Weekly Meeting

1. PROBLEM STATEMENT

Developers frequently use source code examples as the basis for interacting with an application programming interface (API) to obtain the needed functionality. In this case, the source code example serves as the explicit origin for a reuse task when it fits a developer's context sufficiently well [4,13]. We define a reuse task as a series of copy-paste-modify actions to transform a piece of code to the target system, aiming to obtain desired functionalities by invoking a sequence of APIs with corresponding control structure. During the process of reuse, developers often query for a set of examples considering the task, and integrate examples that match the task into the target context [5]. The location and integration phase can be iterative as there is no guarantee that a selected example will be appropriate until the integration results are examined.

While many existing works focus on helping developers locate related source code examples [1,6,10,11,15,19], few approaches exist to support integration of source code [2]. The integration phase is not trivial: First, there exist multiple examples that provide desired features for the task. Recent study [5] shows that it is unlikely to find a 'perfect' example that provides all desired features for the reuse task, and the desired features are always scattered in different example variants. Thus even with the help of code recommendation tools, developers still need to manually investigate multiple recommended code examples that are found based on the relevance to the user query rather than the ability to fit for the target context, in order to find the common API calls that provide expected features. Second, for any selected example in non-trivial system, the number of structural dependencies to follow is much too large to be completely covered by a developers [7]. As a result, developers have to rely on their intuition to determine which elements should be integrated. Finally, when users try to modify the selected example to the target context, they should not only measure structural and semantic similarity between the example and target context for the main API calls that provide desired features [2], but also determine which related elements should be transformed to the target, and eliminate unrelated code to save maintenance cost.

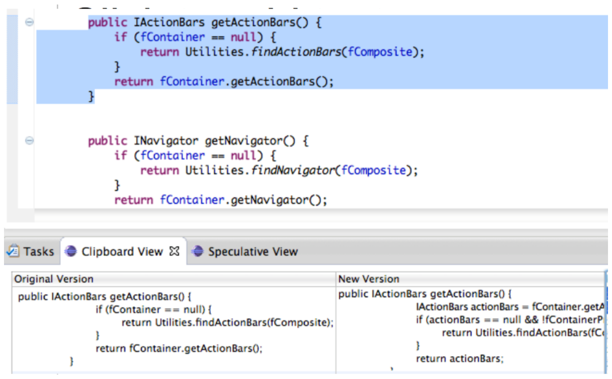
To overcome these challenges, we propose to automatically identify common API calls across different source code examples as well as common coupling elements that interact with common functionalities in a common way. We define coupling elements as ones that are data dependent and control dependent on the common functionalities, callers/callees of these functionalities [14], and sibling elements that share the same callers/callees [16]. We assume that *common API calls are the major features that user intends to reuse based on the user query and the coupling elements of these functionalities should also be investigated if they interact with these APIs in a common way.* Given a set of examples for a desired reuse task, our tool returns a reuse task suggestion in the format of a sequence of API calls and corresponding control structure which have been transformed to fit for the target context [2]. Our tool also lists a set of variants from multiple source code examples for users to select the features they prefer to integrate into the target context, and automatically transforms chosen examples to the target context.

Although some approaches advocate refactoring code rather than reuse code [3], recent researches have found that these kind of 'clone' cannot be easily refactored [9] and have to be modified to meeting requirements in new context [17]. Gilligan [8] and Procrustes [7] try to address the problem of source code integration in the context of large-scale reuse tasks by suggesting program elements that are easy to reuse based on structural relevance and cost of reuse in the source context, but developers still have to manually go through related elements for each example and modify them to integrate into the desired context. Jigsaw [2] supports small-scale integration of source code into target system between the example and target context. By considering structural and semantic similarity measures, it greedily matches each element between two contexts 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 reused code to the target. Our approach overcomes these two limitations: we extract common functionalities from multiple examples and identify how the related elements interact with main API calls in a common way.

2. EXAMPLE

Drag-and-Drop support for systematic editing:

Consider a reuse task to implement a drag-and-drop plu-



```
public IActionBars getActionBars() {
    if (fContainer == null) {
        return Utilities.findActionBars(fComposite);
    }
    return fContainer.getActionBars();
}


public INavigator getNavigator() {
    if (fContainer == null) {
        return Utilities.findNavigator(fComposite);
    }
    return fContainer.getNavigator();
```

Original Version

```
public IActionBars getActionBars() {
        if (fContainer == null) {
            return Utilities.findActionBars(fComposite);
        }
        return fContainer.getActionBars();
    }
```

New Version

```
public IActionBars getActionBars() {
        IActionBars actionBars = fContainer.getA
        if (actionBars == null && !fContainerP
            return Utilities.findActionBars(fC
        }
        return actionBars;
```

Figure 1: Drag-and-drop example for systematic editing

(A) Drag-and-Drop View in the Table Format

```
public class TableViewDisplay extends Display {
@override
 public void run(Composite parent) {
  TableViewer table = new Table(parent, SWT.BORDER |
   SWT.H_SCROLL | SWT.V_SCROLL);
  GridLayout layout = new GridLayout();
  layout.numColumns = 2;
  layout.makeColumnsEqualWidth = true;
  table.setLayout(layout);
  Transfer[] types = new Transfer[] {TextTransfer.getInstance()};
  int dnd = DND.DROP_MOVE | DND.DROP_COPY;
  table.setTransfer(types);
  table.setOperation(dnd);
  table.addDragSupport(new DragListener());
  table.addDropSupport(new DropListener(table));
 } }
```

(B) DropListener that is related to the main functionality

```
public class DropListener implements DropTargetListener {
  private TableViewer table;
  public DropListener(TableView table)
     { this.table = table; }
 @Override
  public void drop(DropTargetEvent event) {
   TableItem item = (TableItem) event.data;
    table.add(item);
  } }
```

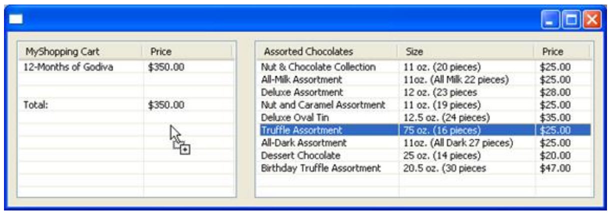
(C) DragListener that is related to the main functionality

```
public class DragListener implements DragSourceListener {
  @Override
 public void dragSetData(DragSourceEvent event) {
  TableItem item = (TableItem) event.getSelection();
  event.data = item.toString();
 }
```

Figure 3: Drag-and-Drop Example for shopping cart

gin called Clipboard for Eclipse to support systematic editing [12]. As shown in Figure 1, users drag and drop the method they want to edit to Clipboard before editing. When they finish changing the method, they take another snapshot using drag-and-drop. Clipboard will automatically recommend locations that are similar to the given example, and recommend similar but not identical edits based on similar context.

To complete this task, developer first queries Google code search (GCS) using the query `Eclipse plugin drag and drop to table`. We choose GCS because existing code example recommendation tools like Strathcona [6] and code



MyShopping Cart	Price
12-Months of Godiva	$350.00
Total:	$350.00

Assorted Chocolates	Size	Price
Nut & Chocolate Collection	11 oz. (20 pieces)	$25.00
All-Milk Assortment	11oz. (All Milk 22 pieces)	$25.00
Deluxe Assortment	12 oz. (23 pieces	$28.00
Nut and Caramel Assortment	11 oz. (19 pieces)	$25.00
Deluxe Oval Tin	12.5 oz. (24 pieces)	$35.00
Truffle Assortment	75 oz. (16 pieces)	$25.00
All-Dark Assortment	11oz. (All Dark 27 pieces)	$25.00
Dessert Chocolate	25 oz. (14 pieces)	$20.00
Birthday Truffle Assortment	20.5 oz. (30 pieces	$47.00

Figure 2: Example for Drag-and-drop

(A) Drag-and-Drop View in Eclipse Viewer

```
public class ListViewPart extends ViewPart {
 @Override
 public void createPartControl(Composite parent) {
  ListViewer viewer = new ListViewer(parent,
     SWT.H_SCROLL|SWT.V_SCROLL);
  int ops = DND.DROP_COPY | DND.DROP_MOVE;
  Transfer[] transT = new Transfer[]{TextTransfer.getInstance()};
  viewer.setTransfer(transT);
  viewer.setOperation(ops);
  viewer.addDragSupport(new DragListener());
  viewer.addDropSupport(new DropListener(viewer));
  viewer.setContentProvider(new TodoModelProvider());
 } }
```

(B) DropListener that is related to the main functionality

```
public class DropListener implements DropTargetListener {
 private ListViewer viewer;
 public DragListener()
 { this.viewer = viewer; }
 @Override
  public void drop(DropTargetEvent event) {
    ISelection sel=(ISelection) event.data;
    viewer.add(sel.toString());
  } }
```

(C) DragListener that is related to the main functionality

```
public class DragListener implements DragSourceListener {
  @Override
 public void dragSetData(DragSourceEvent event) {
  ISelection sel = event.getSelection();
  event.data = sel.toString();
 }
```

(D) Structured Data Provider for Drag-and-Drop

```
public class TodoModelProvider implements IStructuredContentProvider {
 @override
  public Object[] getElements(Object inputElement) {
  List<Todo> list = (List<Todo>) inputElement;
  return list.toArray();
 }
```

Figure 4: Drag-and-Drop Example for TODO labels

query tools like SNIFF [1] do not fit for such a medium scale reuse task. GCS returns 102 examples. We only analyze the first five returned examples because empirical evidence indicates that developers rarely look beyond this limit when searching [18]. The fourth example is much too long (over 1000 lines) without any explanation and the third and fifth example are duplicated with the first one. Figure 2 illustrates a snapshot for the first example returned by GCS. It implements a SWT shopping cart application to select items from all grocery items and put them into 'My shopping cart' via drag-and-drop (Figure 3). The second example implements an example for TODO labels which enables user to drag a TODO label from Editor Panel and drop it in an Eclipse Plugin List View (Figure 4). This is very similar to the task that requires to drag the source code from Editor Panel to an Eclipse Plugin View, yet we hope to use the table view to record multiple examples for both old version and new version, which is the format used in the first example.

Our next step is to extract common API calls for drag-and-drop action. It is always hard to distinguish the main functionality and variants via a single example, but with the help of multiple examples, our tool is able to identify the main API calls that provide desired functionality. In this example, we notice that the main API calls will be {new Viewer(), setTransfer(), setOperation(), addDragSupport(), addDropSupport()} given the hierarchical fact that both `TableViewer` and `ListViewer` are subclasses of `Viewer`.

After identifying the main API calls (and corresponding control structure), we investigate related elements for these API calls. The implementation of {DropTargetListener, DragSourceListener} are identified as common coupling elements, while {TodoModelProvider, TableItem, GridLayout} are excluded as they do not have corresponding mapping in the other example. We notice that we should override {DropTargetListener.drop(), DragSourceListener. dragSetData()} to help us finish the drag-and-drop reuse task.

3. REFERENCES

[1] 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.

[2] 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.

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

[4] W. B. Frakes and K. Kang. Software reuse research: Status and future. *IEEE Trans. Software Eng.*, 31(7):529–536, 2005.

[5] R. Holmes, R. Cottrell, R. J. Walker, and J. Denzinger. The end-to-end use of source code examples: An exploratory study. In *25th IEEE International Conference on Software Maintenance (ICSM 2009), September 20-26, 2009, Edmonton, Alberta, Canada*, pages 555–558, 2009.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] N. Meng, M. Kim, and K. S. McKinley. Systematic editing: generating program transformations from an example. In *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2011, San Jose, CA, USA, June 4-8, 2011*, pages 329–342, 2011.

[13] J. Parsons and C. Saunders. Cognitive heuristics in software engineering: Applying and extending anchoring and adjustment to artifact reuse. *IEEE Trans. Software Eng.*, 30(12):873–888, 2004.

[14] M. P. Robillard. Automatic generation of suggestions for program investigation. 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 11–20, 2005.

[15] 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.

[16] Z. M. Saul, V. Filkov, P. T. Devanbu, and C. Bird. Recommending random walks. 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 15–24, 2007.

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

[18] J. Starke, C. Luce, and J. Sillito. Searching and skimming: An exploratory study. In *25th IEEE*

International Conference on Software Maintenance (ICSM 2009), September 20-26, 2009, Edmonton, Alberta, Canada, pages 157–166, 2009.

[19] 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.