# Proactively Managing Copy-and-Paste Induced Code Clones

Daqing Hou, Ferosh Jacob, and Patricia Jablonski
Electrical and Computer Engineering, Clarkson University, Potsdam, NY 13699
{dhou, jacobf, jablonpa}@clarkson.edu

## Abstract

*Programmers copy and paste code. As a result, similar code fragments (clones) are added into software systems. Like other software artifacts, clones require attention and effort from programmers so that they can be found, understood, and correctly adapted and evolved. In addition to what clone-detection-based tools can offer, other automated support can be developed to better assist programmers in these activities, for example, to compare and contrast code clones, or help edit (a group of) clones consistently and quickly. We describe several such features currently being developed in the CnP project on top of Eclipse and for Java.*

## 1. Introduction

Copying and pasting code often results in duplication, or clones. A clone introduces a dependency between the original and the new code. When the complexity of such dependencies surpasses a programmer's capability of handling them, it starts to create problems, as observed in [4]. It is even a challenge to simply know the locations of clones in a large system, which has motivated the large body of work on clone detection tools and techniques [5]. However, clone detection tools tend to have low precision and recall, may miss type-3, gapped clones, or suffer from poor performance [1]; prior tools either rely on clone detection techniques or require programmers to manually specify clones, e.g., [2, 6]. Better can be done.

We believe that clones should be viewed and supported proactively from a process point of view. As shown by the four rectangles in the middle of Figure 1, a proactive clone management system will assist programmers in capturing clones, visualizing "important" information about clones (for example, the correspondence between two or more clones), editing clones, and removing cloning relationships (clone divergence). In this way, proactive support directly manages clones during active development.

In this demonstration paper, we describe three such features developed in the CnP project [1]. Two other CnP features, LexId for renaming shared common identifier parts and clone importing, are skipped due to space limitation.

[1] www.clarkson.edu/~dhou/projects/CnP

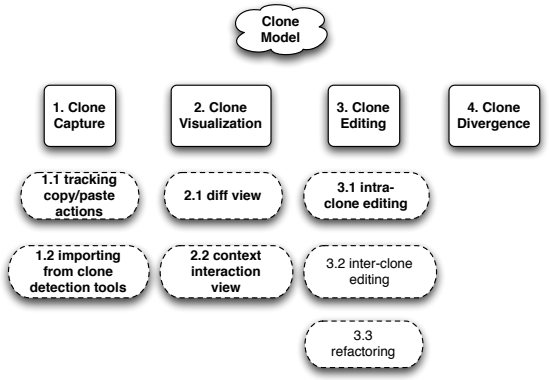



Figure 1: Clone Life Cycle and Possible Tools in Each Stage. Areas Where CnP Has Support Are Shown in Bold.

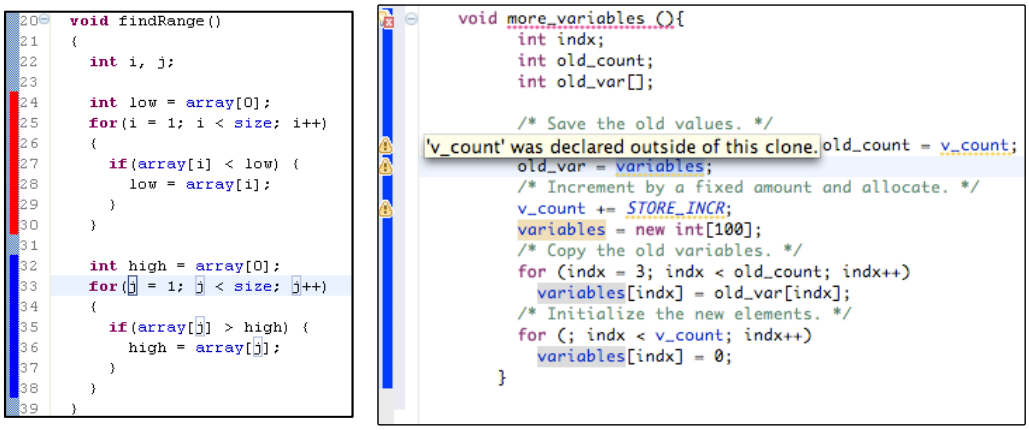
All CnP features rely on a fundamental feature that actively tracks copy-and-paste induced clones (clone tracking).

## 2. Existing CnP features

CnP automatically detects the creation of new clones by intercepting the copying and pasting actions, which may happen within both editors and views and in multiple ways. After clones are created, CnP will accurately maintain their locations when code is added or deleted before or within the clone regions. As long as clones are evolved within the IDE, it is guaranteed than CnP always provides accurate clone locations. CnP also persists the clone model between IDE sessions. [2] CnP currently shows clones visually by displaying colored bars next to them in the editor, shown in Figure 2a.

CnP contains a renaming utility (CReN) [3] that helps rename identifiers consistently *within* clones or any programmer-selected region, shown in Figure 2a. In this way, CReN helps speed up coding efficiency. Perhaps more important, CReN also helps prevent inconsistent renaming errors because manual renaming can miss an instance that was intended to be renamed. The most common use case for CReN is an *open* code fragment that contains *free* identifiers whose definitions appear outside of the fragment. CReN

[2] Support for collaborative work among multiple developers is under investigation as a separate issue.




(a) Clone Tracking with CnP (Original Code Shown with Red Bar and Pasted Code with Blue) and Consistent Renaming with CReN.

(b) Warnings about Accidental Identifier Capture within a Clone.

Figure 2: CnP Features (Clone Tracking, Consistent Renaming, and Accidental Identifier Capture).

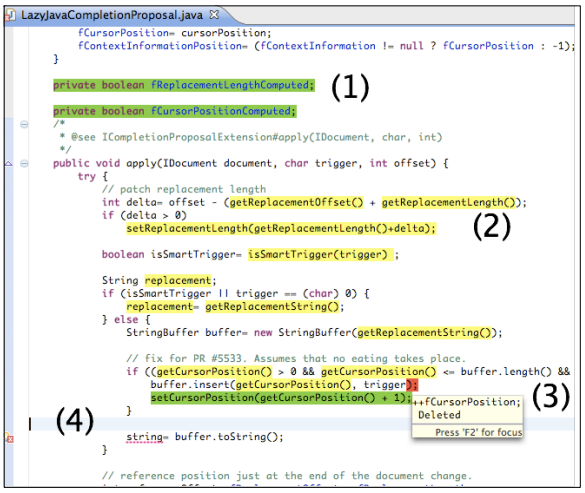



Figure 3: CSeR (Code SEgment Reuse) Incrementally Tracks Edits Made to a Copied-and-Pasted Class and Visualizes Code That Is Added, Updated, and Deleted with Different Colors inside the Editor (Figure Better Viewed Online to See the Colors).

can be beneficial in situations where existing Rename refactorings fall short.

Another feature of CnP is warning about the accidental capture of identifiers (Context Interaction in Figure 1). CnP issues a warning if an identifier in the pasted code binds to a declaration in an enclosing scope. For example, "more_variables()" in Figure 2b was copied and pasted within the same class. CnP provides three warnings for "v_count", "variables", and "STORE_INCR", which are all fields in the class, alerting the programmer that these particular identifiers within the clone may need to be renamed. The programmer can then use CReN to do so, if desired.

Once created, a clone may be subsequently evolved, where code can be added, updated, or deleted. A feature of CnP named CSeR (Code SEgment Reuse) visually highlights these three kinds of change directly in the editor (Diff View in Figure 1). Figure 3 depicts that LazyJavaCompletionProposal was copied from JavaCompletionProposal in jdt.ui of Eclipse 3.2. Specifically, location (1) contains two fields that are newly added, colored in green. The multiple code elements colored in yellow around location (2) are the results of replacing direct access to fields by getters and setters and long boolean expressions by predicates. At location (3), a red marker shows that there are some codes deleted after it, and the tooltip shows the code that was deleted. Finally, near location (4), there is a compilation error on the identifier string. This is because the name of this variable has been changed to replacement a few lines above.

## References


[1] S. Bellon, R. Koschke, G. Antoniol, J. Krinke, and E. Merlo. Comparison and Evaluation of Clone Detection Tools. *IEEE Transactions on Software Engineering*, 33(9):577–591, 2007.

[2] E. Duala-Ekoko and M. Robillard. Tracking Code Clones in Evolving Software. In *ICSE*, 2007.

[3] P. Jablonski and D. Hou. CReN: A Tool for Tracking Copy-and-Paste Code Clones and Renaming Identifiers Consistently in the IDE. In *ETX Workshop at OOPSLA*, 2007.

[4] A. J. Ko, H. Aung, and B. A. Myers. Eliciting Design Requirements for Maintenance-Oriented IDEs: a Detailed Study of Corrective and Perfective Maintenance Tasks. In *ICSE*, 2005.

[5] R. Koschke. Survey of Research on Software Clones. In *Dagstuhl Seminar Proceedings*, 2006.

[6] M. Toomim, A. Begel, and S. Graham. Managing Duplicated Code with Linked Editing. In *IEEE VL/HCC*, 2004.