CoDocent: Support API Usage with Code Example and API Documentation

Ye-Chi Wu∗ Lee Wei Mar∗ Hewijin Christine Jiau†

∗Institute of Computer and Communication Engineering
†Department of Electrical Engineering
National Cheng Kung University, Tainan, Taiwan
{yechiwu, lwmar}@nature.ee.ncku.edu.tw, jiauhjc@mail.ncku.edu.tw

Abstract—API documentation and code example are two major resources to support API usage. To find the best way to use APIs within specific programming tasks, an effective strategy to link related APIs becomes critical. Currently, many code search engines have been proposed to solve this issue. Through those search results, programmers must manually traverse across all API documents to learn the referred API calls. To ensure the productivity in the style of programming with APIs, this work provides CoDocent to help programmers review code examples found by search engines. For each found code example, CoDocent can automatically link related API documents to provide diagrams as abstractions to reflect the semantics of API calls. Two evaluations are conducted to show the effectiveness of CoDocent in investigating and adapting API calls from code examples.

Keywords—programming with APIs; code example; API documentation.

I. INTRODUCTION

To enable effective software reuse, one proven method is to build software applications with framework. Through interacting with application programming interface (API), programming effort can be reduced. To ensure the benefit, API usability assurance is critical to ensure development productivity [1][2]. Linking related API documents together with investigating code example is an effective strategy to learn the best use of APIs for specific programming tasks [3].

To effectively support API usage, many code search engines [4][5][6][7] have been proposed to find code examples for programming with APIs. The search results can reduce programming effort in locating code examples as reference candidates for supporting API usage. During reviewing the code examples, the effort of investigating related API documents to learn the usage of the referred API calls is inevitable. To review a referred API call must go through several related API documents. Through such investigation, programmers can realize the semantics of the API call, which is the API usage context and protocols specified in the framework. However, current API documentation organized with general guidance is not for any specific usage. Programmers must manually traverse all API documents to confirm the usability. This is time-consuming and error-prone process for correctly programming with APIs.

To ensure the programming productivity, this work provides a code example docent tool, CoDocent, to help programmers review code examples found by various code search engines. For each found code example, CoDocent automatically links related APIs by framework structure and the relationship between API documents. CoDocent provides diagrams as abstractions to reflect the semantics of API calls referred in the code example. The diagrams also provide guidance for programmers to investigate those related API documents. With CoDocent, programmers can confirm the semantics of API calls and adapt the API calls in the new programming task.

The rest of the paper is organized as follows: Section II provides a motivating example, while related work is presented in Section III. Section IV then introduces CoDocent and the interpretation of its diagrams provided, and the results of evaluation are discussed in Section V. Finally, Section VI draws the conclusion of this work.

II. A MOTIVATING EXAMPLE

Figure 1 shows a flow that a programmer usually follows in programming with APIs style. A motivating example is demonstrated here for code example investigation. In this example, a programmer uses SWT framework in his programming task, which is to arrange the SWT widgets with grid style. The programmer searches API usage support among official code examples provided by SWT developers [8]. The detail in code example investigation (step 3 and 4 in Figure 1) is further explained as follows.

Check the Code Example Outline. For each selected code example, the programmer needs an outline to evaluate the relevance of the code example in his programming task. Due to the lack of tool support, the programmer has to construct the outline via browsing through the source code of the code example. For example, he go through a code example Snippet335. As the code fragment shown in Figure 2, the programmer finds that classes GridLayout and GridData are used in Snippet335. He notices that the naming of the used classes seems to be highly related to layout with grid style. He considers that Snippet335 is a relevant code example to expose API usage for his...
programming task and keeps investigating it for further details.

```
public class Snippet335 {
  public static void main(String[] args) {
    Display display = new Display();
    Shell shell = new Shell(display);
    shell.setLayout(new GridLayout());
    Label label = new Label(shell, SWT.WRAP | SWT.BORDER);
    GridData dt = new GridData(SWT.FILL, SWT.FILL, true, true);
    dt.widthHint = 10;
    label.setLayoutData(dt);
    .......
  }
}
```

Figure 2. A code fragment in a SWT example.

**Investigate API Usage in Code Example.** During detailed code example investigation, the programmer checks the use of API calls. To confirm the semantics of specific API call, he must traverse through all API documents to organize the usage of a specific API call. As the line 5 in Figure 2, a newly created GridLayout object is set to a Shell widget via invoking setLayout() method. To realize the usage context and protocols of this API call, he traverses needed information from API documents as follows:

1) He checks the API document of class Shell. According to API document of class Shell, the programmer notices that setLayout() method is inherited from Composite. The Shell widget is also an instance of Composite for composing different SWT widgets. With the assigned layout style, the SWT widgets inside the Composite will be arranged accordingly.

2) He also investigates the API document of GridLayout. The GridLayout API document, via See Also, refers that grid layout concept in SWT framework is implemented with classes GridLayout and GridData. By reading the document content of GridLayout and GridData, the programmer learns that GridData object is utilized to configure a widget inside a Composite about how to be layout with GridLayout style.

3) After confirming the usage context of GridLayout, the programmer further checks available usage protocols about GridLayout object instantiation. According to GridLayout API document, there is one alternative constructor can be replaced the instantiation demonstrated in the line 5.

From line 6, he learns that a Label widget is created, assigned as a component of the Shell widget, and arranged with GridLayout style. Line 7 to 9 present GridData setting for Label widget. Line 7 and 8 show that the GridData object can be configured through parameters of constructor and assignment to corresponding fields. The code example only demonstrates partial available configurations in the GridData object. To gain more information about such configurations, the programmer further reviews through the API documents of GridData's methods and fields:

- The first two parameters of the referred GridData constructor are used to configure how control will be positioned horizontally and vertically within a grid cell. The SWT.FILL constant can be altered by different SWT constants such as SWT.BEGINNING, SWT.CENTER or SWT.END.
- There are two alternative constructors which can be used to initialize a GridData object.
- There are several non-static public fields in the GridData, such as horizontalSpan and verticalSpan, which can be used to configure GridData object state.

In this example, the API calls referred in the code example give programmers the usage information. However, API usage with code examples is still laborious for programmers. First, during choosing code examples process for further investigation, there is no support in providing a code example outline for programmers to effectively evaluate the relevance of the selected one to his programming task. Programmers have to construct such outline by themselves via browsing the whole source code. Second, a code example only demonstrates API usage in one specific context. To adapt the API into programming context, programmers must traverse across API documents for needed information. In summary, proper tool support is needed to reduce the programmers effort in constructing code example outline and traversing related API documents during code example investigation.

### III. Related Work

Code example is a widely used resource to support API usage in software development. Several code search engines, such as Google Code Search [9] and Koders [10], provide general code retrieval via user-defined keywords. To further enhance the relevance of search results, some works
have been proposed to find code examples with structural context of a developing program [5] or to extract API usage patterns/paths [4][6][7]. They all emphasize on locating a set of programming reference candidates to programmers.

Based on code examples retrieved by code search engines, Holmes et. al. [11] and Poshvyvanyk et. al. [12] proposed further overall support in programming with APIs. They both refer to two needs, which are still lack of tool support, for programmers: (1) programmers need help in selecting relevant ones among the code examples; and (2) programmers need assistance in accomplishing their programming task based on the selected code examples.

To help programmers confirm usage of API calls in source code, Assieme [13] and some popular IDEs like Eclipse can link a selected API call to corresponding API document. The eMoose [14] further highlights the usage directives in the API document of the selected API call. But these work links the API document of the selected API call only. To confirm the semantics of the API call, programmers must traverse related APIs to find the API call manually.

Traversing related APIs by framework structure is an effective method when programmers have decided to use an API during programming. For example, Long et. al. [15] traverse related APIs by structure overlap in framework to suggest API dependencies during programming with APIs. Holmes et. al. [16] use the relationship like “overriding” and “extension” to provide API usage guidance to beginners in learning Eclipse API. Besides the framework structure, hyperlinks between API documents like See Also tags are also utilized to specify important but implicative dependencies between APIs [17]. To provide programmers better understanding of selected APIs, CoDocent can traverse API documents by framework structure and the relationship between API documents.

IV. CoDocent

Figure 3 presents a screenshot of CoDocent, which has been implemented as an Eclipse plugin. To support the flow shown in Figure 1, CoDocent presents the class cloud view (Figure3 (a)) as an outline with traversing the related APIs to a selected code example. With the class cloud view, programmers can evaluate the relevance of the selected code example in the programming task. During API usage investigation, the API usage tip (Figure3 (b)) will pop out to confirm the selected API call within code example. Both class cloud view and API usage tip serve as guidance for investigating API documents. Programmers can click any specific API element presented in the diagrams (i.e., class cloud view and API usage tip) to review the corresponding API document in the document view (Figure 3 (c)).

To demonstrate the effectiveness of CoDocent, the class cloud view and API usage tip will be further explained as follows.

A. Class Cloud View

To provide outline for a code example, class cloud view summarizes all API classes traversed by CoDocent and organizes these classes according to the package structure. The term cloud presentation style is applied to characterize the importance of the traversed classes with different font sizes:

- The classes explicitly referred in the code example will be presented with large font size. These classes are regarded as most important ones demonstrated by the code example.
- The classes presented with medium font size are utilized when executing the code example but not explicitly referred in the code example. These classes are important to realize the execution mechanism behind the code example.
- The classes with small font size are traversed by CoDocent but not utilized with code example execution. These classes are structurally or conceptually relevant to API calls in the code example and helpful for programmers to adapt the APIs into his programming task.

For example, in Figure 3 (a), programmers realize that Snippet335 demonstrates the API usage about widgets such as Shell and Label with GridLayout. The Composite and Layout with medium font size are two framework classes related the execution mechanism behind Snippet335. And the classes with small font size, such as the subclasses of class Layout (FillLayout, StackLayout, etc.), expose the feasibility to adapt the code example with replacing layout styles. From Figure 3 (a), the programmer can recognize that Snippet335 is highly relevant to the SWT layout mechanism with grid layout style.

B. API Usage Tip

API usage tip aims to support programmers in investigating specific API call within the code example. When a programmer selects specific API call, API usage tip will organize and present the relevant APIs traversed by CoDocent to the programmer. To help the programmer navigate through the traversed APIs, API usage tip organizes the traversed APIs as different blocks according to their relations to the selected API call. To demonstrate the assistance of the API usage tip, an example about investigating the API calls in Snippet335 line 5 is illustrated in Figure 4.

The Snippet335 line 5 includes two API calls of the setLayout(Layout) method invocation and GridLayout() constructor invocation. To investigate the semantics of these API calls, the programmer selects the setLayout(Layout) method invocation. As shown in the API usage tip of Figure 4 (a), signature of the API call is presented. There are two mismatches between the signature
and the selected API call: (1) the signature shows that the method `setLayout(Layout)` is specified in class `Composite` but the API call invokes the method through a `Shell` object; and (2) the API call assigns a `GridLayout` object to parameter of `setLayout(Layout)`. The API usage tip provides the class hierarchies in block 1 and block 2 to help the programmer realize the usage reasons. With the API usage tip, the programmer can link API documents of `Composite` and `Layout` to check the layout mechanism in SWT framework.

After investigating the layout mechanism in SWT framework, the programmer further selects the API call of `GridLayout()` constructor invocation in the parameter of the method `setLayout()`. According to context of the API call in the code example, the API usage tip in Figure 4 (b) provides two different kinds of assistance. First, the API usage tip helps the programmer understand the usage of `GridLayout`. The block 3 reminds the programmer that `GridData` is referred in See Also tags of `GridLayout` API document. The programmer will check `GridData` while learning the use of class `GridLayout`. The block 5 shows alternative constructor to adapt this API call. Second, the API usage tip helps the programmer realize available layout styles for parameter configuration. As the block 4 shown in Figure 4 (b), there are several classes extended from class `Layout`. The objects of these class can be used to replace the parameter of the API call to configure
layout style. The block 6 in Figure 4 (b) lists available constructors of these classes to adapt this API call. By checking corresponding API documents in block 6, the programmer can be aware of the alternative layout styles in SWT.

V. EVALUATION

To evaluate the effectiveness of CoDocent, the following two questions should be addressed:

1) Does the class cloud view provide representative code example outline?
2) Do the traversed APIs support programmers adapt API calls into their programming tasks?

The question 1 is explained in Section V-A. For question 2, a case study is conducted to evaluate the usefulness of the traversed APIs and the results are shown in Section V-B.

A. Effectiveness of Class Cloud View

The SWT framework version 3.5 is chosen as the target framework in the experiment. SWT officially provides 332 code examples categorized in 62 categories. The top five categories are selected and aggregated as experiment subject. The code examples in “Shell” and “Display” categories are filtered because the uses of Shell and Display widgets are too common in almost every SWT code example. The selected categories and the number of code examples are listed in the first two columns of Table I.

For each pair of code examples, the similarity between the presented information in corresponding class cloud views is evaluated as a similarity value. UPGMA [18] is utilized to cluster these code examples based on similarity values. The clustering result is consisted of 16 groups. To match the clustering result in the original categorization, each group is analyzed with following criterion: group X matches to category A if the ratio of code examples belong to category A in group X is higher than or equal to 50%. According the matches between categories and groups, the TP, FP, FN, precision (TP/(TP+FP)) and recall (TP/Num) of each category are listed corresponding columns of Table I.

As shown in Table I, all categories in the clustering have high precision (above 77%) and high recall (above 70%). It means most code examples can be identified to the corresponding categories. For example, among the total 36 code examples in Table category, 34 code examples are correctly clustered into Table groups. The information provided by class cloud view is effective in outlining the API usage demonstrated by the code example. In another word, the class cloud view can effectively help programmers select a relevant code example among the candidates.

B. Effectiveness of Traversed APIs

A case study is conducted to show that the traversed APIs do help programmers use target API calls during performing programming tasks. The execution steps of the study are listed as followed.

1) Choose study subjects that include a framework; a code example as the baseline that demonstrates the use of API calls for achieving specific functionality; and a framework client software as investigation target.
2) Execute CoDocent on the code example to get APIr, which donates a set of APIs traversed by CoDocent but not referred within the code example.
3) Identify the classes in framework client software that implement functionality similar to that implemented in the code example.
4) Check whether the identified classes use APIs in APIr.

In this case study, the SWT framework is chosen as the target framework. The Snippet38 code example from official SWT website is selected as the baseline. The code example demonstrates the basic operations about creation of Table widgets and assigning its content using SWT API calls. JFace, a popular SWT framework client, is selected as the investigation target. In JFace, there are 7 classes that create the SWT Table and assign the content of the Table. The class name of the identified classes and corresponding APIr number used in those classes during constructing Table are listed in Table II.

As shown in Table II, all of the identified classes use at least 3 APIs in APIr during constructing Table widget. It means that JFace developers must traverse these additional APIs with selecting Snippet38 as the programming reference for accomplishing their programming tasks. A code example can only demonstrate a usage combination for a specific programming task. To accomplish the new but similar programming task based on a selected code example, programmers still must traverse additional APIs to confirm

<table>
<thead>
<tr>
<th>Category</th>
<th>Num</th>
<th>TP</th>
<th>FP</th>
<th>FN</th>
<th>Precision</th>
<th>Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table</td>
<td>36</td>
<td>34</td>
<td>6</td>
<td>2</td>
<td>85.00%</td>
<td>94.44%</td>
</tr>
<tr>
<td>Browser</td>
<td>15</td>
<td>14</td>
<td>4</td>
<td>1</td>
<td>77.78%</td>
<td>93.33%</td>
</tr>
<tr>
<td>Text</td>
<td>17</td>
<td>12</td>
<td>3</td>
<td>5</td>
<td>80.00%</td>
<td>70.59%</td>
</tr>
<tr>
<td>Tree</td>
<td>20</td>
<td>17</td>
<td>3</td>
<td>3</td>
<td>85.00%</td>
<td>85.00%</td>
</tr>
<tr>
<td>DnD</td>
<td>13</td>
<td>12</td>
<td>2</td>
<td>1</td>
<td>85.71%</td>
<td>92.31%</td>
</tr>
</tbody>
</table>

* TP: True Positive, FP: False Positive, FN: False Negative.

<table>
<thead>
<tr>
<th>Class Name</th>
<th>Num in APIr</th>
</tr>
</thead>
<tbody>
<tr>
<td>ContentProposalAdapter</td>
<td>4</td>
</tr>
<tr>
<td>ConfigureColumns</td>
<td>8</td>
</tr>
<tr>
<td>TableViewer</td>
<td>3</td>
</tr>
<tr>
<td>CompletionProposalPopup</td>
<td>4</td>
</tr>
<tr>
<td>ContextInformationPopup</td>
<td>3</td>
</tr>
<tr>
<td>CompletionProposalPopup2</td>
<td>3</td>
</tr>
<tr>
<td>ContextInformationPopup2</td>
<td>3</td>
</tr>
</tbody>
</table>
the adaptation of the API calls into their programming context. With traversing related APIs, CoDocent can support this need for programmers.

VI. Conclusion and Future Work

To support programming with APIs, many code search engines retrieve code examples as reference candidates for API usage. To further help programmers effectively select a relevant one, this work introduces CoDocent to link related API documents during reviewing the retrieved code examples. With traversing the related APIs according to referred APIs in a code example, CoDocent provides two diagrams, class cloud view and API usage tip, to guide programmers in investigating the code example. The class cloud view summarizes the outline for a code example to help programmers evaluate the relevance of the code example. During investigating the code example, the API usage tip organizes and presents relevant APIs for each API call referred within the code example. The API usage tip can guide programmers to investigate those API documents for confirming the semantics of the API calls. By the API usage tip, programmers can find the best way to adapt the API calls into their programming context.

The effectiveness of CoDocent has been assessed with an experiment and a case study. The results of the experiment show class cloud view can effectively help programmers select a relevant code example among a set of code example candidates. The case study also presents that the APIs traversed by CoDocent can support programmers in adapting API calls referred in a code example. As shown in the evaluation, CoDocent can be effectively used in selecting and investigating relevant code examples for programming with APIs. Following this work, more study about programming behavior with using CoDocent will be conducted in the near future. The study will provide heuristics for further support in helping programming task accomplishment based on the selected code example.

ACKNOWLEDGMENT

This research is supported by the Taiwan National Science Council (NSC) under contract NSC 97-2221-E-006-177-MY3.

REFERENCES


