Weekly Meeting 10/6/2015

Focus on Problem 2, find examples to prove this problem

1. LAST WEEK

Based on our discussion last week, our problem statement includes:

1. Developers do not know the name of API elements a priori, though locating code example through existing search engine often expects the developers to know API elements' names.

2. Reuse tasks involve multiple related classes and methods, rather than a single method/class. Yet code search engines do not help developers recognize the latent structure among related classes/methods. (at the granularity of multiple methods)

3. During copy/paste based reuse, developers must remove irrelevant parts. Our hypothesis is that having multiple examples of the same kind will help winnow out irrelevant parts. (at the granularity of multiple examples)

We have three hypotheses correspondingly:

1. We believe that NLP text with partial code snippet could help us identify the name of API elements based on keywords from free-form query.

2. Our hypothesis is that we can find related elements by following the structural dependencies of seed API elements in code and by ranking other API elements potentially mentioned in natural language text.

3. We believe that clustering multiple examples of the same kind and finding the commonality may help remove irrelevant parts.

I focus on the second problem, and perform a preliminary study using popular code search engines to illustrate this problem.

(A) User's context

```
import com.sun.java.swing.*;

public class MyTextEditor {
  public void init() {
    JFrame frame = new JFrame("Undo Sample");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    JTextArea textArea = new JTextArea();
   JButton undoBtn_;
   //add undo and redo action to text editor
  }}
```

(E) Expected reuse plan based on the query 'undo redo TextEditor'

```
import com.sun.java.swing.*;
import javax.swing.undo.UndoManager;
import javax.swing.event.UndoableEditListener;
import javax.swing.AbstractAction;

public class MyTextEditor {
    JFrame frame = new JFrame("Undo Sample");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    JTextArea textArea = new JTextArea();
    JButton undoBtn_;
    JButton redoBtn_;

   public void init() {
   //add undo and redo action to text editor
   UndoManager undoManager = new UndoManager();
   textArea.getDocument().addUndoableEditListener(new UndoListener());
   undoBtn_ = new Button();
   undoBtn_.addActionListener(new UndoAction());
   redoBtn_ = new Button().addActionListener(new RedoAction());
   redoBtn_.addActionListener(new RedoAction());
   }
 private class UndoListener implements UndoableEditListener {
  public void undoableEditHappened(UndoableEditEvent e) {
    undoManager.addEdit(e.getEdit());
  }}
private class UndoAction extends AbstractAction {
  public void actionPerformed(ActionEvent e) {
    undoManager.undo();
  }}
private class RedoAction extends AbstractAction {
  public void actionPerformed(ActionEvent e) {
     undoManager.redo();
  }}}
```

Figure 1: Scenario 1: add undo/redo to a TextEditor

2. PROBLEM 2

Problem: Reuse tasks involve multiple related classes and methods, yet code search engines do not help developers recognize the latent structure among related classes/methods.

Hypothesis: We can find and rank related elements by following the structural dependencies of seed API elements in code.

Table 1: Evaluation dataset for feature location tools and concern mining tools

Tool	Input: query / context	Output: method / code snippet
FCA [30]	jEdit: support for 'thick' caret jEdit: rename project source	6 (org.eclipse.swt.widgets.Table.createHandle(), createWidget(), etc) 5 (org.eclipse.core.internal.localstore.FileSystemStore.move, etc)
Portfolio [23] Export [27]	mip map dithering texture image graphics adjust parameters such as brightness for a picture* read and play midi files*	6 methods: ImageTexture:createTextureFromImage, TiledTexture: buildGaussianGlareTexture, LoadTextureFromFile:initReaderer, etc
Rastkar, Murphy et al. [31]	jEdit: add autosave capabilities add undo functionality for a command	They assume that there already exist multiple instances of this concern. 'All of the methods named 'undo': override method 'org.jhotdraw.util.UndoableAdapter.undo'; are a member of the class 'undoActivity';

Strathcona [14]	update status line
	
	generate method signature
		protected void updateStatusLine(IStructuredSelection selection) {//fill
		String msg = getStatusLineMessage(selection);
		getViewSite().getActionBars().getStatusLineManager().setMessage(msg); }
		
		public boolean visit(MethodDeclaration node) {//fill
		node.getModifiers();
		node.getName.getIdentifiers();
		return super.visit(node); }

GraPacc [28]	SWT usage pattern	Display display = new Display();
		Shell shell = new Shell(display); // fill the rest
		shell.open()
		while (!shell.isDisposed()) {
		if (!display.readAndDispatch())
		display.sleep();
		}
		display.dispose();

* represents tasks given to the users. However, the free-form query users use to finish the task is unknown.

Table 2: Evaluation dataset for reuse tool

Tool	Task	LOC	Result / Description
Gilligan [16]	reuse BMP image file parsing in SWT reuse module that serializes objects to XML reuse virtual file system from a third-party app reuse GraphML parsing code from Jung reuse charting component from Azureus reuse NMEA parser from the Gpsylon project reuse the lines of code counter from Metrics project reuse QIF parser from jGnash project	497 900 3000 200	They try to address the problem of source code integration in the context of medium or large-scale reuse tasks. They suggest program elements that are easy to reuse based on structural relevance and cost of reuse in the source context, and guide users to plan a non-trivial reuse task. They assume that developers have a perfect example at hand, and they can finish the reuse task by resolving all dependency conflicts.

LOC represents the lines of code that are reused.

Input: a free-form query that user provides to present her reuse intention.

Output: a list of classes/methods implementation that complete this reuse task.

Sub questions: Do reuse tasks involve multiple related classes and methods? Can existing tools find reusable examples that involve multiple related classes and methods?

To answer these two subquestions, I search for the evaluation dataset for existing feature location/concern mining tools [1,5,30,36] shown in Table 1, reuse tools [15,16] shown in Table 2, and code search tools using free-form query [4,7,13,17,23] shown in Table 3. We find that most medium-scale or large-scale reuse tasks involve multiple methods and classes, and the developer should implement multiple methods and classes to finish the reuse task. Existing code completion tools can complete a single method which involves in multiple API calls, yet they assume that all related APIs or objects used in their suggested code snippet already exist. We argue that this assumption might not hold for a non-trivial reuse task, because developers need to implement the salient API calls that provide the main functionality as well as other related elements based on the latent structure.

3. USAGE SCENARIO

Add undo and redo action for TextEditor.

Table 4: Search Result from Code Search Engine

No.	Name	Project	LOC	# M
1	TextEditor	textmash	1270	12
2,3	AndroidTextEditor	android	576	9
4	DocumentUndoManagerImpl	ide	1227	17
5,6,8	ConsoleTextEditor	groovy	321	7
7	PapyrusCDTEditor	eclipse.papyrus	393	7
9,10	AspectEditorContributor	eclipse	88	2

 # M represents the number of methods that contain the query terms. **No.** represents the rank from CSE. Note that CSE may return the same results for multiple times, as the same file may exist in multiple branches. For instance, we regard 5,6,8 as identical with manual inspection.

To further illustrate that reuse task involves in multiple methods and classes, and the reuse task is hard without tool support, I imagine a scenario that user wants to add undo and redo actions for her Java Swing text editor application with undo/redo buttons, inspired by the evaluation task used in [31]. Without knowing any APIs, developer first queries code search engine (CSE) with a free-form query `'undo redo TextEditor'`. Figure 5 illustrates the implementation of 'add undo/redo' concern mentioned in [31] and blue part represents source code that implements this feature. Table 4 shows top 10 results returned from Search-

Table 3: Evaluation dataset for code search tools that use free-form query or keyword query

Tool	Input: query / keyword	Output: code snippet / code element
SNIFF [7]	get active editor window from eclipse workbench	IWorkbenchWindow window = PlatformUI.getWorkbench().getActiveWorkbenchWindow();
	parse a java source and create ast	ASTParser parser = ASTParser.newParser(AST.JLS3); CompilationUnit cu = (CompilationUnit) parser.createAST(null);
	connect to a database using jdbc	Connection conn = DriverManager.getConnection(path)
Sourcer [4]	Find a XML parser	org.apache.jasper.xmlparser
	Find the use of Depth First Search	org.jeat.search.dfs.BasicDepthFirstSearch
	Find the use of Newton Raphson Method	org.cdk.NewtonRaphsonMethod
anyCode [13]	copy file fname to destination	FileUtils.copyFile(new File(fname), new File(destination))
	does x begin with y	x.startsWith(y)
	get the current year	new Date().getYear()
EVOLIZER [38]	What methods call addChart?	ServletUtilities: registerChartForDeletion(File, HttpSession)
	What method accesses the attribute labelFontField?	DefaultAxisEditor:attemptLabelFontSelection()
	What are the classes that extend JPanel?	DefaultAxisEditor
Keivanloo et al. [17]	{getOptionValue, CommandLine}	CommandLine.getValue(Option)
	{ISelection, isEmpty}	ISelection.isEmpty()
	{WebElement, click}	WebElement.click()

```
public class TextEditor extends JTextPane {
 public UndoAction undoAction = new UndoAction();
 public RedoAction redoAction = new RedoAction();
 public CompoundUndoManager undo;
 HashMap<Object, Action> actions = new HashMap<Object, Action>();
 public TextEditor(Workspace workspace) {
   this.workspace = workspace;
   undo = new CompoundUndoManager(workspace);
   actions.put("undo", undoAction);
   actions.put("redo", redoAction);
   }
 public void discardUndoRedo() {
    undo.discardAllEdits();
    undoAction.updateUndoState();
    redoAction.updateRedoState();
   }
 public class UndoAction extends AbstractAction {
  public void actionPerformed(ActionEvent e) {
   try {
     undo.undo();
     updateUndoState();
     redoAction.updateRedoState();
    } catch (CannotUndoException ex) {
    }}
   public void updateUndoState() {
    setEnabled(undo.canUndo());
   }}
 public class RedoAction extends AbstractAction {
   public void actionPerformed(ActionEvent e) {
    try {
      undo.redo();
      updateRedoState();
      undoAction.updateUndoState();
     } catch (CannotRedoException ex) {
     }}
      public void updateRedoState() {
       setEnabled(undo.canRedo());
     }}
```

Figure 2: Result No 1: textmash:TextEditor

```
public class ConsoleTextEditor extends JScrollPane {
  private UndoAction undoAction = new UndoAction();
    private RedoAction redoAction = new RedoAction();
    private TextUndoManager undoManager;
    public ConsoleTextEditor () {
     this.undoManager = new TextUndoManager();
     doc.addUndoableEditListener(undoManager);
     undoManager.addPropertyChangeListener(undoAction);
     undoManager.addPropertyChangeListener(redoAction);
     doc.addDocumentListener(undoAction);
     doc.addDocumentListener(redoAction);
  }
 private class RedoAction extends UpdateCaretListener
  implements PropertyChangeListener {
  public void actionPerformed(ActionEvent ae) {
   undoManager.redo();
   setEnabled(undoManager.canRedo());
   undoAction.setEnabled(undoManager.canUndo());
  }
    public void propertyChange(PropertyChangeEvent pce) {
     setEnabled(undoManager.canRedo());
   } }
 private class UndoAction extends UpdateCaretListener
 implements PropertyChangeListener {
   public void actionPerformed(ActionEvent ae) {
    undoManager.undo();
    setEnabled(undoManager.canUndo());
    redoAction.setEnabled(undoManager.canRedo());
 }
  public void propertyChange(PropertyChangeEvent pce) {
    setEnabled(undoManager.canUndo());
  }}
```

Figure 3: Result No 5,6,8: groovy: DocumentUndoManagerImpl

Code CSE. We find that all code examples require multiple methods and classes to implement a feature. We also find that all code examples include other auxiliary features that are not directed related to the undo and redo feature.

We assume that user provides a seed code example for the reuse task. She selects the first result from CSE. She uses keyword search to locate undo and redo feature in 6 methods and regards them as seed API calls. Shown in Figure 5, she removes the other 11 methods (865 LOC) that she regards as irrelevant to her reuse task. She recognizes that she needs to implement an `UndoAction` and `RedoAction` which are the subclass of `AbstractAction`, and overrides their actionPerform() method. In the actionPerformed() methods, she should invokes the `CompoundUndoManager.undo()` and `CompoundUndoManager.redo()` correspondingly. Without knowing anything about `CompoundUndoManager`, she has to query CSE again for 'CompoundUndoManager textmash'. This class consists of 17 methods (221 LOC) and she has to repeat to keyword search again to locate undo and redo feature in this class. With manual inspection, she notices that this `CompoundUndoManager` is a subclass of `javax.swing.UndoManager` and it overrides four methods {`canUndo, canRedo, undo, redo`} that are

related to the feature. She notices that she actually does not need this `CompoundUndoManager` and decides to invoke its parent class `UndoManager` instead. She also notices that she does not need the `CannotRedoException` by using `UndoManager`.

After she integrates `UndoManager, UndoAction, and RedoAction` to her context, she tests it and it fails to perform the feature. She has to look at other examples, and she finds the No.5 example returned from CSE. This example seems promising because it is also an implementation of a TextEditor. She notice that the API `doc.addUndoableEditListener (undoManager)` seems the one that she misses now, but again, she has to investigate the class `TextUndoManager` again.

With this example, we illustrate that salient API calls that implement a desired feature are always interleaving with other related elements. To finish a reuse task, developer has to remove irrelevant parts. However, it requires significant effort to tease out these auxiliary features without tool support. Since the examples always contain both main feature and auxiliary features, there seldom exists a perfect example for reuse and developers has to iteratively search for new examples and integrate it to their context until the integration results are examined.

4. PROBLEM 3

Problem: During copy/paste based reuse, developers must remove irrelevant parts and find salient API elements based on the context.

Hypothesis: Our hypothesis is that having multiple examples of the same kind will help winnow out irrelevant parts, i.e, clustering multiple examples of the same kind and finding the commonality may help remove irrelevant parts.

5. APPROACH

To identify methods that implement concerns, I build a prototype which invokes Code Search Engine (CSE) API and analyzes the results from CSE using partial program analysis [10]. I select SearchCode [34] because it is an open source code search engine with over 7000 projects from Github, Bitbucket, Google Code, and Sourceforge, with complete API documentations. To identify queried features in the returned source code, I use the mean of TF-IDF weight for each query term as a weighting factor and select top k (k=5) methods that are related to the given query. This approach is similar to prior works that use IR [30] and NL analysis [36] for feature location. I choose IR approach because other approaches require history or structural analysis that might not be feasible for partial program. TF-IDF = $avg(\log(1 + f_{t,d}) \times \log \frac{N}{n_t})$, $f_{t,d}$ is the frequency of term t in method d, N is the total number of methods, n_t is the number of methods that have the term t.

6. RELATED WORK

Feature Location Tools. Poshyvanyk et al. [30] use information retrieval approach to locate a queried feature in source code. They evaluate the similarity between documents and user query and cluster the source code based on formal concept analysis. Portforlio [23] and Export [27] identify related functions by combining both latent structure similarity and lexical information similarity. Our feature location approach is similar to [23] yet we focus on suggesting implementation for the feature rather than identify feature location. Rastkar et al. [31] summarize the structure of multiple instances of a crosscutting concern in natural language, yet they only extract structural facts in the level of method signature and class hierarchy. We make it one step further to suggest feature implementation based on the context and user query.

Code Search Tools. Our example clustering approach is similar to some prior works that extract representative examples for specific APIs or user query. MAPO [39] leverages frequent call sequences to cluster the usage of specific APIs and rank abstract usage patterns based on the context similarity. Buse et al [6] propose to generate abstract API usages by synthesizing code examples using symbolic execution for a particular API. Different from these works that generate abstract usage patterns for specific API or data type, SNIFF [7] performs type-based intersection of code chunks based on the keywords in the free-form query and cluster the common part of the code chunks for concrete code examples. However, these works only focus on providing code examples based on the popularity or textural relevance while developers have to manually resolve structural dependencies before reusing the examples. MUSE [26] addresses this limitation using slicing to generate concrete usage examples and selects the most representative ones based on the popularity and readability while Keivanloo et al [17] uses clone detection to cluster examples involving loops and conditions. We make it one step further to support partial code example clustering and identify structural correspondence to identify both common features and alternative features. PRIME [25] supports code search over partial programs based on type state transition, yet it requires users to provide partial temporal specification for generalized *typestate*. There exists a number of code example suggestion tools that recommend call chains [21, 33, 37] or contexts [14, 29]. But these tools can only recommend code examples in the method level which make them insufficient for reuse tasks across multiple classes.

Identify structural correspondence for code reuse Although some approaches advocate refactoring code rather than reuse code [12], recent researches have found that these kind of 'clone' cannot be easily refactored [18] and have to be modified to meeting requirements in new context [35]. Jigsaw [9] supports small-scale integration of source code into target system between the example and target context. Based on its ancestor [8] that identifies structural correspondence based on AST similarity, it greedily matches each element between two contexts, transforms correspondent elements to the target context, and simply copies the source element to the target if it does not correspond with any element in the target. Unfortunately, developer has to provide source and target to enable a one-to-one transformation and resolve all dependencies when pasting code to the target. Our approach overcomes these two limitations: we extract common functionalities from multiple examples and identifies how related elements interact with main features in a common way to resolve dependencies based on the mapping from the source to the target. Our idea of leveraging multiple examples to discover commonality and eliminate specificity is similar to LASE [24], which applies similar but not identical changes to multiple code locations based on context similarity. Our approach works in a similar manner of Programing-by-Example, but focuses on a task-based code reuse across different methods or even different classes, while LASE is confined to the systematic edit

within a single method and requires users to specify all input examples. Other related works on code reuse include Gilligan [16] and Procrustes [15] which try to address the problem of source code integration in the context of medium or large-scale reuse tasks. They automatically suggest program elements that are easy to reuse based on structural relevance and cost of reuse in the source context, and guide users to investigate and plan a non-trivial reuse task. They assume that developers have a perfect example at hand, and they can finish the reuse task by resolving all dependency conflicts and integrating the example to the desired context. However, we note that it is not easy to identify a good example as an example is always interleaving with other auxiliary features that should not be integrated. We observe that it is equally difficult, if not more so, to distinguish the major functionality and auxiliary ones from multiple examples than to identify related elements in a pragmatic reuse plan. We target the problem to identify the major features across different reusable examples and leverage Procrustes to evaluate the cost of reuse when recommending the best-fit reusable plan.

7. REFERENCES

[1] B. Adams, Z. M. Jiang, and A. E. Hassan. Identifying crosscutting concerns using historical code changes. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1, ICSE 2010, Cape Town, South Africa, 1-8 May 2010*, pages 305–314, 2010.

[2] G. Antoniol, G. Canfora, G. Casazza, A. D. Lucia, and E. Merlo. Recovering traceability links between code and documentation. *IEEE Trans. Software Eng.*, 28(10):970–983, 2002.

[3] A. Bacchelli, M. Lanza, and R. Robbes. Linking e-mails and source code artifacts. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1, ICSE 2010, Cape Town, South Africa, 1-8 May 2010*, pages 375–384, 2010.

[4] S. K. Bajracharya, J. Ossher, and C. V. Lopes. Sourcerer: An infrastructure for large-scale collection and analysis of open-source code. *Sci. Comput. Program.*, 79:241–259, 2014.

[5] S. Breu and J. Krinke. Aspect mining using event traces. In *19th IEEE International Conference on Automated Software Engineering (ASE 2004), 20-25 September 2004, Linz, Austria*, pages 310–315, 2004.

[6] R. P. L. Buse and W. Weimer. Synthesizing API usage examples. In *34th International Conference on Software Engineering, ICSE 2012, June 2-9, 2012, Zurich, Switzerland*, pages 782–792, 2012.

[7] S. Chatterjee, S. Juvekar, and K. Sen. SNIFF: A search engine for java using free-form queries. In *Fundamental Approaches to Software Engineering, 12th International Conference, FASE 2009, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2009, York, UK, March 22-29, 2009. Proceedings*, pages 385–400, 2009.

[8] R. Cottrell, J. J. C. Chang, R. J. Walker, and J. Denzinger. Determining detailed structural correspondence for generalization tasks. In *Proceedings of the 6th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2007, Dubrovnik, Croatia, September 3-7, 2007*, pages 165–174, 2007.

[9] R. Cottrell, R. J. Walker, and J. Denzinger. Semi-automating small-scale source code reuse via structural correspondence. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2008, Atlanta, Georgia, USA, November 9-14, 2008*, pages 214–225, 2008.

[10] B. Dagenais and L. J. Hendren. Enabling static analysis for partial java programs. In *Proceedings of the 23rd Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2008, October 19-23, 2008, Nashville, TN, USA*, pages 313–328, 2008.

[11] B. Dagenais and M. P. Robillard. Recovering traceability links between an API and its learning resources. In *34th International Conference on Software Engineering, ICSE 2012, June 2-9, 2012, Zurich, Switzerland*, pages 47–57, 2012.

[12] M. Fowler. *Refactoring - Improving the Design of Existing Code*. Addison Wesley object technology series. Addison-Wesley, 1999.

[13] T. Gvero and V. Kuncak. Interactive synthesis using free-form queries. In *37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Florence, Italy, May 16-24, 2015, Volume 2*, pages 689–692, 2015.

[14] R. Holmes and G. C. Murphy. Using structural context to recommend source code examples. In *27th International Conference on Software Engineering (ICSE 2005), 15-21 May 2005, St. Louis, Missouri, USA*, pages 117–125, 2005.

[15] R. Holmes, T. Ratchford, M. P. Robillard, and R. J. Walker. Automatically recommending triage decisions for pragmatic reuse tasks. In *ASE 2009, 24th IEEE/ACM International Conference on Automated Software Engineering, Auckland, New Zealand, November 16-20, 2009*, pages 397–408, 2009.

[16] R. Holmes and R. J. Walker. Supporting the investigation and planning of pragmatic reuse tasks. In *29th International Conference on Software Engineering (ICSE 2007), Minneapolis, MN, USA, May 20-26, 2007*, pages 447–457, 2007.

[17] I. Keivanloo, J. Rilling, and Y. Zou. Spotting working code examples. In *36th International Conference on Software Engineering, ICSE '14, Hyderabad, India - May 31 - June 07, 2014*, pages 664–675, 2014.

[18] M. Kim, V. Sazawal, D. Notkin, and G. C. Murphy. An empirical study of code clone genealogies. In *Proceedings of the 10th European Software Engineering Conference held jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2005, Lisbon, Portugal, September 5-9, 2005*, pages 187–196, 2005.

[19] Koders. http://code.openhub.net.

[20] Krugle. http://krugle.com.

[21] D. Mandelin, L. Xu, R. Bodík, and D. Kimelman. Jungloid mining: helping to navigate the API jungle. In *Proceedings of the ACM SIGPLAN 2005*

Conference on Programming Language Design and Implementation, Chicago, IL, USA, June 12-15, 2005, pages 48–61, 2005.

[22] A. Marcus and J. I. Maletic. Recovering documentation-to-source-code traceability links using latent semantic indexing. In *Proceedings of the 25th International Conference on Software Engineering, May 3-10, 2003, Portland, Oregon, USA*, pages 125–137, 2003.

[23] C. McMillan, M. Grechanik, D. Poshyvanyk, Q. Xie, and C. Fu. Portfolio: finding relevant functions and their usage. In *Proceedings of the 33rd International Conference on Software Engineering, ICSE 2011, Waikiki, Honolulu , HI, USA, May 21-28, 2011*, pages 111–120, 2011.

[24] N. Meng, M. Kim, and K. S. McKinley. LASE: locating and applying systematic edits by learning from examples. In *35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013*, pages 502–511, 2013.

[25] A. Mishne, S. Shoham, and E. Yahav. Typestate-based semantic code search over partial programs. In *Proceedings of the 27th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2012, part of SPLASH 2012, Tucson, AZ, USA, October 21-25, 2012*, pages 997–1016, 2012.

[26] L. Moreno, G. Bavota, M. D. Penta, R. Oliveto, and A. Marcus. How can I use this method? In *37th IEEE/ACM International Conference on Software Engineering, ICSE 2015, Florence, Italy, May 16-24, 2015, Volume 1*, pages 880–890, 2015.

[27] E. Moritz, M. L. Vásquez, D. Poshyvanyk, M. Grechanik, C. McMillan, and M. Gethers. Export: Detecting and visualizing API usages in large source code repositories. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering, ASE 2013, Silicon Valley, CA, USA, November 11-15, 2013*, pages 646–651, 2013.

[28] A. T. Nguyen, T. T. Nguyen, H. A. Nguyen, A. Tamrawi, H. V. Nguyen, J. M. Al-Kofahi, and T. N. Nguyen. Graph-based pattern-oriented, context-sensitive source code completion. In *34th International Conference on Software Engineering, ICSE 2012, June 2-9, 2012, Zurich, Switzerland*, pages 69–79, 2012.

[29] L. Ponzanelli, G. Bavota, M. D. Penta, R. Oliveto, and M. Lanza. Mining stackoverflow to turn the IDE into a self-confident programming prompter. In *11th Working Conference on Mining Software Repositories, MSR 2014, Proceedings, May 31 - June 1, 2014, Hyderabad, India*, pages 102–111, 2014.

[30] D. Poshyvanyk, M. Gethers, and A. Marcus. Concept location using formal concept analysis and information retrieval. *ACM Trans. Softw. Eng. Methodol.*, 21(4):23, 2012.

[31] S. Rastkar, G. C. Murphy, and A. W. J. Bradley. Generating natural language summaries for crosscutting source code concerns. In *IEEE 27th International Conference on Software Maintenance, ICSM 2011, Williamsburg, VA, USA, September 25-30, 2011*, pages 103–112, 2011.

[32] P. C. Rigby and M. P. Robillard. Discovering essential code elements in informal documentation. In *35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013*, pages 832–841, 2013.

[33] N. Sahavechaphan and K. T. Claypool. Xsnippet: mining for sample code. In *Proceedings of the 21th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2006, October 22-26, 2006, Portland, Oregon, USA*, pages 413–430, 2006.

[34] SearchCode. https://searchcode.com.

[35] R. W. Selby. Enabling reuse-based software development of large-scale systems. *IEEE Trans. Software Eng.*, 31(6):495–510, 2005.

[36] D. C. Shepherd, Z. P. Fry, E. Hill, L. L. Pollock, and K. Vijay-Shanker. Using natural language program analysis to locate and understand action-oriented concerns. In *Proceedings of the 6th International Conference on Aspect-Oriented Software Development, AOSD 2007, Vancouver, British Columbia, Canada, March 12-16, 2007*, pages 212–224, 2007.

[37] S. Thummalapenta and T. Xie. Parseweb: a programmer assistant for reusing open source code on the web. In *22nd IEEE/ACM International Conference on Automated Software Engineering (ASE 2007), November 5-9, 2007, Atlanta, Georgia, USA*, pages 204–213, 2007.

[38] M. Würsch, G. Ghezzi, G. Reif, and H. C. Gall. Supporting developers with natural language queries. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1, ICSE 2010, Cape Town, South Africa, 1-8 May 2010*, pages 165–174, 2010.

[39] H. Zhong, T. Xie, L. Zhang, J. Pei, and H. Mei. MAPO: mining and recommending API usage patterns. In *ECOOP 2009 - Object-Oriented Programming, 23rd European Conference, Genoa, Italy, July 6-10, 2009. Proceedings*, pages 318–343, 2009.

Result 7: PaparusCDTEditor

```
public class PapyrusCDTEditor extends CEditor {
 protected IAction gmfUndo, gmfRedo;
 protected IAction textUndo, textRedo;
 @Override
 public void createPartControl(Composite parent) {
  IActionBars actionBars = getEditorSite().getActionBars();
  if((actionBars != null) && (gmfUndo == null)) {
   gmfUndo =  actionBars.getGlobalActionHandler(ITextEditorActionConstants.UNDO);
    gmfRedo =  actionBars.getGlobalActionHandler(ITextEditorActionConstants.REDO);
  }
  super.createPartControl(parent);
  if(actionBars != null) {
   textUndo = actionBars.getGlobalActionHandler(ITextEditorActionConstants.UNDO);
   textRedo = actionBars.getGlobalActionHandler(ITextEditorActionConstants.REDO);
    actionBars.setGlobalActionHandler(ITextEditorActionConstants.UNDO, gmfUndo);
    actionBars.setGlobalActionHandler(ITextEditorActionConstants.REDO, gmfRedo);
   actionBars.updateActionBars();
  }
  @Override
  public ISourceViewer createSourceViewer(Composite parent, IVerticalRuler ruler, int styles) {
   final ISourceViewer viewer = super.createSourceViewer(parent, ruler, styles);
   focusListener = new FocusListener() {
   public void focusLost(FocusEvent e) {
    if(isDirty()) {
     syncCpp.syncCDTtoModel();
     Classifier classifier = (Classifier)papyrusTextInstance.getEditedObject();
     doSave(new NullProgressMonitor());
     SyncModelToCDT.syncModelToCDT(classifier);
    }
     IActionBars actionBars = getEditorSite().getActionBars();
     if(actionBars != null) {
      if((gmfUndo != null) && (gmfRedo != null)) {
       actionBars.setGlobalActionHandler(ITextEditorActionConstants.UNDO, gmfUndo);
       actionBars.setGlobalActionHandler(ITextEditorActionConstants.REDO, gmfRedo);
       actionBars.updateActionBars();
     } } }
   public void focusGained(FocusEvent e) {
    IActionBars actionBars = getEditorSite().getActionBars();
    if(actionBars != null) {
     if((textUndo != null) && (textRedo != null)) {
      actionBars.setGlobalActionHandler(ITextEditorActionConstants.UNDO, textUndo);
      actionBars.setGlobalActionHandler(ITextEditorActionConstants.REDO, textRedo);
      actionBars.updateActionBars();
     } } } };
  return viewer;
} }
```

Result No.9,10: AspectEditorContributor

```
public class AspectEditorContributor extends MultiPageEditorActionBarContributor {
 private IEditorPart activeEditorPart;
 private AspectEditor aspectEditor;
 @Override
 public void setActivePage(IEditorPart part) {
  if (activeEditorPart == part)   return;
  activeEditorPart = part;
  IActionBars actionBars = getActionBars();
  if (activeEditorPart != null && activeEditorPart instanceof ITextEditor) {
   IActionBars siteActionBars = ((IEditorSite)activeEditorPart.getEditorSite()).getActionBars();
    siteActionBars.setGlobalActionHandler(ITextEditorActionConstants.UNDO, getAction((ITextEditor)activeEditorPart, ITextEditorActionConstants.UNDO));
    siteActionBars.setGlobalActionHandler(ITextEditorActionConstants.REDO, getAction((ITextEditor)activeEditorPart, ITextEditorActionConstants.REDO));
   siteActionBars.updateActionBars();
  } else {
   if (part instanceof AspectEditor) {  aspectEditor = (AspectEditor) part; }
   IWorkbenchPartSite site = aspectEditor.getSite();
   if (site instanceof IEditorSite) {
    ITextEditor textEditor = aspectEditor.getMultipageEditor().getTextEditor();
    IActionBars siteActionBars = ((IEditorSite) site).getActionBars();
    siteActionBars.setGlobalActionHandler(ITextEditorActionConstants.UNDO, getAction(textEditor, ITextEditorActionConstants.UNDO));
    siteActionBars.setGlobalActionHandler(ITextEditorActionConstants.REDO, getAction(textEditor, ITextEditorActionConstants.REDO));
    siteActionBars.updateActionBars();
 } }}
 protected IAction getAction(ITextEditor editor, String actionID) {
   return (editor == null ? null : editor.getAction(actionID));
 } }
```

Figure 4: Results from SearchCode CSE

Result No 2,3.: AndroidTextEditor

```
public abstract class AndroidTextEditor extends FormEditor implements IResourceChangeListener {
 protected void createAndroidPages() {
  mIsCreatingPage = true;
  createFormPages();
  createTextEditor();
  createUndoRedoActions();
  postCreatePages();
  mIsCreatingPage = false;
 }
 private void createUndoRedoActions() {
  IActionBars bars = getEditorSite().getActionBars();
  if (bars != null) {
    IAction action = mTextEditor.getAction(ActionFactory.UNDO.getId());
    bars.setGlobalActionHandler(ActionFactory.UNDO.getId(), action);
    action = mTextEditor.getAction(ActionFactory.REDO.getId());
    bars.setGlobalActionHandler(ActionFactory.REDO.getId(), action);
    bars.updateActionBars();
  }}
 private void createTextEditor() {
   mTextEditor = new TextEditor();
   int index = addPage(mTextEditor, getEditorInput());
   mTextPageIndex = index;
   setPageText(index, mTextEditor.getTitle());
   IDocumentProvider provider = mTextEditor.getDocumentProvider();
   mDocument = provider.getDocument(getEditorInput());
   mDocument.addDocumentListener(new IDocumentListener() {
    public void documentChanged(DocumentEvent event) {
    onDocumentChanged(event);
 }); } }
```

Result 4: DocumentUndoManagerImpl

```
public class DocumentUndoManagerImpl implements DocumentUndoManager {
 private static class UndoableTextChange extends AbstractOperation {
 protected long fUndoModificationStamp = Document.UNKNOWN;
 protected long fRedoModificationStamp = Document.UNKNOWN;
 protected void reinitialize() {
   fUndoModificationStamp = Document.STAMP;
   fRedoModificationStamp = Document.STAMP;
  }
 protected void undoTextChange() {
  try {
     if (fDocumentUndoManager.fDocument instanceof Document)
         ((Document)fDocumentUndoManager.fDocument).replace(fStart, fText.length(), fPreservedText,fUndoModificationStamp);
     else
          fDocumentUndoManager.fDocument.replace(fStart, fText.length(), fPreservedText);
            } catch (BadLocationException x) {
   } }
 public boolean canUndo() {
   if (isValid()) {
    if (fDocumentUndoManager.fDocument instanceof Document) {
      long docStamp = ((Document)fDocumentUndoManager.fDocument).getModificationStamp();
      boolean canUndo = docStamp == Document.STAMP ||  docStamp >= getRedoModificationStamp();
      return canUndo;
  }
  return false;
}
public IStatus undo() {
  if (isValid()) {
    fDocumentUndoManager.fireDocumentUndo(fStart, fPreservedText, fText, null, DocumentUndoEvent.ABOUT_TO_UNDO, false);
    undoTextChange();
    fDocumentUndoManager.resetProcessChangeState();
    fDocumentUndoManager.fireDocumentUndo(fStart, fPreservedText, fText, null);
    return Status.OK_STATUS;
  }
 return IOperationHistory.OPERATION_INVALID_STATUS;
 }
 protected void redoTextChange() {
  try {
    if (fDocumentUndoManager.fDocument instanceof Document)
       ((Document)fDocumentUndoManager.fDocument).replace(fStart, fEnd - fStart, fText, fRedoModificationStamp);
   else
     fDocumentUndoManager.fDocument.replace(fStart, fEnd - fStart, fText);
   } catch (BadLocationException x) {
  }}
protected void updateTextChange() {
   fText = fDocumentUndoManager.fTextBuffer.toString();
   fDocumentUndoManager.fTextBuffer.setLength(0);
   fPreservedText = fDocumentUndoManager.fPreservedTextBuffer.toString();
   fDocumentUndoManager.fPreservedTextBuffer.setLength(0);
}
 protected UndoableTextChange createCurrent() {
   if (fDocumentUndoManager.fFoldingIntoCompoundChange)
     return new UndoableCompoundTextChange(fDocumentUndoManager);
  return new UndoableTextChange(fDocumentUndoManager);
 } }
```

Figure 5: Results from SearchCode CSE