CSeR: A Code Editor For Tracking And Highlighting Detailed Clone Differences

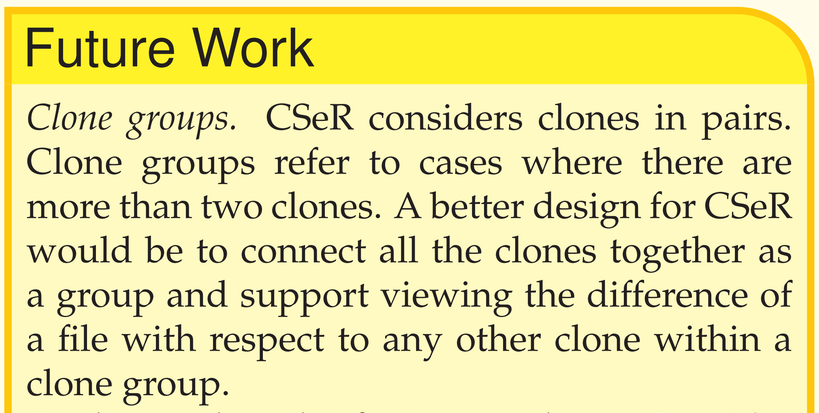
Ferosh Jacob, Jeff Gray - {jacobf, gray}@cis.uab.edu - University of Alabama at Birmingham
Daqing Hou - dhou@clarkson.edu - Clarkson University

Introduction

We designed and implemented a code editor named CSeR (Code Structure Reuse), which keeps a record of clones created by "Copy and Paste" operations. In addition, CSeR detects and highlights the changes made to a clone with distinct colors using an edit-based algorithm. An empirical study was conducted with 37 test cases collected from both industry and research projects to test the robustness and usefulness of the tool. A total of 533 changes were identified and categorized into 20 different types. A comparative study with related tools is included, which demonstrates the uniqueness of CSeR.

Implementation

CSeR is implemented as an Eclipse plugin that extends the Java editor without disturbing any of its existing features. The changes will be shown directly in the Eclipse code editor. Changes representing inserted or new (AST) nodes are shown in green, removed nodes in red, updated nodes in yellow, and moved nodes in blue. Mouse hover events will reveal more details about the change.



Copy and Paste operations are tracked within the development environment and the copied and pasted code is identified. The initial step requires parsing the code of a class to obtain subtrees to build up the correspondences among the clones. When a class is modified within a code editor, the smallest block that contains the change will be identified and the corresponding block will be compared to identify what kind of change has been made.

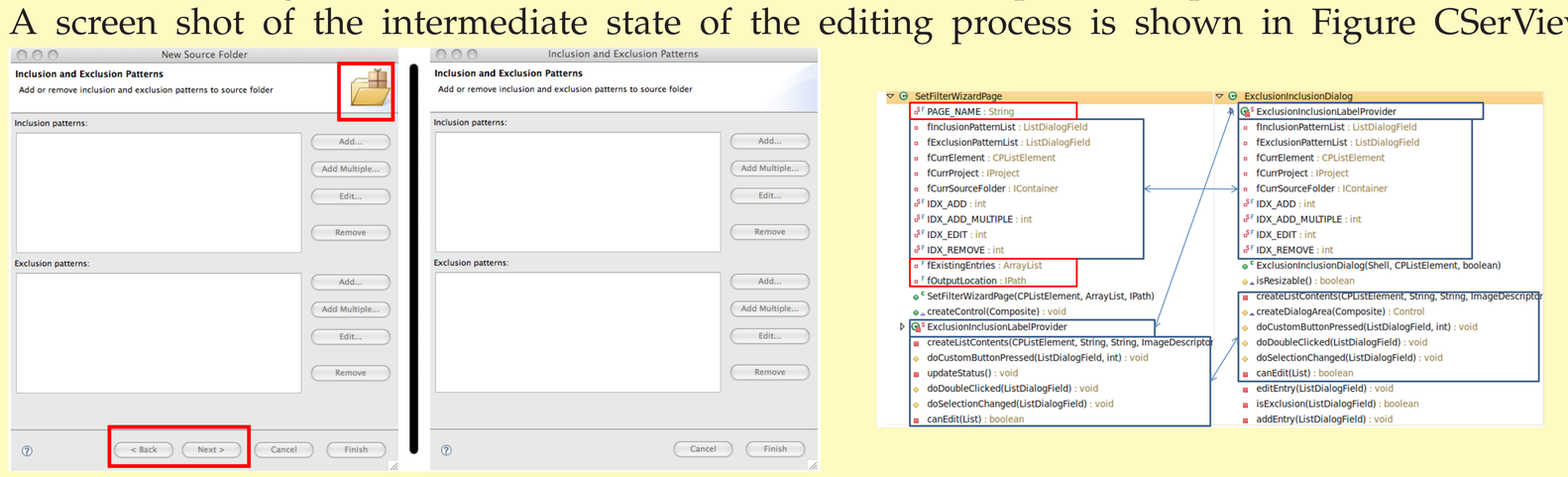
Future Work

Clone groups. CSeR considers clones in pairs. Clone groups refer to cases where there are more than two clones. A better design for CSeR would be to connect all the clones together as a group and support viewing the difference of a file with respect to any other clone within a clone group.

Tracking code and inferring templates. Consider a scenario in which a developer is working on a large file and something is not working and he is not sure what he has changed. CSeR could be extended to track changes of code between two time frames of editing. CSeR can even provide a capability to go back before an edit. Since CSeR calculates the editing regions in clones, we can integrate it with the Eclipse template feature. For example, while pasting a few lines of code, an option is provided to activate CSeR templates. CSeR calculates the editing regions and passes the information to the template model to form a new template.

Example Scenario

A study of the two classes (`SetFilterWizardPage`, `ExclusionInclusionDialog`) reveals that there is a good chance that the two classes were copied and pasted from each other. A screen shot of the intermediate state of the editing process is shown in Figure CSerView.



a) External View b) Outline,Source View

```
58 public class ExclusionInclusionDialog extends StatusDialog {1   {NewElementWizardPage->StatusDialog}
59                                                                  Press 'F2' for focus
60     private static class ExclusionInclusionLabelProvider extends LabelProvider {
61
62         private Image fElementImage;
63
64         public ExclusionInclusionLabelProvider(ImageDescriptor descriptor) {
65             ImageDescriptorRegistry registry= JavaPlugin.getImageDescriptorRegistry();
66             fElementImage= registry.get(descriptor);
67         }
68
69         public Image getImage(Object element) {
70             return fElementImage;
71         }
72
73         public String getText(Object element) {
74             return BasicElementLabels.getFilePattern((String) element);
75         }
76
77     }
78
79
80     private ListDialogField fInclusionPatternList;  2   private static final String PAGE_NAME="SetFilterWizardPage";
81     private ListDialogField fExclusionPatternList;                                   Press 'F2' for focus
82     private CPListElement fCurrElement;
83     private IProject fCurrProject;
84
85     private IContainer fCurrSourceFolder;
86
87     private static final int IDX_ADD= 0;
88     private static final int IDX_ADD_MULTIPLE= 1;
89     private static final int IDX_EDIT= 2;
90     private static final int IDX_REMOVE= 4;  3
91
92                                                  4                5                  6
93     public ExclusionInclusionDialog(Shell parent,CPListElement entryToEdit, boolean focusOnExcluded) {
94         super(parent);  7
95         fCurrElement= entryToEdit;  8
96         setTitle(NewWizardMessages.ExclusionInclusionDialog_title);
```

c) CSeRView

In the figure above, the change marked '1' is categorized as an update change (change of superclass from `StatusDialog` to `NewElementWizardPage`). The changes shown in green represent new code. For example, '2' is a new field and '6' and '7' are two newly added parameters. Code that remains unchanged is shown normally, e.g., those lines near '3'. The two changes marked '4' and '5' correspond to the deletion of two parameters before and after `entryToEdit`, respectively. Finally, the change marked '8' indicates a move operation from several lines above. In case of update and delete changes, the original code can be seen within a popup box when the mouse is positioned near the marker, as shown in change '1'.

Validation

Robustness The kinds of edits that CSeR supports are shown in Table 1. For every edit, there will be one *goal*, which is the purpose of editing while there can be different ways to achieve it, *actions*. Names refer to anything that is not a keyword in Java. Names can be method names, class names, and variable names. Lists correspond to structures which appear between list delimiters such as {}s surrounding lists of statements and the ()s surrounding lists of parameters. List elements are delimited by single characters such as ";"s between statements and ","s between parameters.

No	Type	Goal Description	Action Description	Implemented
1	Names	Creating a name	Paste or Type	√
2		Replacing part	Paste or Type	√
3		Correcting typos	Backspace and Type	√
4		Replacing name	Backspace, Type or Paste	√
5		Removing name	Backspace, Delete or Type	√
6		Splitting a name	Type in between	√
7		Renaming	Using tools	×
8	Lists	Creating a new list	Type or Paste	√
9		Inserting a new element	Type or Paste	√
10		Removing an element	Delete, Type or Backspace	√
11		Moving an element	Cut and Paste or Copy Paste and Delete	√
12		Removing entire list	Backspace or Delete	√
13		Flattening a list inside a list	Backspace or Delete	√
14	Expressions	Inserting a new expression	Type or Paste	√
15		Updating an expression	Type or Paste	√
16		Removing an expression	Delete, Type or Backspace	√
17		Moving an expression	Cut and Paste or Copy Paste and Delete	√
18	Comments	Comment code	Type Line or Block comment	√
19		Creating annotations		×
20		Inside expressions	Type Block comments	√
21	Keywords	Insert keyword		×
22		Update keyword		×
23		modify keyword		×

Table 1. Common code edits and CSeR

No	Change Distribution	Description	Internal Distribution
1	Update (49 %, 261)	Variable Name (V)	49 %
2		Variable Type (T)	26 %
3		Method (M)	15 %
4		Literal (L)	8 %
5		Other (O)	<2 %
6	Delete (33 %, 177)	Statement (S)	40 %
7		Method Declaration (M)	28 %
8		Field Declaration (F)	21 %
9		Expression (E)	8 %
10		Parameter (P)	< 1 %
11		Class Declaration (C)	< 1 %
12	Insert (16 %, 82)	Statement (S)	46 %
16		Field Declaration (F)	26 %
15		Method Declaration (M)	14 %
13		Parameter (P)	10 %
14		Expression (E)	2 %
17		Class Declaration (C)	2 %
18	Move (2 %, 13)	Method Declaration (M)	54 %
19		Statement (S)	39 %
20		Class Declaration (C)	7 %

Table 2. Summary of the 533 changes

Experiment Results Table 2 shows change classification from the test cases (37 pairs of cloned classes and code fragments) collected from three sources: Eclipse, more specifically from JDT UI, JDT Core and SWT projects; JavaLobby Community Platform (JLCP), a project that aims to write a number of components to produce a free Java Portal site based on an internal forum system; research literature (10 pairs of clones). The clones from the literature deal with extreme cases, and also from diverse domains.