Renaming Parts of Identifiers Consistently within Code Clones

Patricia Jablonski and Daqing Hou
School of Engineering
Clarkson University
Potsdam, NY 13699
{jablonpa, dhou}@clarkson.edu

***Abstract*—Copying and pasting source code results in code duplication. A common form of software reuse involves modifying the new duplicate to fit a current task. The similar code fragments (code clones) may be edited inconsistently by the programmer, for various reasons, leaving a bug in the software that may remain undetected by both the programmer and the compiler. A previously published tool, CReN, helps the programmer by automatically renaming all instances of the same identifier consistently within a clone when one is edited. In this tool demo, we introduce an extension of CReN, an Eclipse plug-in named LexId, which renames the same parts of different identifiers consistently together within code clones.**

Keywords-code clone; copy-and-paste programming; Eclipse integrated development environment; identifier renaming; Java.

I. INTRODUCTION

The copy, paste, and modify programming practice often involves the modification of a newly pasted code fragment (which is at first identical to the original copied code) to fit a current task. The programmer considered it beneficial to create a new solution in this way [4]. Often the programmer only needs to make small changes to the pasted code, like to the identifier names and literal constants [1, 3]. The copied and pasted code fragments (copy-and-paste-induced code clones) usually remain textually similar by definition [2] as otherwise the code would not have been duplicated. In this type of editing, the changes may be intended to be made only within a single clone rather than between the related clones. Although the physical change itself is not to be repeated across the clones, a correspondence relationship (similarity) still exists between the clones that must be maintained.

Regardless of whether the amount of changes to be made are small or not, the programmer can still make a mistake and leave an instance of an identifier unchanged that was supposed to be renamed with the others. This kind of error can remain undetected by both the programmer and the compiler, since the compiler would not give a warning, for example, if the unedited identifier instance is syntactically correct and still in scope. The compiler does not infer the programmer's intention and does not know that an instance was not renamed with others.

We have a previously published tool, named CReN, which addresses this issue by consistently renaming all instances of the same identifier within a clone together when one instance is being edited by the programmer [8]. CReN also allows the user to control which identifier instances are renamed together if this default behavior of consistent renaming is not desired. Furthermore, CReN was improved to rename within any user-specified code fragment, not only within clones that were made via copying and pasting. The CReN base was then extended to create a new tool, named LexId, which handles a different use case than CReN and instead focuses on inferring lexical patterns across different identifier instances, but still within a clone.

II. THE LEXID TOOL

CReN and LexId are both plug-ins that utilize the abstract syntax tree (AST) source code representation that is available in the Eclipse framework. First, the tools track the cloning relationship right when the code is copied and pasted before any changes are made. Each clone's location is accurately tracked according to its starting character position and length in number of characters within a source code file. Only copied and pasted code that is fully contained within an AST node is captured in our model. Related clones from the same copy and paste sequence are also noted.

In addition to clone tracking, CReN and LexId track identifiers within these related clones. First, we match the identifier instance locations between the clones (which are AST leaf nodes of type SimpleName), which represents the correspondence relationship. (Note: this correspondence is not used by CReN or LexId yet). Then, we group together all of the same identifier instances, which are assumed to be renamed together consistently. This way when the programmer edits any one of the identifier instances, all others of the same program element or name are renamed with it automatically and consistently. All identifier instances that are currently being edited within a clone are shown boxed (similar to Eclipse's Linked Renaming).

LexId further adds onto this default functionality of CReN by tracking and grouping together common substrings between the different identifiers within a clone. LexId tracks corresponding identifier pieces and renames these identical parts of identifier names consistently together within copied and pasted code fragments. All instances of a common substring between all identifiers within a clone are renamed together as one of those is renamed by the programmer.

LexId determines substrings as those parts of an identifier that are separated by an underscore "_" or dollar sign "$" character, or by changes in character type (digits or letters) or case (uppercase or lowercase letters). The "_" and "$" are never substrings or part of a substring (they are strictly only separation characters). The standard Java CamelCase naming convention is supported.

A use case for LexId substring renaming is in GUI software. First, GUI and web software often contain many common code fragments that were intentionally copied and pasted and not abstracted away into procedures [11, 12]. Second, they often contain symmetric naming conventions like "leftButton" and "rightButton", or "topPanel" and "bottomPanel". Figure 1 shows an example from production code [the code that is shown is used by permission from Toshihiro Kamiya: http://www.ccfinder.net/img/snapshot-e/4.png]. In this example, an if block was copied and pasted. In the pasted code (on the bottom), the substring "left" was modified to become "right" in all instances. The "left" substring was a common substring between the three different identifiers "leftFile", "leftBegin", and "leftEnd", which needed to be renamed to "rightFile", "rightBegin", and "rightEnd" to fit the new, related task. Programmers can copy and paste when following a certain naming scheme such as this one. In this situation, some substrings in the pasted code remain the same to instances in the original code (the "File", "Begin", and "End"), while the modified portion ("left" to "right") follows a convention or standard.

```
if (p.leftFile == SourcePane.this.fileIndex){
  int begin = tokens[p.leftBegin].beginIndex;
  int end = tokens[p.leftEnd - 1].endIndex;
  if (begin <= pos && pos < end){
    ids.add(new Long(p.classID));
    selectedClonePairs[i] = true;
  }
}

if (p.rightFile == SourcePane.this.fileIndex){
  int begin = tokens[p.rightBegin].beginIndex;
  int end = tokens[p.rightEnd - 1].endIndex;
  if (begin <= pos && pos < end){
    ids.add(new Long(p.classID));
    selectedClonePairs[i] = true;
  }
}
```

Figure 1. LexId changes the substrings "left" to "right" in the pasted if block (on the bottom) when one of those substring instances is edited.

III. LexId User Study and Related Work

A user study was conducted on LexId to test its ability to assist programmers in renaming within clones both quickly and correctly [7]. Subjects used the renaming tools already in the IDE or they used copy/paste or manual typing to rename when not using LexId. We concluded from this study that LexId performed statistically quicker than without it on similar tasks within a GUI Paint program and that LexId helps prevent inconsistent renaming that otherwise happens.

Our prior work [7, 8] gives some reasons why the related renaming tools in Eclipse (Find & Replace, Rename Refactoring, Linked Renaming, and Rename Type Refactoring) each has its own set of limitations and differences from LexId. A related identifier tracking tool, Vaci [9], can detect possible renaming inconsistencies, but LexId instead can proactively prevent the inconsistencies from occurring at all.

IV. Conclusion and Future Work

LexId is a plug-in for the Eclipse integrated development environment that renames common substrings consistently within copied and pasted code clones. LexId works well with source code that has identifier names that follow a naming convention such that some substrings remain the same, while others are meant to be modified together. While LexId's current algorithm for identifier splitting is quite simple, more sophisticated approaches such as those in [5, 6, 10] could be exploited as future work to more accurately capture the semantic information that is embedded inside of identifiers.

Also in the future, LexId can be made to automatically infer the substring "right" in the pasted code (as in Figure 1) based on the substring "left" by maintaining a database of common naming pairs ("left/right", "top/bottom", etc.). The inferred substrings can then be recommended to the programmer. In addition to inferring patterns in identifiers, LexId could also include support for tokens (numbers as literal constants or for array access). Finally, LexId could prevent other inconsistencies by inferring type and subtype patterns in classes (similar to Rename Type Refactoring).

References

[1] B.S. Baker, "Parameterized duplication in strings: algorithms and an application to software maintenance", *SIAM Journal on Computing*, 1997.

[2] S. Bellon, R. Koschke, G. Antoniol, J. Krinke, and E. Merlo, "Comparison and evaluation of clone detection tools", *IEEE Transactions on Software Engineering (TSE)*, 2007.

[3] M. de Wit, A. Zaidman, and A. van Deursen, "Managing code clones using dynamic change tracking and resolution", *IEEE International Conference on Software Maintenance (ICSM)*, 2009.

[4] F. Detienne, "Reasoning from a schema and from an analog in software code reuse", *Workshop on Empirical Studies of Programmers (ESP)*, 1991.

[5] E. Enslen, E. Hill, L. Pollock, and K. Vijay-Shanker, "Mining source code to automatically split identifiers for software analysis", *IEEE International Working Conference on Mining Software Repositories (MSR)*, 2009.

[6] H. Feild, D. Binkley, and D. Lawrie, "An empirical comparison of techniques for extracting concept abbreviations from identifiers", *IASTED International Conference on Software Engineering and Applications (SEA)*, 2006.

[7] P. Jablonski and D. Hou, "Aiding software maintenance with copy-and-paste clone-awareness", *IEEE International Conference on Program Comprehension (ICPC)*, 2010.

[8] P. Jablonski and D. Hou, "CReN: a tool for tracking copy-and-paste code clones and renaming identifiers consistently in the IDE", *Eclipse Technology Exchange Workshop at OOPSLA (ETX)*, 2007.

[9] T. Kamiya, "Variation analysis of context-sharing identifiers with code clones", *IEEE International Conference on Software Maintenance (ICSM)*, 2008.

[10] D. Lawrie, H. Feild, and D. Binkley, "Extracting meaning from abbreviated identifiers", *IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM)*, 2007.

[11] D.C. Rajapakse and S. Jarzabek, "Using server pages to unify clones in web applications: a trade-off analysis", *ACM SIGSOFT-IEEE International Conference on Software Engineering (ICSE)*, 2007.

[12] M.B. Rosson and J.M. Carroll, "The reuse of uses in smalltalk programming", *ACM Transactions on Computer-Human Interaction (TOCHI)*, 1996.