Weekly Meeting 9/29/2015

Discussion on Related Work, and Input/Expected Output

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 [7,24]. 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 [9]. 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 [3,10,16,17,29,33], few approaches exist to support integration of source code [5]. The integration phase is not trivial: First, there exist multiple examples that provide desired features for the task. Recent study [9] 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 [11]. 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 [5], 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 [27], and sibling elements that share the same callers/callees [30]. 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 [5]. 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.

2. RELATED WORK

Search for code example. Recent study shows that developers frequently search for code and the most common question deals with finding examples or sample code [28]. Our example clustering approach is similar to some prior works that extract representative examples for specific APIs or user query. MAPO [36] and its successor [35] leverage frequent call sequences to cluster the usage of specific APIs and rank abstract usage patterns based on the context similarity. Buse et al [2] propose to generate documented abstract API usages by extracting and synthesizing code examples for a particular API data type. They identify the instantiations of the given data types and the statements relevant to it, cluster these examples, and generate an abstract example for each cluster. Different from these works that generate abstract usage patterns for specific API or data type, SNIFF [3] performs type-based intersection of code chunks that contain keywords from free-form query to cluster the most relevant and common part of the code chunks, and uses textural relevance to rank concrete code examples. These works only focus on providing code examples based on the popularity or textural relevance thus developers have to manually resolve structural dependencies before reusing the examples.

To overcome this limitation, some techniques have been proposed to recommend code examples based on context or structural dependency. Strathcona [10] automatically gener-

Table 1: Comparison of code search tools, code reuse tools, and code transformation tools

Tool	Source	I	O_m	Related	Cluster	Transfer
MAPO [36]	C	M			✓	
Buse et al. [2]	C	M			✓	
SNIFF [3]	C	Q			✓	
Strathcona [10]	C	C		✓		
Prompter [25]	W	C		✓		
Portfolio [18]	W	Q		✓		
Export [23]	C	M		✓		
Sourcerer [1]	C	Q		✓		
CodeGenie [15]	C	Q		✓		
PRIME [21]	C	Q		✓		
Prospector [16]	C	T		✓		
PRIME [21]	C	Q		✓		
MUSE [22]	C	M		✓	✓	
Keivanloo [13]	C	Q		✓	✓	
Gilligan [12]	U	E	✓	✓		
Procrustes [11]	U	E	✓	✓		
Jigsaw [5]	U	E		✓		✓
LASE [20]	U	E		✓	✓	✓
Ours	W	Q	✓	✓	✓	✓

The **Source** column specifies the input source (**C**orpus, **W**eb, **U**ser); The **I** column specifies the input format (**M**ethod, **C**ontext, free-form **Q**uery, **T**ype, **E**xample); The O_m column specifies if the output is across the method boundary; The **Related** column specifies if the tool considers related methods when recommending example or performing transformation; The **Cluster** column specifies if the output results are clustered to reduce redundancy; The **Transfer** column specifies if the tool is able to transfer the code to another context.

ate queries from code context, and recommends code snippet through six heuristics based on inheritance links, method calls, and used types. Prompter [25] builds on a similar idea to match the generated query with Stack Overflow entries, to automatically push discussions relevant to the developers' task at hand. Other tools, such as Portfolio [18] and ExPort [23], consider related methods based on structural dependency. Portfolio [18] takes a textural query as input, and generate a list of suggested functions based on call graph and textural similarity. ExPort [23] uses call graph to generate Rational Topic Model, thus linked methods have a high probability of being associates to the same topics. It further visualizes the structural dependency for browsing code examples. Sourcerer [1] uses relational models to represent the structural information in code and records both lexical and structural relationships to support structure-based search. CodeGenie [15] leverages the tokens used in test cases to automatically generate code search query and query Sourcerer for desired functionality. Prospector [16] and a series of its variants either synthesize type instantiation [29, 34] or recommend call chain [8], which are generally not suitable at the level of code snippet suggestion with complicated API related fingerprints. Other tools that support semantic code search require users to provide formal specification [26] or partial-code query [21], which are not always available for a medium-scale reuse task. Although these approaches consider structural dependency to generate a concrete code example, they usually return a number of examples and user has to identify the most representative usages manually.

MUSE [22] and Keivanloo et al [13] addresses this limitation by grouping similar usage examples together and measuring syntactically correctness. MUSE [22] uses slicing to generate concrete usage examples and selects the most representative ones based on the popularity and readability. But it can only extract examples from compilable code while most task-based reuse examples are partial code that is most likely non-executable, and often cannot even be compiled using a standard compiler. Apart from mining API usage like MUSE, Keivanloo et al [13] uses clone detection to mine examples involving loops and conditions such as bubble sort problem, measures syntactically and semantically correctness for loops and conditions, and abstracts code patterns (e.g., '##.#()') to rank the most popular working example. Both tools only suggest implementation within a single method, which make it not suitable for task-based code reuse described below.

Code Reuse. Although some approaches advocate refactoring code rather than reuse code [6], recent researches have found that these kind of 'clone' cannot be easily refactored [14] and have to be modified to meeting requirements in new context [31]. Gilligan [12] and Procrustes [11] 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, there is no guarantee that a selected example will be appropriate until the integration results are examined [9]. We find that one example is not sufficient to create a pragmatic reuse plan as the main functionality is always interleaving with auxiliary functionalities. Even with the help of current tools that support code reuse, developers still need to manually investigate related elements in each example, identify structural correspondence in different context, and distinguish common functionality and specific ones that should be modified in integration phase. Our approach provides partial abstraction from multiple examples, which might reduce false positives and negatives. By distinguishing main functionality and auxiliary functionality, we provide a main framework for reuse task and a set of variants for auxiliary functionalities, so that user can select which features are necessary based on their needs.

Code Transformation Based on Similar Contexts. Jigsaw [5] supports small-scale integration of source code into target system between the example and target context. Based on its ancestor [4] that identifies structural correspondence based on clone detection technique, 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 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. LASE [20] applies similar but not identical changes to multiple code locations based on context similarity. Based on two or more examples provided by users, LASE extracts common edit script shared by all these examples, identifies common edit context via structural dependency, and transforms the edit script to a new context accordingly. 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.

3. EXAMPLE

Drag-and-Drop support 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

Consider a reuse task to implement a drag-and-drop plugin called Clipboard for Eclipse to support systematic editing [19]. 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 [10] and code query tools like SNIFF [3] 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 [32]. 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'

(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

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.

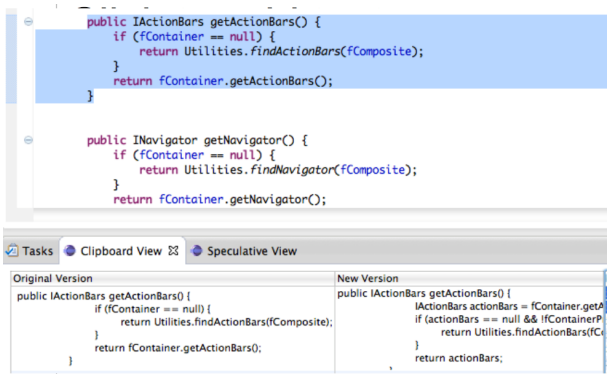

Figure 1: Drag-and-drop example for systematic editing

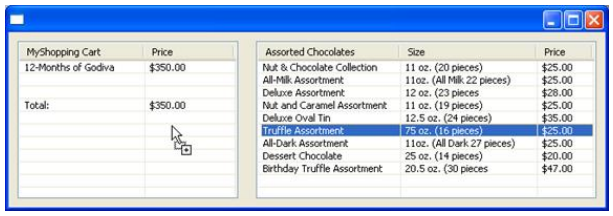

Figure 2: Example for Drag-and-drop

4. REFERENCES

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

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

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

[8] J. Galenson, P. Reames, R. Bodík, B. Hartmann, and K. Sen. Codehint: dynamic and interactive synthesis of code snippets. In *36th International Conference on Software Engineering, ICSE '14, Hyderabad, India - May 31 - June 07, 2014*, pages 653–663, 2014.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] O. A. L. Lemos, S. K. Bajracharya, J. Ossher, P. C. Masiero, and C. V. Lopes. Applying test-driven code search to the reuse of auxiliary functionality. In *Proceedings of the 2009 ACM Symposium on Applied Computing (SAC), Honolulu, Hawaii, USA, March 9-12, 2009*, pages 476–482, 2009.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] S. P. Reiss. Semantics-based code search. In *31st International Conference on Software Engineering, ICSE 2009, May 16-24, 2009, Vancouver, Canada, Proceedings*, pages 243–253, 2009.

[27] 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.

[28] C. Sadowski, K. T. Stolee, and S. G. Elbaum. How developers search for code: a case study. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, Bergamo, Italy, August 30 - September 4, 2015*, pages 191–201, 2015.

[29] 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.

[30] 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.

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

[32] 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.

[33] 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.

[34] 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.

[35] J. Wang, Y. Dang, H. Zhang, K. Chen, T. Xie, and D. Zhang. Mining succinct and high-coverage API usage patterns from source code. In *Proceedings of the 10th Working Conference on Mining Software Repositories, MSR '13, San Francisco, CA, USA, May 18-19, 2013*, pages 319–328, 2013.

[36] 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.