, 17, 18, 19, 5]. Intentional cloning is done for reasons like quick prototyping, performance gain, and separation of concerns [18] . Unintentional clones, however, get created by accidents when developers are unaware of a similar pieces of code in their software [1].

It has been noted in both research and practice that clones lead to maintenance issues. For example, modifications made in one piece of code may need to be propogated to its clone siblings, adding to the maintenance cost [10]. The cost may further increase if developers are not aware of the existence of the clone siblings, leading to inconsistent changes [4, 13, 14, 15, 29, 22, 27]. Moreover, studies have found that cloning may lead to the replication of bug patters in the system [32, 27, 7].

Despite the negative aspects being well recognized, cloning seems to be a prevalent practice in software engineering. In general a large software consists of 10-15 % of cloned code [3]. As such, researchers seem to have acknowledged the fact that clones in software are unavoidable and should be seen as a part of the development activity [20, 23, 27].

More recently, we conducted a preliminary survey to assess developers' practices of code cloning. We gathered responses from 72 developers, most of whom had several years of industrial experience (see Figure 1. In summary, we found 54/72 developers actively search for code before any coding task, thus increasing the likelihood of creating a clone (see Figure 2). Moreover, 69/72 developers copy the code found. We also asked if they wanted a feature in the IDE to highlight similar pieces of code in their codebase? To that, 65/72 responded "Yes". Although, this analysis is not exhaustive, it provides an idea of how persive the practice of code cloning has become in practice. Moreover, recent studies have shown that cloning is not as bad as earlier thought to be [28, 21, 32], generating enough motivation for researchers to look into compensative clone management techniques [34, 12, 6, 33] over clone removal. For each of these clone management techniques to work, discovery of clones (clone detection) is an important first step. Though a lot of work has already been done in this area [2, 9, 16, 24, 25, 17, 27, 30, 12], not much work is done to integrate clone management as a part of development process. To this end, Lague et al. [26] conducted a case study to assess the impact of integrating clone detection with the development process as a preventive control for maintainenance issues. 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.

As a result, although many tools have been developed to automatically detect clones adoption of these tools during software development still faces many challenges:

(i) Most of the clone detection tools in SE research are created as a prototype to demonstrate a novel clone detection technique. Thus leaving little motivation for researchers to package them for distribution. Due to lack of this effort, the overhead incurred in setting up these tools, and configuring the right envirornment turns many developers away.

(ii) It is worth nothing that there are few tools that are well documented and readily available for use [11, 30, 17], however, these tools are not designed to be integrated as a part of the development process. They are mostly stand alone softwares, designed for batch processing of sofwares. For example, a typical workflow for these tools is a developer points them to the code base, run the tool as a separate process, wait for the tool to produce results, and then manually mine the clone to link them back to the code base. While such tools might be useful to perform certain analysis of the code base (e.g., # of clones in the system), this workflow is not suited for detecting clones during development. We posit that the context switch from the regular development task to detect clones might dull developer's interest in using a tool at all.

Our contribution. To address the above limitations, we developed InstaCF, an eclipse plug-in that instantaneously reports intra & inter project method level clones in Java software. InstaCF detects the method the developer is cur-

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. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright 20XX ACM X-XXXXX-XX-X/XX/XX ...$15.00.

rently working on, then pro-actively reports the clones of the method in a non-obtrusive manner. The tool is based on an index based clone detection technique that we proposed earlier [31].

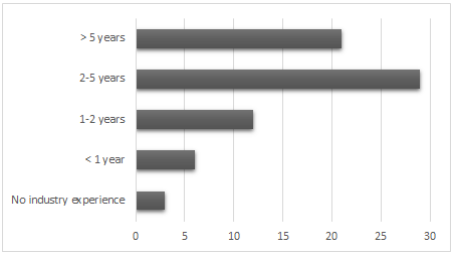
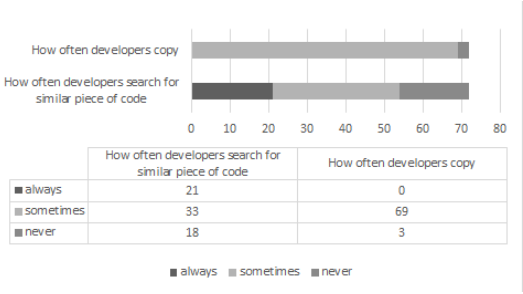


Figure 1: Industrial Experience of Survey Participants



	How often developers search for similar piece of code	How often developers copy
always	21	0
sometimes	33	69
never	18	3

Figure 2: Participants' Responses

2. THE INSTACF TOOL

2.1 Plug-in Overview

The eclipse based plug-in is built on top of the index-based token code clone detection proposed in [31]. It works on the principle of computing similarity between code blocks using a user specified simiarlity function and simiarlity threshold.

InstaCF can be installed easily by following 3 steps in 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 InstaCF, and in the location field enter the url `http://mondego.ics.uci.edu/projects/clonedetection/tool/latest`; and finally, (iii) follow the steps given in eclipse wizard to install the plug-in.

After the plug-in is installed, it gets activated when eclipse launches. The first time, it traverses all open projects in eclipse's workspace and creates indexes. Simultaneously, the plug-in identifies the method or code block, a developer is currently working on the editor. It then computes the similarity score of each code block with the current method using the pre-computed indexes. All the code blocks which bear similarity score greater than the user specified threshold are reported as clones. The entire process is real time with no observable lag even for workspace having more millions LOC. For example, we noticed average (over 500 queries) clone detection time to be 51.20 ms[1] on an index of 36 projects consisting of 1.92 million LOC. Moreover the index creation is reasonably fast (e.g., it took 10.66 minutes for the above code base). and incremental - only updating existing indexes with the addition of new code blocks in the code base.

[1]Time observed on a regular laptop running OSX with Intel i7, 2.3 Ghz processor and 16 GB DDR3 Ram.

2.2 Architecture

InstaCF has five modules as shown in Figure 3. In the following, we discuss each of the modules.

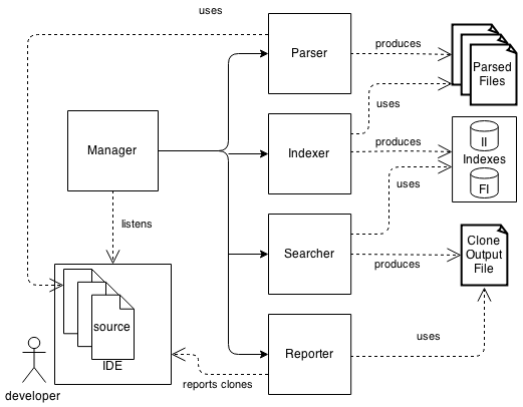


Figure 3: Tool Architecture

2.2.1 *Manager: controls interaction between different modules*

Manager module acts as a controller in our plug-in. Its role is to delegate jobs like *create indexes*, *update indexes*, *search clones*, and *report clones* to other modules, and also to mediate the data flow among them. It also, listens to the *change* and *selection* events generated by the editor. On detection of such events, it performs action such as *update indexes* or *search clones*.

2.2.2 *Parser: parses projects and creates input for the indexer*

Job of the traverser is to generate input files for the Indexer Module. On activation of the plug-in, Manager delegates the job of creating indexes to the Parser Module, which creates parsed files using Eclipse's JDT (Java Development Toolkit).

2.2.3 *Indexer: creates inverted and forward indexes of code blocks*

Indexer uses the parsed files created by the Parser Module is used as an input to the Indexer Module, which then creates a partial inverted index and a forward inverted index for each of the open projects in the eclipse worksapce. 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 previous study [31].

2.2.4 *Searcher: searches clones in the indexes*

The job of the Searcher module is to search clones of a given method. Manager detects the current method on which the developer is working and then creates a query block of the current method. Searcher uses the query block as an input and detect its clones using the indexes. It produces clone output file containing meta information of clones.

2.2.5 *Reporter: reports detected clones*

After the clone detector module detects the clones of the current method, 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) read the clone output file created by Searcher create a view

part to display the list of clone methods in the Eclipse. The user can quickly navigate to the clone methods by clicking any item in this list of clone methods.

2.3 Plug-in User Interface

The Eclipse Java editor acts as the primary user interface for InstaCF. The plug-in is designed to have a minimalistic design in order to minimize the cognitive burden on the user for using and understanding a new utility in the environment. Figure 4 shows a red colored marker (annotated inside the top blue rectangular region in the eclipse's editor) signifying that the InstaCF has found clones of the method, on which the developer is currently working. A marker can have one out of three colors: a) *Red* - signifies there are more than 10 clones of the current method in the project; b) *Yellow* - signifies there are five to 10 clones and c) *Green* - signifies the existence of less than five clones. On hovering the mouse pointer above the marker would show a view-part (annotated inside the bottom blue rectangular region), listing all the clones of the current method. Developers can navigate to the clone methods by simply clicking on the method names in the list.

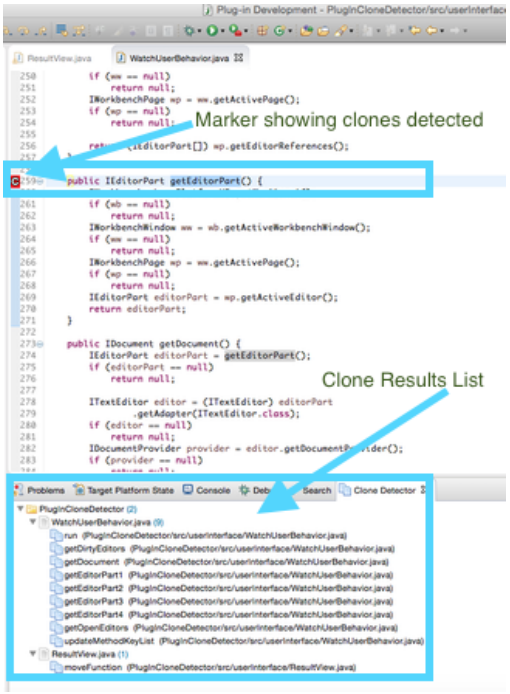


Figure 4: Screenshot of eclipse, displaying that the plug-in has detected clones.

3. RELATED WORK

In this section we present related tools that are available for clone detection. We first compare our tool with the tools which are integrated with the development environment and later we compare our tool with the stand alone tools.

Tools integrated with Development Environment. Patricia Jablonski's proposed CnP, a tool to detect copy-and-paste clones in the IDE [12]. 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 [27]. 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 [8]. For it's 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 InstaCF, does not detects the clones in real time.

Stand Alone Tools. There are some stand alone tools, namely Dup [3], iClones [11], CCFinder [17], and Nicad [30] 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. Baxter et al. created an AST based clone detection tool [5]. It detects clones by finding identical subtrees. They used the tool to demonstrate the use of AST to detect clones and did not release the tool. Moreover, it is a stand alone tool that does batch processing and they did not show if it can be integrated to a development environment.

4. CONCLUSION AND FUTURE WORK

In this paper, we presented InstaCF, 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 and reports them in a non-obtrusive manner. The motivation behind building this tool was the fact that though a lot of research has been done on clone detection and a large number of tools have been released by researchers, not many clone detection tools are integrated with the development process. The existing tools suffer from one or more of the following limitations: (i) they are not packaged well for distribution; (ii) they stand alone software and detect clones in batches; and (iii) they are not available free of cost. A pre-assessment survey confirmed the demand of such a tool among software developers. Currently, our tool can successfully and pro-actively detect method level clones within all the open projects in Eclipse, for code written in Java.

InstaCF is based on an efficient index based approach that could find clones on the fly. The next step would be to improve the tool's index generation phase. Currently the indexes are created sequential; since index generation could take a considerable time particularly for large projects, it is important these executions are implemented in a multi-threaded fashion such that the work flow of the developer is not interrupted while the index generation takes place.

5. TOOL ARTIFACTS

Link to the screencast that shows the tool in action: `http://mondego.ics.uci.edu/projects/clonedetection/tool/demo`

Link to install the Eclipse plug-in: `http://mondego.ics.uci.edu/projects/clonedetection/tool/latest`

Link to the source of underlying code clone detector: `http://mondego.ics.uci.edu/projects/clonedetection/tool/source`

6. REFERENCES

[1] R. Al-Ekram, C. Kapser, R. Holt, and M. Godfrey. Cloning by accident: an empirical study of source code cloning across software systems. In *Empirical Software Engineering, 2005. 2005 International Symposium on*, pages 10–pp. IEEE, 2005.

[2] B. Baker. A program for identifying duplicated code. In *Computing Science and Statistics*, pages 24–49, 1992.

[3] B. S. Baker. On finding duplication and near-duplication in large software systems. In *Reverse Engineering, 1995., Proceedings of 2nd Working Conference on*, pages 86–95. IEEE, 1995.

[4] T. Bakota, R. Ferenc, and T. Gyimothy. Clone smells in software evolution. In *Software Maintenance, 2007. ICSM 2007. IEEE International Conference on*, pages 24–33. IEEE, 2007.

[5] 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*, pages 368–377. IEEE, 1998.

[6] A. Begel and R. DeLine. Codebook: Social networking over code. In *Software Engineering-Companion Volume, 2009. ICSE-Companion 2009. 31st International Conference on*, pages 263–266. IEEE, 2009.

[7] A. Chou, J. Yang, B. Chelf, S. Hallem, and D. Engler. *An empirical study of operating systems errors*, volume 35. ACM, 2001.

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

[9] S. Ducasse, M. Rieger, and S. Demeyer. A language independent approach for detecting duplicated code. In *Proceedings of the IEEE International Conference on Software Maintenance*, pages 109–118. IEEE Computer Society, 1999.

[10] R. Geiger, B. Fluri, H. C. Gall, and M. Pinzger. Relation of code clones and change couplings. In *Fundamental Approaches to Software Engineering*, pages 411–425. Springer, 2006.

[11] N. Gode and R. Koschke. Incremental clone detection. In *Software Maintenance and Reengineering, 2009. CSMR'09. 13th European Conference on*, pages 219–228. IEEE, 2009.

[12] 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*, pages 238–242. IEEE, 2009.

[13] P. Jablonski and D. Hou. Cren: a tool for tracking copy-and-paste code clones and renaming identifiers consistently in the ide. In *Proceedings of the 2007 OOPSLA workshop on eclipse technology eXchange*, pages 16–20. ACM, 2007.

[14] P. Jablonski and D. Hou. Renaming parts of identifiers consistently within code clones. In *Program Comprehension (ICPC), 2010 IEEE 18th International Conference on*, pages 38–39. IEEE, 2010.

[15] L. Jiang, Z. Su, and E. Chiu. Context-based detection of clone-related bugs. In *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 55–64. ACM, 2007.

[16] J. H. Johnson. Identifying redundancy in source code using fingerprints. In *Proceedings of the 1993 conference of the Centre for Advanced Studies on Collaborative research and Software Engineering*, pages 171–183. IBM Press, 1993.

[17] T. Kamiya, S. Kusumoto, and K. Inoue. Ccfinder: a multilinguistic token-based code clone detection system for large scale source code. *Software Engineering, IEEE Transactions on*, 28(7):654–670, 2002.

[18] C. Kapser and M. W. Godfrey. Toward a taxonomy of clones in source code: A case study. In *ELISA workshop*, page 67, 2003.

[19] C. Kapser and M. W. Godfrey. Aiding comprehension of cloning through categorization. In *Software Evolution, 2004. Proceedings. 7th International Workshop on Principles of*, pages 85–94. IEEE, 2004.

[20] C. J. Kapser and M. W. Godfrey. Supporting the analysis of clones in software systems. *Journal of Software Maintenance and Evolution: Research and Practice*, 18(2):61–82, 2006.

[21] C. J. Kapser and M. W. Godfrey. âĂIJcloning considered harmfulâĂİ considered harmful: patterns of cloning in software. *Empirical Software Engineering*, 13(6):645–692, 2008.

[22] R. Kerr and W. Stuerzlinger. Context-sensitive cut, copy, and paste. In *Proceedings of the 2008 C 3 S 2 E conference*, pages 159–166. ACM, 2008.

[23] M. Kim, L. Bergman, T. Lau, and D. Notkin. An ethnographic study of copy and paste programming practices in oopl. In *Empirical Software Engineering, 2004. ISESE'04. Proceedings. 2004 International Symposium on*, pages 83–92. IEEE, 2004.

[24] R. Komondoor and S. Horwitz. Using slicing to identify duplication in source code. In *SAS '01 Proceedings of the 8th International Symposium on Static Analysis*, pages 40–56. ACM, 2001.

[25] J. Krinke. Identifying similar code with program dependence graphs. In *Proceedings of the Eighth Working Conference on Reverse Engineering (WCRE'01)*, pages 301–309. IEEE Computer Society, 2001.

[26] B. Lague, D. Proulx, J. Mayrand, E. M. Merlo, and J. Hudepohl. Assessing the benefits of incorporating function clone detection in a development process. In *Software Maintenance, 1997. Proceedings., International Conference on*, pages 314–321. IEEE, 1997.

[27] 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*, volume 4, pages 289–302, 2004.

[28] F. Rahman, C. Bird, and P. Devanbu. Clones: what is that smell? *Empirical Software Engineering*, 17(4-5):503–530, 2012.

[29] C. K. Roy and J. R. Cordy. A survey on software clone detection research. Technical report, Technical Report 541, QueenâĂŹs University at Kingston, 2007.

[30] 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*, pages 172–181. IEEE, 2008.

[31] H. Sajnani, V. Saini, and C. Lopes. A parallel and efficient approach to large scale clone detection. *Journal of Software: Evolution and Process*, pages n/a–n/a, 2015.

[32] H. Sajnani, V. Saini, and C. V. Lopes. A comparative study of bug patterns in java cloned and non-cloned code. In *Source Code Analysis and Manipulation (SCAM), 2014 IEEE 14th International Working Conference on*, pages 21–30. IEEE, 2014.

[33] N. E. Schwarz, E. Wernli, and A. Kuhn. Hot clones, maintaining a link between software clones across repositories. In *Proceedings of the 4th International Workshop on Software Clones*, pages 81–82. ACM, 2010.

[34] M. Toomim, A. Begel, and S. Graham. Managing duplicated code with linked editing. In *IEEE Symposium on Visual Languages and Human Centric Computing*, pages 173–180, 2004.