# Recommending API Documentation

Chandan R. Rupakheti, Daqing Hou, and Russell Seidel
*Department of Electrical and Computer Engineering*
*Clarkson University, Potsdam, New York 13699*
{*rupakhcr, dhou, seidelrj*}*@clarkson.edu*

***Abstract*—APIs (Application Programming Interfaces) provide a way to access the functionalities behind software frameworks and libraries. To be usable, frameworks and libraries must be sufficiently documented. However, even with enough API documentation, programmers still face two more fundamental difficulties: 1) finding the right API elements for their task, and 2) knowing how to use them to achieve the desired solution. While search tools may help programmers solve the first problem, novices often have difficulty in formulating a useful search query and in assessing the quality and relevance of the search results. We present a recommendation system that is designed to address some of these problems. Based on a static analysis of the API client code, our system automatically recommends relevant API documentation. To evaluate the potential usefulness of the system, we manually analyzed 150 discussions from the Java Swing Forum, each corresponding to a real-world scenario where a programmer is having some difficulty with the API. For each case, we assessed how our recommendation system could help. We report our findings from this manual evaluation.**

***Keywords*-API; Critic; Recommendation Systems; Java; AWT/Swing; Forum Discussions**

## I. Introduction

Past studies on API usage have elicited two fundamental challenges in reuse-based development: 1) *finding* the right API elements for the programming task, and 2) properly *using* them to achieve the desired goal [3], [5]–[7], [10], [11]. RSSE systems (Recommendation Systems for Software Engineering) help programmers overcome both challenges by utilizing a developer's programming context and a system's data models to produce useful recommendations [12].

There have been significant efforts toward developing tools for finding relevant code and documentation. Even though search based tools help programmers by bringing code and documentation to IDEs [1] or by providing search interface to locate similar code and examples [4], [14], programmers still face a significant challenge in using such tools especially when they lack the knowledge of the framework's design to formulate the right search queries [9]. Sequence analysis tools such as MAPO [16] and PARSEWeb [15] build on and improve the results of code search engines but still require programmers to formulate the queries. Furthermore, assessing the quality of results returned by such tools may be a concern for novices who may not know enough of the semantics of the API elements present in the results. Hence, an RSSE system that is capable of making useful suggestions with minimum or zero involvement of the novice is deemed necessary in such a situation.

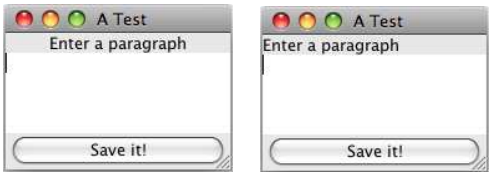
A key to a successful RSSE system is its ability to infer users' requirements and intentions from their programming context. In particular, the notion of context should not be limited to users' code, but be general enough to include information from both the requirement and the solution space. Utilizing a variety of artifacts such as requirement specifications, design documents, source code, and API usage rules and conventions, RSSE should be able to recommend a useful set of actions, software features, and help documents to assist the users. With these concepts in mind, we have developed a recommendation system called **CriticAL** (A **Critic** for **A**PIs and **L**ibraries) [13] [1].

CrticAL helps programmers by critiquing their code in three ways: 1) it *explains* the complex interactions between API elements, 2) it *criticizes* the improper use of the API, and 3) it *recommends* other related API elements for future use. CriticAL achieves these based on the symbolic execution [8] of the API client code. CriticAL is configured with expert-prepared API usage rules and documentation. When a rule gets triggered by the current state of the client code, useful advice is presented for the code. For such an approach to work, we must answer two key research questions:

1) What are the characteristics of API usage problems that can be helped through recommendations? Can we formulate useful rules by analyzing these problems?
2) Do API usage problems recur? Can we justify the investment for API experts to prepare the rules and documentation for a recommendation system?

To answer these questions concretely through field data, we conducted a case study using the Java Swing Forum [2], focusing on problems related to *GUI composition and layout*. We summarize the classification of 150 discussion threads as well as the recommendation rules discovered in the process. Our main contributions are the answers to the two research questions above by showing that similar API usage problems recur in practice and that they can be handled by formulating usage rules in terms of the programming context. We also demonstrate the use of recommendations in three kinds of contexts, that is, *generic recommendations* (Sec-

[1] http://sf.net/p/critical (All URLs verified on Feburary 14, 2012.)
[2] https://forums.oracle.com/forums/forum.jspa?forumID=950




(a) Desired design. (b) Alignment problem with the label on top.

Figure 1. User requirement and achieved solution.

tion III-A), *syntax-based recommendations* (Section III-B), and *program-state-based recommendations* (Section III-C).

The rest of the paper is organized as follows: Section II demonstrates the use of CriticAL with an example, Section III summarizes our findings from the analysis of Swing forum, and finally, Section IV concludes the paper.

## II. A Motivating Example

In this section, we present two simple use cases of CriticAL to motivate readers, with each case representing a possible different stage and level of understanding of the Swing API from a novice.

### A. Case 1: Learning to Use the API

Consider the scenario where a novice programmer learns the Java Swing API by designing the GUI shown in Figure 1(a). Assume that he does not know much more than that `JFrame` class can be used to make the window. This assumption can be valid as several Swing forum users had been found to have exactly such problems [3] [4]. Figure 2(a) shows that the user manages to create an empty `JFrame` object in the Eclipse IDE and probably with the help of Content Assist, is able to set the title of the window. But the user eventually gets stuck. Working from such a preliminary stage, CriticAL will iteratively guide him through the coding process to achieve the desired solution, where it will reason about his API code and provide critiques within the IDE.

To get help from CriticAL, the user presses the help button (Figure 2(a)). As a result, CriticAL symbolically executes the user's code and checks the API usage rules. In this case, it finds that the `JFrame` object is missing some important properties on the execution path, and consequently, some critiquing rules are triggered. The triggered rules are shown as markers, for example, the one at line 6 of Figure 2(a).

The user right-clicks over the marker to get the context menu as shown in Figure 2(b). The context menu lists all of the criticisms, explanations, and recommendations made by CriticAL for the given line number. On hovering over each of the menu item under *Critiques*, CriticAL displays a short tool-tip message describing the critique. In this case, Figure 2(b) shows a tool-tip for the criticism generated due to the violation of the rule that *a top-level GUI widget if initialized, must eventually be visible.* Figure 2(c) shows the

[3]https://forums.oracle.com/forums/thread.jspa?messageID=5768843
[4]https://forums.oracle.com/forums/thread.jspa?messageID=5848584

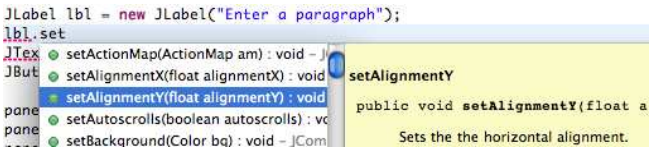



Figure 3. Eclipse's content assist showing alignment related methods.

explanation generated when *the default layout manager of the content pane of the `JFrame` is used instead of the user-configured one.* And Figure 2(d) shows the recommendation generated when *the content pane of a `JFrame` is empty.* When the user clicks on a menu item, CriticAL opens a detailed view showing the description of the problem with code snippets as well as links to the related online documents. In this case, Figure 2(e) shows the detailed document for the recommendation of Figure 2(d). Hence, the user can now learn about `JFrame` and `BorderLayout` and even copy-and-paste the code snippet into the editor.

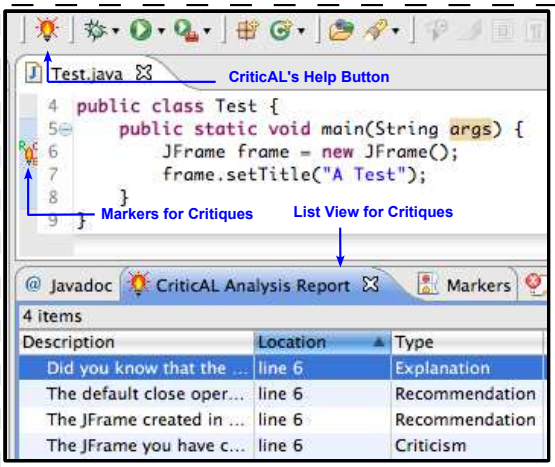
Listing 1. The user's code that results in the `JFrame` of Figure 1(b).

```
1  // Initialize a frame and other components
2     JFrame frame = new JFrame(''A Test'');
3     JPanel panel = (JPanel)frame.getContentPane();
4     JLabel lbl = new JLabel(''Enter a paragraph'');
5     JTextArea textArea = new JTextArea(4,15);
6     JButton button = new JButton(''Save it!'');
7  // Add components to the content pane
8     panel.add(lbl, BorderLayout.PAGE_START);
9     panel.add(textArea, BorderLayout.CENTER);
10    panel.add(button, BorderLayout.PAGE_END);
11 // Compute the frame's size and display
12    frame.pack();
13    frame.setVisible(true);
```

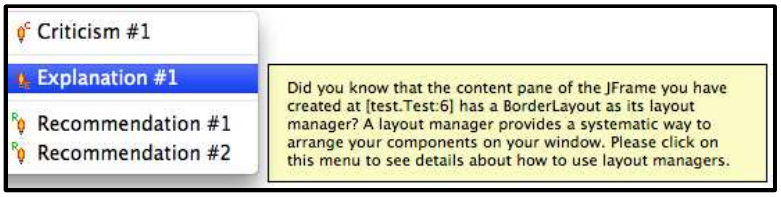
### B. Case 2: Using the API

Assume that the user now understands how to use `JFrame` and `BorderLayout` and produces the code in Listing 1. When he runs the code, he gets the window shown in Figure 1(b) where the label is aligned to the left, rather than in the center. Using Content Assist (Figure 3), he suspects that the `setAlignmentX(float)` method could be used to align a component horizontally, but to no avail. He gets further confused when discovering that there are two more methods with similar sounding functionalities, that is, `setHorizontalAlignment()` and `setHorizontalTextPosition()`, but none of their Javadoc communicates how to use them. CrticAL would again help him by recommending a document that distinguishes these API methods by the respective situations they are designed for. After reading this document, the user would insert `lbl.setHorizontalAlignment(JLabel.CENTER)` in line 7 to finally get the desired GUI of Figure 1(a). These methods will be discussed in detail in Section III-B1.

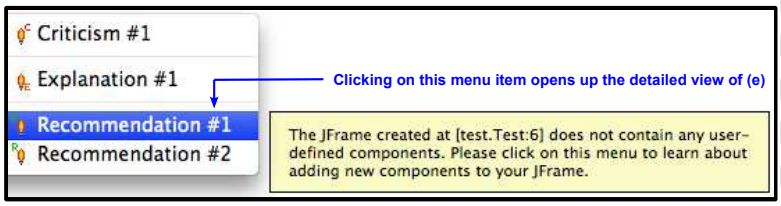
This iterative process of providing contextual recommendation is a rather ambitious goal and we do not claim that



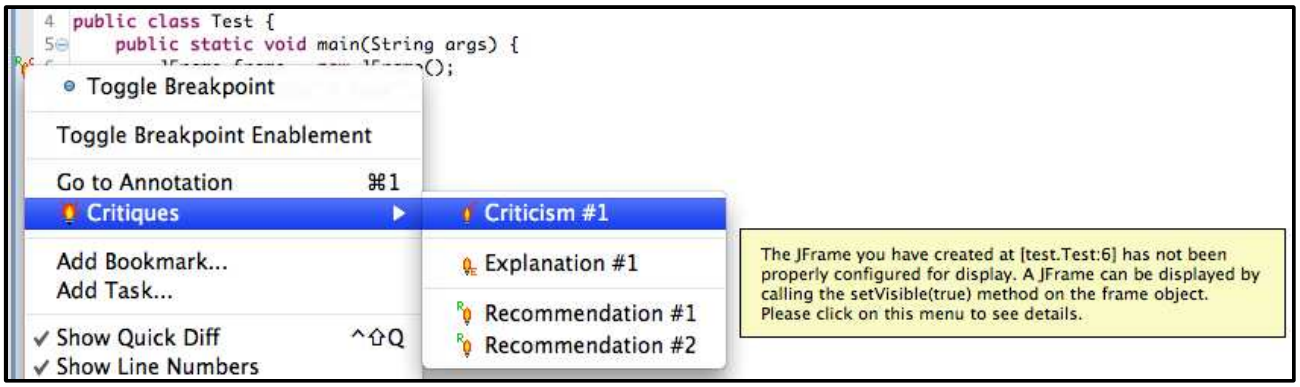

(a) CriticAL Plugin in Eclipse IDE with the user's code



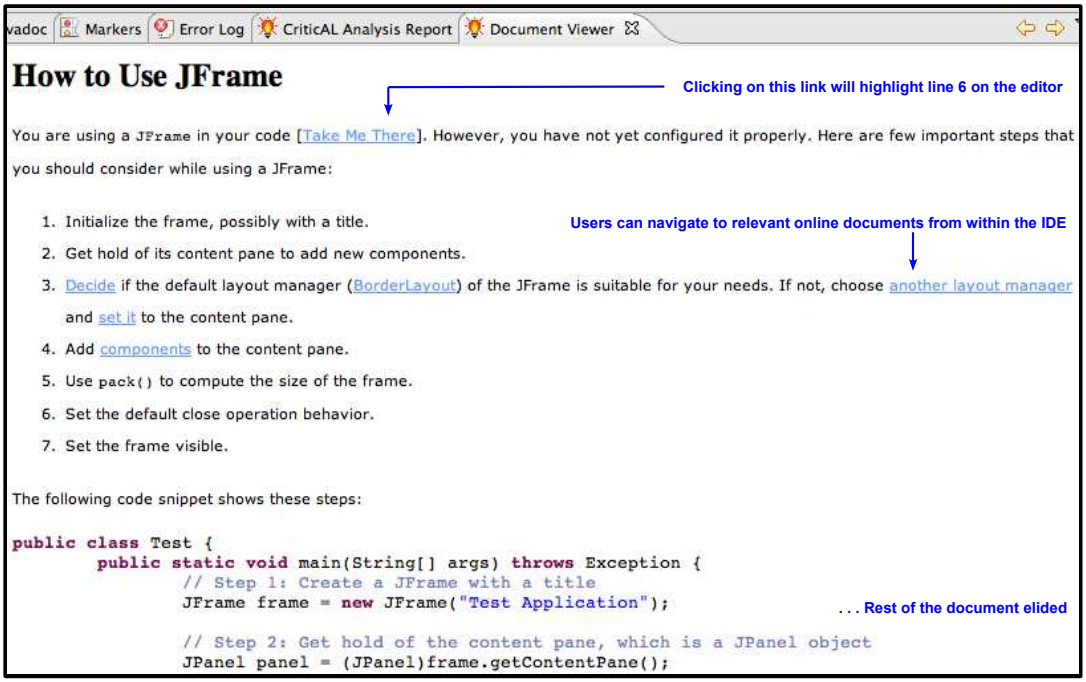

(c) Context menu showing an explanation.




(d) Context menu showing a recommendation.




(b) Context menu showing a criticism of the user's code at line 6. Each critique has a short tool-tip description.




(e) Each critique also has a detailed description that can be accessed by clicking on the menu item in the context menu. This particular document is associated with the recommendation shown in (d).

Figure 2. CriticAL helping a programmer build the Swing application shown in Figure 1(a) using a `JFrame`.

we have completely achieved it. Currently, we recommend bits and pieces that may serve a particular moment in a long programming process. To achieve this ambitious goal, the tool need to adapt to different levels of expertise and provide sufficient coverage. Nevertheless, these recommendations are already useful. The basis for justifying these recommendations is because they are the normal way of using the API (for Section II-A) and similar problems recur in the forum (for Section II-B). The cost of recommending in these cases is minimal because these recommendations passively appear as a marker in the IDE and do not interfere with the user's regular programming, but the benefit can be substantial.

## III. Case Study of Swing Forum

We have shown in Section II how CriticAL can help programmers learn and use APIs. In order to be useful, however, we need to identify the characteristics of API usage problems experienced by practical programmers and learn to formulate concrete API usage rules to solve these problems. We also need to justify this effort with solid evidence from the field. We performed a manual investigation of the Java Swing forum to address these issues.

The manual analysis was conducted using Eisenhardt's case study research methodology [2], where each thread served as a case for our research. To isolate manageable cases from 46,000 message threads, the forum's default search engine was used with "layout" as the keyword. We sequentially analyzed 150 message threads out of 264 returned by the search [5]. Each thread contains multiple discussions with questions, answers, and code snippets. Most of the snippets are self-contained compilable programs that users can run to assess the situation of the original poster (OP). We studied each thread thoroughly classifying it under the rules that apply. After filtering out unclear and irrlevant threads, we found that 60 threads could potentially be helped through recommendations, 50 through criticisms, and 21 through explanations [6]. In this paper, we will mostly focus on recommendations, as summarized in Table I. Note that the total number of threads in the table is more than 60 because a thread may be helped by multiple recommendations.

### A. Generic Recommendations

Often it happens in reuse-based development that a novice programmer does not know how to start with his code because he has not completely planned out how to map his requirements to the relevant API elements. For example, consider the following message from the forum [7]:

[5]https://forums.oracle.com/forums/search.jspa?q=layout&objID=f950&dateRange=all&numResults=15&rankBy=10001

[6]The results of our analysis consists of a spreadsheet for the classification and an Eclipse project containing 189 runnable test programs extracted from the discussions at http://sf.net/projects/critical/files/results/.

[7]https://forums.oracle.com/forums/thread.jspa?messageID=5833295

Table I
Classification of recommendations discovered in the forum.

| Section | Description | Total |
|---|---|---|
| **Generic Recommendations** | | |
| III-A | Generic Recommendations | 43 |
| **Syntax-Based Recommendations** | | |
| III-B1 | Confusing APIs | 5 |
| III-B2 | Lite Context | 6 |
| **State-Based Recommendations** | | |
| III-C1 | Unused Features | 7 |
| III-C2 | Alternative Design | 3 |
| **Total** | | **64** |

*"Hi, im [i am] quite new to java, and just want to know, how to layout labels and buttons onscreen, [... elided]. If someone could print some sample code with a sample label and layout details that would be a great help. [...]"*

Most such problems can be solved through documents that are already available online, but they need to be brought to the programmer's attention. We found 43 instances that can be helped through a generic document containing a list of *How-Tos* about the layout API such as "How to use layout managers", "How to achieve absolute positioning", and "How to combine multiple layout managers." In fact, many forum replies, including the one shown above, contain a link to the Swing layout tutorial [8] in their answers, which can be pushed as a generic documentation. CriticAL can recommend such documents to users without any context, or as soon as it detects the presence of a Swing API element in their code. Hence the title of this section.

### B. Syntax-Based Recommendation

Generic recommendations are applicable without, or with only minimal, programming context. CriticAL can provide more directed recommendations based on the syntax and the signature of the API methods used in the code. We have identified two useful syntax-based recommendations.

*1) Confusing API Methods:* A user may be confused by multiple methods that all look similar, not knowing which is the correct one to use [9]:

- `setAlignmentX()` and `setAlignmentY()` of `JComponent` used with parameters such as `TOP_ALIGNMENT` and `CENTER_ALIGNMENT`. They are used to position widgets within a `BoxLayout`.
- `setHorizontalAlignment()` and `setVerticalAlignment()` of `JLabel` and `JButton` with parameters such as `LEFT` and `CENTER`. They are used to align the content within a widget itself.
- `setHorizontalTextPosition()` and `setVerticalTextPosition()` of `JLabel` and `JButton` with parameters such as `LEFT` and `TOP`. They are used to align the text within a `JLabel` and a `JButton` relative to its icon image.

[8]http://docs.oracle.com/javase/tutorial/uiswing/layout/

[9]http://forums.oracle.com/forums/thread.jspa?messageID=5827139

CriticAL makes a recommendation of all three kinds of methods after detecting the presence of anyone of the confusing methods from the code. The recommendation explains the suitable situation for each method. Hence it helps the user choose the correct method.

*2) Layout Recommendation in Lite Context:* It is sometimes possible to infer partial requirements of a user based on the API elements used in his code. Based on such inferences, a recommendation for possible layout managers can be made. For instance, when a user has multiple components in a container, one of which is a `JTextArea`, we can recommend the `CENTER` location of `BorderLayout` for the text area on the basis that it can grow in both direction, e.g., [10]. Such a recommendation would also list other possibilities such as `BoxLayout` and `GridLayout` in case the user wants the text area to grow proportionally with other widgets, but definitely not `FlowLayout`, which keeps components at their fixed preferred sizes. Furthermore, a `GridbagLayout` can also be recommended in case the user wants to partition the container disproportionately.

### C. State-Based Recommendations

By reasoning about the program state in the client code, CriticAL can capture the programming context of the user more concretely and precisely and, thus, produce more pertinent recommendations. We present two such examples.

*1) Recommendations of Unused Features:* We have identified some of the common features used in GUI programming from the analysis of the forum. CriticAL can be configured to recommend those features to users based on the current state of their program. For instance, the recommendation of a border for the `JPanel` in [11] would help solve the problem of the user related to shifting components right by a small distance from the left edge of the container. Similarly, recommendation for unused alignment properties of `FlowLayout` and `BoxLayout` or missing constraints for `BorderLayout` and `GridBagLayout` may help novices in using those API elements properly. Such recommendations help programmers discover potentially unknown features of the API that they have been using.

*2) Alternative Design:* APIs are designed to solve a particular set of problems. It is almost always beneficial to use the corresponding API elements for the intended problem rather than trying other API elements and complicating the code. For instance, the `GridbagConstraints` used in [12] tries to mimic the behavior of a `FlowLayout` with left alignment. The alternative solution is much shorter and achieves similar effect and can be recommended to the user. Of course, the detection of alternative design requires CriticAL to be pre-configured with instances of easily confused alternative cases found in the forum.

[10] http://forums.oracle.com/forums/thread.jspa?messageID=9201990
[11] http://forums.oracle.com/forums/thread.jspa?messageID=5716439
[12] https://forums.oracle.com/forums/thread.jspa?messageID=5729204

## IV. Conclusion

We present CriticAL, a recommendation system designed to help programmers achieve their solution through an iterative process of critiquing within an IDE. CriticAL helps bridge the long standing information gap between the API designers and application programmers by bringing important API related information to the workbench where it is needed the most. Our study shows that the API usage problems that CriticAL helps solve recur in practice and that the effort on creating API usage rules and documentation can be justified. We also distinguish between three kinds of recommendations, that is, *generic*, *syntax-based*, and *state-based*, and illustrate them with concrete examples from the forum. Future research should further investigate the quality, quantity, and usability of these recommendations.

## References


[1] J. Brandt, M. Dontcheva, M. Weskamp, and S. R. Klemmer, "Example-centric programming: Integrating web search into the development environment," in *CHI*, 2010, pp. 513–522.

[2] K. M. Eisenhardt, "Building Theories from Case Study Research," *Academy of Management Review*, vol. 14, no. 4, pp. 532–550, 1989.

[3] G. Fischer, "Cognitive view of reuse and redesign," *IEEE Softw.*, vol. 4, pp. 60–72, July 1987.

[4] M. Grechanik, C. Fu, Q. Xie, C. McMillan, D. Poshyvanyk, and C. Cumby, "A search engine for finding highly relevant applications," in *ICSE*, 2010, pp. 475–484.

[5] D. Hou, "Investigating the effects of framework design knowledge in example-based framework learning," in *ICSM*, 2008, pp. 37–46.

[6] D. Hou and L. Li, "Obstacles in using frameworks and APIs: An exploratory study of programmers' newsgroup discussions," in *ICPC*, 2011, pp. 91–100.

[7] D. Hou, C. Rupakheti, and H. Hoover, "Documenting and evaluating scattered concerns for framework usability: A case study," in *APSEC*, 2008, pp. 213 –220.

[8] J. C. King, "Symbolic execution and program testing," *Commun. ACM*, vol. 19, pp. 385–394, July 1976.

[9] A. J. Ko and Y. Riche, "The role of conceptual knowledge in API usability," in *VL/HCC*, 2011, pp. 173–176.

[10] A. J. Ko, B. A. Myers, and H. H. Aung, "Six learning barriers in end-user programming systems," in *VL/HCC*, 2004, pp. 199–206.

[11] M. P. Robillard and R. Deline, "A field study of API learning obstacles," *Empirical Softw. Engg.*, vol. 16, pp. 703–732, 2011.

[12] M. P. Robillard, R. J. Walker, and T. Zimmermann, "Recommendation systems for software engineering," *IEEE Software*, vol. 27, no. 4, pp. 80–86, July/August 2010.

[13] C. R. Rupakheti and D. Hou, "Satisfying programmers' information needs in API-based programming," in *ICPC*, 2011, pp. 250–253.

[14] J. Stylos and B. A. Myers, "Mica: A web-search tool for finding API components and examples," in *VLHCC*, 2006, pp. 195–202.

[15] S. Thummalapenta and T. Xie, "Parseweb: A programmer assistant for reusing open source code on the web," in *ASE*, 2007, pp. 204–213.

[16] T. Xie and J. Pei, "MAPO: mining API usages from open source repositories," in *MSR*, 2006, pp. 54–57.