Weekly Meeting 10/27/2015

Approach: Example Clustering Phase

1. PROBLEM STATEMENT

Problem: The code examples returned from code search engine or other code search tools are not structured.

Input: a set of code examples returned from code search engine.

Output: a list of code example clusters based on their structural similarity.

2. APPROACH

Without confined to an established code corpus, we use code search engine to extract code examples based on free-form queries from users. We build our tool in a plugin mechanism so that any code search engine can be easily planted into our system. We choose SearchCode [14] in our prototype because 1) it has good support for free-form queries, 2) it is an open source code search engine with over 7000 projects from Github, Bitbucket, Google Code, and Sourceforge, 3) it has complete API documentation.

2.1 Example Clustering Phase

Reformulate User Query. Most code search engines rely on text matching to extract code examples, thus vocabulary mismatching can affect their performance [5]. We therefore refine the given query using NL approaches before querying the code search engine. For each queried term, we perform stemming to remove derivational affixes using Porter stemmer [12] (e.g., parsing→parse). *and identify salient terms (verbs, nouns, and adjectives) using a POS tagger [16]. We choose these terms because prior work shows that they are salient elements in the query and other terms such as proposition, conjunction, and pronoun always impact query performance [5]. For example, the proposition* `in` *always matches the term* `plugin`*, leading to a false positive result).* To achieve better search result, we leverage WordNet to generate English synonyms (e.g., `inquire` and `query`) and software-specific word thesaurus [15, 19] to identify semantically-related terms (e.g., `source` and `target`). Our approach is similar to other *automatic query refinement* approaches, which have shown up to 41% improvement in precision and 30% increase in recall [4, 7]. Our preliminary study also shows that our query refinement approach works well to extract more related code examples. For instance, code search engine returns nothing for the query `inspect hard-drive for music files`, yet when we refine the query using the synonym of the term `inspect`—`check`, we successfully obtain 10 related code examples.

Locate Queried Features. Considering that source code examples are always entangled with multiple features, we separate program elements that are related to the queried features from big code examples. We do not cluster code examples based on the entire code example because it leads to too much noise in our preliminary study shown in Table 1. For instance, one of the code examples for the query '`undo redo TextEditor`' contains 10 inner classes with 1447 lines of code: `CopyAction, PasteAction, MoveAction,..., UndoAction, RedoAction`, and only the last two inner classes (147 lines) are related to the queried feature.

There exist a variety of feature location approaches based on IR approach, dynamic execution trace, and historical information. We adopt a plugin mechanism and select an IR approach similar to [8]. We extract type names, method calls, and identifier names, split them into tokens using Samurai identifier splitting approach [3], and convert each method into a document of Latent Semantic Index (LSI) space using an off-the-shelf tool call Semantic Vector [17]. *Existing corpus-based feature location tools always leverage comments or even informal documents for feature location, but we only use reliable information from code elements because web-based code search engine provides us a fruitful of code examples.* The user query is also converted to a document of LSI space. We measure the LSI similarity between user query and each method to identify query-related methods. We set a threshold for LSI similarity score to include all related methods for the given query and filter out unrelated elements.

Cluster Extracted Features. After extracting queried features from each code examples, we automatically cluster search results using Latent Dirichlet allocation (LDA). LDA is a generative model to express each document as a probability distribution of topics, and each topic is a probability distribution of words. Words can belong to multiple topics and documents can contain multiple topics. We use a popular topic modeling toolkit called MALLET [9] to train topic model. To find a near-optimal LDA configuration (the number of topic, Dirichlet prior on the per-topic word distribution, etc), we reimplement a simple Genetic-

Table 1: Preliminary Study for Example Clustering Phase

		Querying			Feature Location		Clustering
	Reformulated Query	# Examples	Time(sec)	Avg LOC	Avg Extr #M	Avg Extr LOC	Clusters
1	undo redo TextEditor	68	2.7	477	7	107	6
2	copy paste data clipboard	24	0.7	886	8	178	5
3	open url html browser	426	13.1	780	4	68	7
4	track mouse hover	33	1.0	770	10	177	4
5	open file external editor	341	13.4	927	11	192	7
6	batch workspace change single operation	11	0.6	928	11	192	3
7	remove problem marker from resource	211	7.6	664	12	206	8
8	highlight text range editor	111	4.4	293	12	219	6
9	update status line	445	24.6	921	4	102	7
10	prompt user select directory	114	4.2	893	6	129	5
11	use share image workbench	17	0.7	841	4	84	3
12	open dialog ask yes no question	30	1.0	382	6	98	4
13	parse source string ast node	323	11.8	371	6	97	8
14	extract return type method declaration	15	0.3	810	4	101	3
15	fill table background	457	16.2	678	5	97	8
16	platform debug trace	80	2.7	969	4	96	5
17	get display create current thread	390	27.2	778	5	77	6
18	run job ui thread	548	17.3	748	6	94	7
19	open external file	479	18.6	514	8	161	5
20	BMP image file parse	23	0.8	824	11	392	4
21	serialize objects XML	459	15.3	752	7	205	8
22	NMEA parser	17	0.3	775	15	277	3
23	line code counter	549	25.8	617	5	61	8
24	update status line	445	21.0	921	6	92	6
25	autosave	321	7.6	634	8	163	6
26	picture brightness	34	0.7	943	9	294	4

Time: represents the performance time from sending the first query to obtaining all identical results.
Extr #M: Extracted methods from feature location

Algorithm according to [11] for each query task. We set the maximum number of topics k as 9 following Miller's law in cognitive psychology [10]. We present the generated topics to users, so that user can select the example clusters they prefer. Hindle et al. conducted a case study in Microsoft to investigate if the topics extracted from requirements were relevant to development feature [6], and their result shows that experienced developers who are familiar with features are comfortable to identify behaviors based on topic words. We hypothesize that our users can also identify expected behavior based on topic words. User can choose to view more words in a topic to help them identify which topic they prefer. As shown in Table 2, if user prefers to implement a `java.swing.TextEditor` that supports undo/redo action, she may choose Topic 1 because it contains keyword `viewer, action, listener` which seems related; yet if she wants to implement a TextEditor for a Eclipse Plugin, Topic 7 is the most promising one because of the keyword `workbench, plugin`.

Table 2: Clustering Result

Query: '`undo redo TextEditor`'		
Topic	Topic Words	# E
1	viewer undo manager redo action listener provider	17
2	selection event undo change redo builder target	14
3	action factory editor undo redo constants text bars	23
4	editor text menu action site group page	18
5	document string outline file command set java	8
6	undo manager context operation element tab history	8
7	undo redo workbench listener shared plugin tool	6
Query: '`track mouse hover`'		
Topic	Topic Words	# E
1	mouse hover property handler event drag data	13
2	mouse component object key select item component	11
3	track mouse scene parent node local client	4
4	mouse bound height width series listener native	11

E: the number of Examples

2.2 Example Skeleton Generation Phase

We take three steps in this phase. For the selected code example cluster, we first extract type facts for the located methods that belong to the queried feature. For each example in each cluster, we identify main facts that are shared among all examples, and perform slicing to identify related methods as auxiliary feature. Finally, we rank the related elements and select the best-fit one to generate a reusable code example skeleton with common facts. Our tool also enables users to drag and drop other auxiliary features to generate desired code example.

Extract Structural Facts. Before we extract structural facts, we construct symbol table for partial program based on the partial program analysis [2]. We extract structural facts in an ontology instance. The ontology schema we use is the SEON Java ontology, which has been used in several prior works [13,18]. Every extracted fact is represented in the ontology instance by one or more `(subject, predicate, object)` triples. For instance, the fact that class c_1 extends class c_2 is represented by triple (`$c_2$, subType, $c_1$`). To manipulate the relationship between different facts, we use Resource Description Framework (RDF) to represent each `(subject, predicate, object)` triple as an edge from *subject* to *object* labeled with *predicate*. We use `Apache Jena`, which is a Java framework for linked data applications, to create and process ontology instances. We define `subType` as transitive predicate as (c_1, subType, c_2) and (c_2, subType, c_3) $\rightarrow$ (c_1, subType, c_3).

Extract Example Skeleton.

We greedily select the most suitable structural correspondence [1] and propagate the similarity of *object* nodes to the *subject* node.

The comparison is done by investigating the lexical simi-

Table 3: Structural Fact Types

Level	Type	Description
class level	(c_1, subType, c_2)	class c_1 extends c_2
	(c_1, innerclass, c_2)	class c_1 has an inner class c_2
	(c, field, v)	class c has field v
	(c, method, m)	class c contains method m
method level	(m, parameter, v)	method m has parameter v
	(m, return, t)	method m returns an object of type t
	(e_1, accesses, e_1)	$e_1 = e_2$, e_1 and e_2 are expressions
	(m_1, call, m_2)	method m_1 calls method m_2
	(m, variable, v)	method m has variable m_2
	(o, name, s)	The name of object o is s

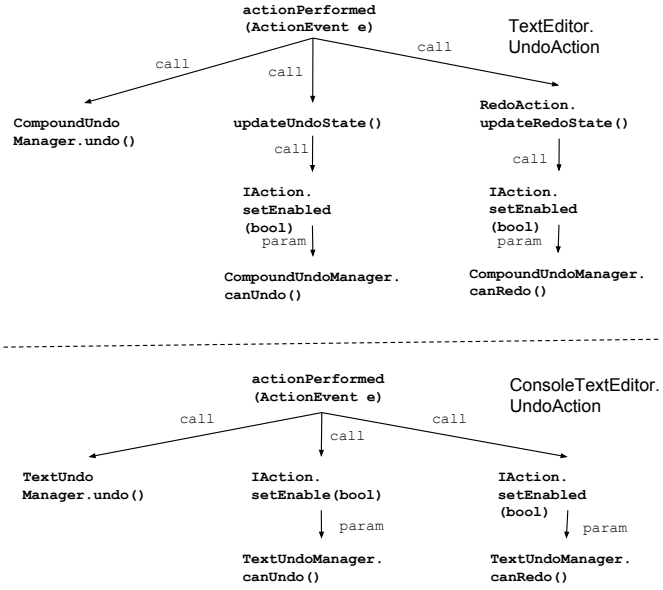

Figure 1: RDF graph

larity and similar paths in RDF graphs. For example, when comparing UndoAction.actionPerformed() in Figure 2 (A), we first try to match the fact `(actionPerformed, calls, CompoundUndoManager. undo())`. We only look at the facts with the same type. We find the fact `(actionPerformed, calls, TextUndoManager.undo())` which is lexically similar though not identical to the target fact. We calculate the similarity score as follows: 1) We split the name for each element in the tuple by camel-case splitter and separators such as '.' and '('. Thus `TextUndoManager.undo()` is split as `[Text, Undo, Manager, undo]`. 2) We calculate the distance for each element as the $editDistance(s_1, s_2)/Min\ (|s_1|, |s_2|)$, thus the similarity score between `TextUndoManager.undo()` and `CompoundUndoManager.undo()` is 0.25. 3) The distance between two tuples are the average distance for three elements. Thus the distance of two facts are 0.08. We select the most similar fact by minimizing the similarity distance and we identify `(actionPerformed, calls, CompoundUndoManager.undo())` as the structural correspondence of the fact `(actionPerformed, calls, TextUndoManager.undo())`.

We try to match the fact `(actionPerformed, calls, updateUndoState())`. Knowing that there is an edge from `updateUndoState()` to `setEnabled(undo.canUndo()` in RDF graph, we recursively match the fact `(updateUndoState, calls, setEnabled(undo.canUndo()))` with `(actionPerformed, calls, setEnabled(undo.canUndo()))`. The similarity score of *object* node `(updateUndoState, calls, setEnabled(undo.can Undo()))` is propagated to *subject* node `(actionPerformed, calls, updateUndoState())` as 0.4 compared to fact `(actionPerformed, calls, setEnabled(undo.canUndo()))` in class `UndoAction`. Our tool keeps on performing this matching until all facts in the target (Figure 2 (A)) is assigned a similarity score as 0 (i.e., fail to find correspondence) or a similarity score with other facts in the compared node.

We extract all matched *object* nodes (without outer edge) and use them as match seeds.

Rank Related Elements. We perform dependency analysis to extract example skeleton. An example skeleton should start from an element without any inner edge and end at an element without any outer edge which covers all match seeds. When selecting features, we use three rules to rank features in different examples: 1) the cost of reuse; 2) the number of resolved relationship; 3) the frequency of the feature in multiple examples; 4) the simplicity of code implementation. The cost of reuse is measured as the number of outer edges in RDF graph, i.e., the more outer edges in RDF graph, the more expensive to reuse because it has more dependencies. We have rule 2) because some of the relationships can not be resolved in partial program. When user tries to reuse the example, they have to manually resolve these external relationships.

For instance, in Figure 3 (D), the feature group B.1 (`mUndo Action = new LintEditAction(undoAction, getEditorDelegate().getEditor())`) is not selected because the class `LintEditAction` cannot be resolved based on this partial program. The first two rules ease the reusability of our suggested code example. The third rule makes sure that the feature is representative and the last rule improves the understandability of our results. The simplicity is measured as the length of the code. For instance, we prefer `actionBars.setGlobalActionHandler (ITextEditorActionConstants .UNDO, undo)` in Figure 4 (D) than `actionBars.setGlobal ActionHandler (ITextEditorActionConstants.UNDO, getAction((ITextEditor) part, ITextEditorActionConstants.UNDO))` because the former one is easier to interpret.

Enable User Interaction. Apart from the selected best-fit related program elements, our tool also lists a fuzzy set of suggested program elements that can be added into the example skeleton. Users can drag-and-drop their preferred features to the code skeleton and our tool is able to select related program elements associated with this feature to fill in the skeleton. For instance, in Figure 3 (D), if user wants to add `propertyChange()` methods into the code skeleton, our tool will add super interface `PropertyChangeListener` to both `UndoAction` and `RedoAction` based on RDF graph, which are tagged as `B.4` in Figure 3 (D).

3. USAGE SCENARIO

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 [13].

Query: undo redo TextEditor

Without knowing any APIs, developer first queries code search engine (CSE) with a free-form query '`undo redo`

Table 4: Search Result from Code Search Engine

No.	Name	Project	LOC	#M	#C	Involved objects and API calls
1	TextEditor	textmash	1271	12	1	UndoManager, UndoAction, RedoAction
4	ConsoleTextEditor	groovy-core	322	7	1	UndoManager, UndoAction, RedoAction
2	AndroidTextEditor	android-sdk	587	9	2	IActionBars.setGlobalActionHandler(), ActionFactory.UNDO, ActionFactory.REDO
8	LayoutCanvas	sdk	1721	3	2	IActionBars.setGlobalActionHandler(), ActionFactory.UNDO, ActionFactory.REDO
9	IUEditorContributor	buckminster	193	2	2	IActionBars.setGlobalActionHandler(), ActionFactory.UNDO, ActionFactory.REDO
5	PapyrusCDTEditor	papyrus	393	7	4	IActionBars.setGlobalActionHandler(), ITextEditorActionConstants.UNDO
6	AspectEditorContributor	eclipse-plugin	89	2	4	IActionBars.setGlobalActionHandler(), ITextEditorActionConstants.UNDO
10	WSDLContributor	eclipse	254	2	4	IActionBars.setGlobalActionHandler(), ITextEditorActionConstants.UNDO
3	DocumentUndoImpl	ide	1228	17	3	FileDocumentUndoManager
7	ATETextPane	antlrworks	650	7	5	AbstractUndoableEdit.undo(), AbstractUndoableEdit.redo(),

No. represents the rank from CSE. **#M** represents the number of methods that contain the query terms. **#C** represents the number of clusters the example belongs to.

`TextEditor'`. Table 4 shows top 10 results returned from SearchCode CSE.

Before using our tool, she has to manually investigate all code search examples to identify which API calls are frequently used. She selects the first result from CSE and uses keyword search to locate undo and redo feature in 6 methods and regards them as search seed. She started to explore the relationships between these methods. Shown in Figure 2 (A), 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 fifth example from CSE is similar to the first example. Shown in Figure 2 (B), This example seems similar because it uses an `TextUndoManager`, and implements `UndoAction` and `RedoAction`. She notice that the API `doc.addUndoable EditListener(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.

Our tool is able to represent code examples returned from CSE in a structured manner. We cluster the examples that share similar API calls and extract the queried features. Shown in Figure 2 (C), our tool extracts a skeleton of reuse code example for the queried feature. At the same time, we group auxiliary features based on program slicing and lexical similarity so that the user can select the features that she wants, as shown in Figure 2 (D).

4. REFERENCES

[1] 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.

[2] 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.

[3] E. Enslen, E. Hill, L. L. Pollock, and K. Vijay-Shanker. Mining source code to automatically split identifiers for software analysis. In *6th International Working Conference on Mining Software Repositories, MSR'09*, pages 71–80, 2009.

[4] S. Haiduc, G. Bavota, A. Marcus, R. Oliveto, A. D. Lucia, and T. Menzies. Automatic query reformulations for text retrieval in software engineering. In *35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013*, pages 842–851, 2013.

[5] E. Hill, M. Roldan-Vega, J. A. Fails, and G. Mallet. Nl-based query refinement and contextualized code search results: A user study. In *2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering, CSMR-WCRE 2014, Antwerp, Belgium, February 3-6, 2014*, pages 34–43, 2014.

[6] A. Hindle, C. Bird, T. Zimmermann, and N. Nagappan. Relating requirements to implementation via topic analysis: Do topics extracted from requirements make sense to managers and developers? In *28th IEEE International Conference on Software Maintenance, ICSM 2012, Trento, Italy, September 23-28, 2012*, pages 243–252, 2012.

[7] O. A. L. Lemos, A. C. de Paula, F. C. Zanichelli, and C. V. Lopes. Thesaurus-based automatic query expansion for interface-driven code search. In *11th Working Conference on Mining Software Repositories, MSR 2014, Proceedings, May 31 - June 1, 2014, Hyderabad, India*, pages 212–221, 2014.

[8] A. Marcus, A. Sergeyev, V. Rajlich, and J. I. Maletic. An information retrieval approach to concept location in source code. In *11th Working Conference on Reverse Engineering, WCRE 2004, Delft, The Netherlands, November 8-12, 2004*, pages 214–223, 2004.

[9] A. K. McCallum. Mallet: A machine learning for language toolkit. http://www.cs.umass.edu/ mccallum/mallet, 2002.

[10] G. Miller. The magical number seven, plus or minus two: Some limits on our capacity for processing information, 1956. One of the 100 most influential papers in cognitive science: http://cogsci.umn.edu/millennium/final.html.

[11] A. Panichella, B. Dit, R. Oliveto, M. D. Penta, D. Poshyvanyk, and A. D. Lucia. How to effectively use topic models for software engineering tasks? an approach based on genetic algorithms. In *35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013*, pages 522–531, 2013.

[12] M. F. Porter. An algorithm for suffix stripping. *Readings in information retrieval*, pages 313–316, 1997.

[13] 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.

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

[15] Y. Tian, D. Lo, and J. L. Lawall. Sewordsim: software-specific word similarity database. In *36th International Conference on Software Engineering, ICSE'14*, pages 568–571, 2014.

[16] K. Toutanova, D. Klein, C. D. Manning, and Y. Singer. Feature-rich part-of-speech tagging with a cyclic dependency network. In *HLT-NAACL*, 2003.

[17] D. Widdows and T. Cohen. The semantic vectors package: New algorithms and public tools for distributional semantics. In *Proceedings of the 4th IEEE International Conference on Semantic Computing (ICSC 2010), September 22-24, 2010, Carnegie Mellon University, Pittsburgh, PA, USA*, pages 9–15, 2010.

[18] 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.

[19] J. Yang and L. Tan. Inferring semantically related words from software context. In *9th IEEE Working Conference of Mining Software Repositories, MSR'12*, pages 161–170, 2012.

(A) Cluster 1: textmash:TextEditor

```
1. public class TextEditor extends JTextPane {
2.  public UndoAction undoAction = new UndoAction();
3.  public RedoAction redoAction = new RedoAction();
4.  public CompoundUndoManager undo;
5.  HashMap<Object, Action> actions = new HashMap<Object, Action>();
6.  public TextEditor(Workspace workspace) {
7.    this.workspace = workspace;
8.    undo = new CompoundUndoManager(workspace);
9.    actions.put("undo", undoAction);
10.   actions.put("redo", redoAction);
11.   }
12. public void discardUndoRedo() {
13.    undo.discardAllEdits();
14.    undoAction.updateUndoState();
15.    redoAction.updateRedoState();
16.   }
17. public class UndoAction extends AbstractAction {
18.   public UndoAction()
19.{     super("undo"); }
20.  public void actionPerformed(ActionEvent e) {
21.   try {
22.     undo.undo();
23.     updateUndoState();
24.     redoAction.updateRedoState();
25.    } catch (CannotUndoException ex) {
26.    }}
27. public void updateUndoState() {
28.     setEnabled(undo.canUndo());
29.   }}
30. public class RedoAction extends AbstractAction {
31.   public RedoAction()
32.{     super("redo"); }
33.   public void actionPerformed(ActionEvent e) {
34.    try {
35.      undo.redo();
36.      updateRedoState();
37.      undoAction.updateUndoState();
38.     } catch (CannotRedoException ex) {
39.     }}
40.  public void updateRedoState() {
41.      setEnabled(undo.canRedo());
42.     }}
```

(B) Cluster 1: groovy: ConsoleTextEditor

```
1. public class ConsoleTextEditor extends JScrollPane {
2.     private UndoAction undoAction = new UndoAction();
3.     private RedoAction redoAction = new RedoAction();
4.     private TextUndoManager undoManager;
5.    private TextEditor textEditor = new TextEditor();
6.    private Document doc = textEditor.getDocument();
7.    public ConsoleTextEditor () {
8.      this.undoManager = new TextUndoManager();
9.      doc.addUndoableEditListener(undoManager);
10.     undoManager.addPropertyChangeListener(undoAction);
11.     undoManager.addPropertyChangeListener(redoAction);
12.     doc.addDocumentListener(undoAction);
13.     doc.addDocumentListener(redoAction);
14.  }
15. private class RedoAction extends UpdateCaretListener
16.  implements PropertyChangeListener {
17.    public RedoAction()
18.{     setEnable(false); }
19.  public void actionPerformed(ActionEvent ae) {
20.   undoManager.redo();
21.setEnabled(undoManager.canRedo());
22.undoAction.setEnabled(undoManager.canUndo());
23.  }
24.    public void propertyChange(PropertyChangeEvent pce) {
25.     setEnabled(undoManager.canRedo());
26.   } }
27. private class UndoAction extends UpdateCaretListener
28. implements PropertyChangeListener {
29.   public RedoAction()
30.{     setEnable(false); }
31.   public void actionPerformed(ActionEvent ae) {
32.    undoManager.undo();
33.    setEnabled(undoManager.canUndo());
34.    redoAction.setEnabled(undoManager.canRedo());
35. }
36.  public void propertyChange(PropertyChangeEvent pce) {
37.    setEnabled(undoManager.canUndo());
38.  }
39.    private abstract class UpdateCaretListener extends AbstractAction
40.     implements DocumentListener {
41.       public void actionPerformed(ActionEvent ae) {}
42.     }}
```

(C) Cluster 1: suggested skeleton

```
public class TextEditor extends JComponent {
    private UndoAction undoAction = new UndoAction();
    private RedoAction redoAction = new RedoAction();
    private TextUndoManager undoManager;

   public TextEditor () {
     undoManager = new TextUndoManager();









  }
 private class UndoAction extends AbstractAction {


  public void actionPerformed(ActionEvent e) {
    undoManager.undo();
    setEnabled(undoManager.canUndo());
   redoAction.setEnabled(undoManager.canRedo());
 }


  }
  private class RedoAction extends AbstractAction {


  public void actionPerformed(ActionEvent e) {
   undoManager.redo();
   setEnabled(undoManager.canRedo());
   undoAction.setEnabled(undoManager.canUndo());
 }


   }  }
class TextUndoManager extends UndoManager() {}
```

(D) Suggested skeleton with alternative features

```
public class TextEditor extends JComponent {
    private UndoAction undoAction = new UndoAction();
    private RedoAction redoAction = new RedoAction();
    private TextUndoManager undoManager;
//@B.1,B2  private Document doc = textEditor.getDocument();
   public TextEditor () {
     undoManager = new TextUndoManager();
     //@B.1 doc.addUndoableEditListener(undoManager);
     //@B.2 doc.addDocumentListener(undoAction);
     //@B.2 doc.addDocumentListener(redoAction);
     //@B.4 doc.addPropertyChangeListener(undoAction);
     //@B.4 doc.addPropertyChangeListener(redoAction);
  }
 private class UndoAction extends AbstractAction {
 //@B.2 implements DocumentListener
 //@B.4 implements PropertyChangeListener
  public void actionPerformed(ActionEvent ae) {
    undoManager.undo();
    setEnabled(undoManager.canUndo());
    redoAction.setEnabled(undoManager.canRedo());
 }
//B.4  public void propertyChange(PropertyChangeEvent pce) {
//B.4    setEnabled(undoManager.canUndo());  }
}
  private class RedoAction extends AbstractAction {
 //@B.2 implements DocumentListener
 //@B.4 implements PropertyChangeListener
  public void actionPerformed(ActionEvent e) {
   undoManager.redo();
   setEnabled(undoManager.canRedo());
   undoAction.setEnabled(undoManager.canUndo());
  }
  //@B.4  public void propertyChange(PropertyChangeEvent pce) {
  //@B.4    setEnabled(undoManager.canUndo());  }
  }  }
  class TextUndoManager extends UndoManager() {}
```

Figure 2: Cluster 1: source code and expected results

(A) Cluster 2: android: AndroidTextEditor

```
1. public abstract class AndroidTextEditor extends FormEditor
2.    implements IResourceChangeListener {
3.    private TextEditor mTextEditor;
4.  public void createTextEditor() {
5.    mTextEditor = new TextEditor();
6.    IDocumentProvider provider = mTextEditor.getDocumentProvider();
7.    mDocument = provider.getDocument(getEditorInput());
8.    mDocument.addDocumentListener(new IDocumentListener() {
9.      public void documentChanged(DocumentEvent event) {
10.        onDocumentChanged(event);
11.     } });  }
12. private void createUndoRedoActions() {
13. IActionBars bars = getEditorSite().getActionBars();
14.  if (bars != null) {
15.  IAction action = mTextEditor.getAction(ActionFactory.UNDO.getId());
16.  bars.setGlobalActionHandler(ActionFactory.UNDO.getId(), action);
17.  action = mTextEditor.getAction(ActionFactory.REDO.getId());
18.  bars.setGlobalActionHandler(ActionFactory.REDO.getId(), action);
19.  bars.updateActionBars();
20.       } }    }
```

(B) Cluster 2: android-sdk: LayoutCanvas

```
1. public class LayoutCanvas extends Canvas {
2.    private IAction mUndoAction, mRedoAction;
3.    private LayoutEditorDelegate mEditorDelegate;;
4.   @Override
5.    public void updateGlobalActions(@NonNull IActionBars bars) {
6.    ITextEditor editor = mEditorDelegate.getEditor().getStructuredTextEditor();
7.      if (mUndoAction == null) {
8.           IAction undoAction = editor.getAction(ActionFactory.UNDO.getId());
9.          mUndoAction = new LintEditAction(undoAction,
10.           getEditorDelegate().getEditor());
11.    }
12.      bars.setGlobalActionHandler(ActionFactory.UNDO.getId(), mUndoAction);
13.    if (mRedoAction == null) {
14.         IAction redoAction = editor.getAction(ActionFactory.REDO.getId());
15.        mRedoAction = new LintEditAction(redoAction,
16.        getEditorDelegate().getEditor());
17.    }
18.     bars.setGlobalActionHandler(ActionFactory.REDO.getId(), mRedoAction);
19.     bars.updateActionBars();
20.   }  }
```

(C) Cluster 4: eclipse: PapyrusCDTEditor

```
1. public class PapyrusCDTEditor extends CEditor {
2.     protected IAction textUndo, textRedo;
3.
4.   @Override
5.   public void createPartControl(Composite parent) {
6.       IActionBars actionBars = getEditorSite().getActionBars();
7.      if (actionBars != null) {
8.          textUndo = actionBars.getGlobalActionHandler
9.            (ITextEditorActionConstants.UNDO);
10.         textRedo = actionBars.getGlobalActionHandler
11.     (ITextEditorActionConstants.REDO);
12.     actionBars.setGlobalActionHandler
13.       (ITextEditorActionConstants.UNDO, textUndo);
14.   actionBars.setGlobalActionHandler
15.     (ITextEditorActionConstants.REDO, textRedo);
16.   actionBars.updateActionBars();
17.     }
18.   }
```

(D) Cluster 4: eclipse: AspectEditorContributor

```
1. public class AspectEditorContributor extends MultiPageEditorActionBarContributor
2.  private IEditorPart activeEditorPart;
3.  @Override
4.  public void setActivePage(IEditorPart part) {
5.   activeEditorPart = part;
6.   IActionBars actionBars = getActionBars();
7.   if (activeEditorPart != null && activeEditorPart instanceof ITextEditor) {
8.     IActionBars siteActionBars = ((IEditorSite)activeEditorPart.
9.         getEditorSite()).getActionBars();
10.  siteActionBars.setGlobalActionHandler(ITextEditorActionConstants.UNDO,
11.    getAction((ITextEditor)activeEditorPart, ITextEditorActionConstants.UNDO));
12.  siteActionBars.setGlobalActionHandler(ITextEditorActionConstants.REDO,
13.    getAction((ITextEditor)activeEditorPart, ITextEditorActionConstants.REDO));
14.  siteActionBars.updateActionBars();
15.  } }
16. protected IAction getAction(ITextEditor editor, String actionID) {
17.   return (editor == null ? null : editor.getAction(actionID));
18.  }  }
```

(E) Suggested Skeleton for Cluster 2

```
1. public class TextEditor extends JComponent {
2.  public void createUndoRedoAction() {
3.   ITextEditor editor = new TextEditor();
4.   IActionBars actionBars = getEditorSite().getActionBars();
5.      if (actionBars != null) {
6.        IAction undoAction = editor.getAction
7.          (ActionFactory.UNDO.getId());
8.        actionBars.setGlobalActionHandler
9.          (ActionFactory.UNDO.getId(), undoAction);
10.       IAction redoAction = editor.getAction
11.         (ActionFactory.REDO.getId());
12.       actionBars.setGlobalActionHandler
13.          (ActionFactory.REDO.getId(), redoAction);
14.       actionBars.updateActionBars();
15.} }
```

(F) Suggested Skeleton for Cluster 4

```
1. public class TextEditor extends JComponent {
2.  public void createUndoRedoAction() {

3.   IActionBars actionBars = getEditorSite().getActionBars();
4.    IAction undo = actionBars.getGlobalActionHandler
5.      (ITextEditorActionConstants.UNDO);
6.    if (actionBars != null) {
7.      IAction redo = actionBars.getGlobalActionHandler
8.        (ITextEditorActionConstants.REDO);
9.     actionBars.setGlobalActionHandler
10.      (ITextEditorActionConstants.UNDO, undo)
11.    actionBars.setGlobalActionHandler
12.      (ITextEditorActionConstants.REDO, redo)
13.    actionBars.updateActionBars();
14.  } }
```

(G) Suggested Skeleton for Cluster 2, 4

```
1. public class TextEditor extends JComponent {
2.  public void createUndoRedoAction() {
3.    IActionBars actionBars = getEditorSite().getActionBars();
4.
5.    if (actionBars != null) {
6.      IAction undo = ...

7.      IAction redo = ...

8.     actionBars.setGlobalActionHandler(..., undo)

9.     actionBars.setGlobalActionHandler(..., redo)
10.   actionBars.updateActionBars();
11.  } }
```

(H) Alternative features for Suggested Skeleton

```
1.
2.
3.
4. //@B.1 ITextEditor editor = new TextEditor();
5.
6. //@A.1 IAction undo = actionBars.getGlobalActionHandler.(ITextEditorActionConsta
6. //@B.1 IAction undo = editor.getAction(ActionFactory.UNDO.getId());
7. //@A.1 IAction redo = actionBars.getGlobalActionHandler.(ITextEditorActionConsta
7. //@B.1 IAction redo = editor.getAction(ActionFactory.REDO.getId());
8. //@A.1 actionBars.setGlobalActionHandler(ITextEditorActionConstants.UNDO, undo)
8. //@A.1 actionBars.setGlobalActionHandler(ActionFactory.UNDO.getId(), undo);
9. //A.1   actionBars.setGlobalActionHandler(ITextEditorActionConstants.REDO, redo)
9. //A.1   actionBars.setGlobalActionHandler(ActionFactory.REDO.getId(), redo)
 } }
```

Figure 3: Cluster 2, 4: source code and suggested skeleton

(C) Cluster 2: suggested skeleton

```
public class TextEditor extends JComponent {

   public void createUndoRedoActions() {

     ITextEditor editor = new TextEditor();

     IActionBars actionBars = getEditorSite().getActionBars();
     if (bars != null) {

       IAction undoAction = editor.getAction
         (ActionFactory.UNDO.getId());

       actionBars.setGlobalActionHandler
         (ActionFactory.UNDO.getId(), undoAction);

       IAction redoAction = editor.getAction
         (ActionFactory.REDO.getId());

       bars.setGlobalActionHandler
          (ActionFactory.REDO.getId(), redoAction);

       bars.updateActionBars();
    } }
```

(D) Cluster 2: suggested skeleton with alternative features

```
public class TextEditor extends JComponent {
//@A.1 ITextEditor editor;
//@A.1 public void createTextEditor() {
//@A.1 editor = new TextEditor();
//@A.1 IDocumentProvider provider = mTextEditor.getDocumentProvider();
//@A.1 mDocument = provider.getDocument(getEditorInput());
//@A.1 mDocument.addDocumentListener(new IDocumentListener() {
//@A.1 public void documentChanged(DocumentEvent event) {onDocumentChanged(event);}
//@B.1 IAction mUndoAction = new LintEditAction(redoAction,
//@B.1   getEditorDelegate().getEditor());
//@B.1 IAction mRedoAction = new LintEditAction(redoAction,
//@B.1   getEditorDelegate().getEditor());
   public void createUndoRedoActions() {
//@B.2 (@NonNull IActionBars actionBars)
     ITextEditor editor = new TextEditor();
//@A.1 editor = new TextEditor();
//@B.3 ITextEditor editor = getStructuredTextEditor()
     IActionBars actionBars = getEditorSite().getActionBars(); //@deleteif B.2
     if (bars != null) {
//@B.1 if (mUndoAction == null) {
       IAction undoAction = editor.getAction(ActionFactory.UNDO.getId());
//@B.1 mUndoAction = new LintEditAction(undoAction,
//@B.1 getEditorDelegate().getEditor());
//@B.1 }
       actionBars.setGlobalActionHandler
          (ActionFactory.UNDO.getId(), undoAction);
 //@B.1 actionBars.setGlobalActionHandler
//@B.1   (ActionFactory.UNDO.getId(), mUndoAction);
//@B.1 if (mRedoAction == null) {
       IAction redoAction = editor.getAction
          (ActionFactory.REDO.getId());
//@B.1 mUndoAction = new LintEditAction(undoAction,
//@B.1  getEditorDelegate().getEditor());
//@B.1 }
       bars.setGlobalActionHandler
          (ActionFactory.REDO.getId(), redoAction);
 //@B.1 actionBars.setGlobalActionHandler
//@B.1   (ActionFactory.REDO.getId(), mRedoAction);
       bars.updateActionBars();
    } }
```

(C) Cluster 4: Suggested Skeleton

```
public class TextEditor extends JComponent {

 public void setActivePage(Composite parent) {

   IActionBars actionBars = getEditorSite().getActionBars();

   IAction undo = actionBars.getGlobalActionHandler
     (ITextEditorActionConstants.UNDO);

   IAction redo = actionBars.getGlobalActionHandler
     (ITextEditorActionConstants.REDO);

   actionBars.setGlobalActionHandler
      (ITextEditorActionConstants.UNDO, undo)
   actionBars.setGlobalActionHandler
      (ITextEditorActionConstants.REDO, redo)
} }
```

(D) Cluster 4: Suggested Skeleton with alternative features

```
public class TextEditor extends JComponent {

 public void setActivePage(Composite parent) {
//@B.1 (IEditorPart part)
   IActionBars actionBars = getEditorSite().getActionBars();
//@B.1 IActionBars actionBars = ((IEditorSite)activeEditorPart.
//@B.1  getEditorSite()).getActionBars();
   IAction undo = actionBars.getGlobalActionHandler
     (ITextEditorActionConstants.UNDO);
//@B.1 IAction undo = getAction((ITextEditor) part,
//@B.1   ITextEditorActionConstants.UNDO)
   IAction redo = actionBars.getGlobalActionHandler
     (ITextEditorActionConstants.REDO);
//@B.1 IAction undo = getAction((ITextEditor) part,
//@B.1    ITextEditorActionConstants.UNDO)
   actionBars.setGlobalActionHandler
     (ITextEditorActionConstants.UNDO, undo)
   actionBars.setGlobalActionHandler
     (ITextEditorActionConstants.REDO, redo)
} }
```

Figure 4: Cluster 2, 4: more alternative features