CriticAL: A Critic for APIs and Libraries

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

Abstract—It is well-known that APIs can be hard to learn and use. Although search tools can help find related code examples, API novices still face other significant challenges such as evaluating the relevance of the search results. To help address the broad problems of finding, understanding, and debugging API-based solutions, we have built a critic system that offers recommendations, explanations, and criticisms for API client code. Our critic takes API usage rules as input, performs symbolic execution to check that the client code has followed these rules properly, and generates advice as output to help improve the client code. We demonstrate our critic by applying it to a real-world example derived from the Java Swing Forum.

Keywords-API; Critic; Symbolic Execution; AWT/Swing

I. INTRODUCTION

Behind each API (Application Programming Interfaces) is a system that offers proven solutions for a set of common problems in some domain. The API facilitates the access to such a system so that new systems can be built on top of it. To be effective in using the API, one must learn enough of the domain, its problems and solutions, and how to map between them appropriately. For a system of rich functionalities with a large problem and solution space, learning its API can be a substantial endeavor [3],[6].

Due to time pressure and an urge to solve problems quickly, many programmers prefer to learn API’s on demand and learn by doing. That is, they try to learn just enough of an API so that they can solve the current task. Search tools can partially support this practice by helping locate relevant code examples [1]. But programmers, especially novices, cannot always formulate good queries for what they are looking for. Furthermore, even if they find relevant code examples, they would still face the significant challenge to understand and evaluate them for relevance.

With only limited knowledge of the API, a novice’s solution is often incomplete, incorrect or suboptimal. Programmers can seek help from online forums, but there can be some time lag before they can get one. It would be ideal to engage a human expert who has substantive experience with the API. Our experience with building a critic for the GUI layout logic is that the cost of building such a useful critic is reasonable [7].

Our critic system requires a set of API usage rules and associated documentation to explain these rules. The API rules specify special conditions about program states for which advice should be produced. The critic symbolically executes API client code to obtain program states [4]. Based on the resulting states as well as knowledge of the API usage rules, our critic generates contextual advice. It is assumed that a human expert who has substantive experience with the API will design and implement the rules. 

Our critic takes API usage rules as input, performs symbolic execution to check that the client code has followed these rules properly, and generates advice as output to help improve the client code. We demonstrate our critic by applying it to a real-world example derived from the Java Swing Forum.

CriticAL: A Critic for APIs and Libraries

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

Abstract—It is well-known that APIs can be hard to learn and use. Although search tools can help find related code examples, API novices still face other significant challenges such as evaluating the relevance of the search results. To help address the broad problems of finding, understanding, and debugging API-based solutions, we have built a critic system that offers recommendations, explanations, and criticisms for API client code. Our critic takes API usage rules as input, performs symbolic execution to check that the client code has followed these rules properly, and generates advice as output to help improve the client code. We demonstrate our critic by applying it to a real-world example derived from the Java Swing Forum.

Keywords-API; Critic; Symbolic Execution; AWT/Swing

I. INTRODUCTION

Behind each API (Application Programming Interfaces) is a system that offers proven solutions for a set of common problems in some domain. The API facilitates the access to such a system so that new systems can be built on top of it. To be effective in using the API, one must learn enough of the domain, its problems and solutions, and how to map between them appropriately. For a system of rich functionalities with a large problem and solution space, learning its API can be a substantial endeavor [3],[6].

Due to time pressure and an urge to solve problems quickly, many programmers prefer to learn API’s on demand and learn by doing. That is, they try to learn just enough of an API so that they can solve the current task. Search tools can partially support this practice by helping locate relevant code examples [1]. But programmers, especially novices, cannot always formulate good queries for what they are looking for. Furthermore, even if they find relevant code examples, they would still face the significant challenge to understand and evaluate them for relevance.

With only limited knowledge of the API, a novice’s solution is often incomplete, incorrect or suboptimal. Programmers can seek help from online forums, but there can be some time lag before they can get one. It would be ideal to engage a human expert who has substantive experience with the API. Our experience with building a critic for the GUI layout logic is that the cost of building such a useful critic is reasonable [7].

Our critic system requires a set of API usage rules and associated documentation to explain these rules. The API rules specify special conditions about program states for which advice should be produced. The critic symbolically executes API client code to obtain program states [4]. Based on the resulting states as well as knowledge of the API usage rules, our critic generates contextual advice. It is assumed that a human expert who has substantive experience with the API will design and implement the rules. 

Our critic takes API usage rules as input, performs symbolic execution to check that the client code has followed these rules properly, and generates advice as output to help improve the client code. We demonstrate our critic by applying it to a real-world example derived from the Java Swing Forum.

CriticAL: A Critic for APIs and Libraries

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

Abstract—It is well-known that APIs can be hard to learn and use. Although search tools can help find related code examples, API novices still face other significant challenges such as evaluating the relevance of the search results. To help address the broad problems of finding, understanding, and debugging API-based solutions, we have built a critic system that offers recommendations, explanations, and criticisms for API client code. Our critic takes API usage rules as input, performs symbolic execution to check that the client code has followed these rules properly, and generates advice as output to help improve the client code. We demonstrate our critic by applying it to a real-world example derived from the Java Swing Forum.

Keywords-API; Critic; Symbolic Execution; AWT/Swing

I. INTRODUCTION

Behind each API (Application Programming Interfaces) is a system that offers proven solutions for a set of common problems in some domain. The API facilitates the access to such a system so that new systems can be built on top of it. To be effective in using the API, one must learn enough of the domain, its problems and solutions, and how to map between them appropriately. For a system of rich functionalities with a large problem and solution space, learning its API can be a substantial endeavor [3],[6].

Due to time pressure and an urge to solve problems quickly, many programmers prefer to learn API’s on demand and learn by doing. That is, they try to learn just enough of an API so that they can solve the current task. Search tools can partially support this practice by helping locate relevant code examples [1]. But programmers, especially novices, cannot always formulate good queries for what they are looking for. Furthermore, even if they find relevant code examples, they would still face the significant challenge to understand and evaluate them for relevance.

With only limited knowledge of the API, a novice’s solution is often incomplete, incorrect or suboptimal. Programmers can seek help from online forums, but there can be some time lag before they can get one. It would be ideal to engage a human expert who has substantive experience with the API. Our experience with building a critic for the GUI layout logic is that the cost of building such a useful critic is reasonable [7].

Our critic system requires a set of API usage rules and associated documentation to explain these rules. The API rules specify special conditions about program states for which advice should be produced. The critic symbolically executes API client code to obtain program states [4]. Based on the resulting states as well as knowledge of the API usage rules, our critic generates contextual advice. It is assumed that a human expert who has substantive experience with the API will design and implement the rules. 

Our critic takes API usage rules as input, performs symbolic execution to check that the client code has followed these rules properly, and generates advice as output to help improve the client code. We demonstrate our critic by applying it to a real-world example derived from the Java Swing Forum.

CriticAL: A Critic for APIs and Libraries

Chandan R. Rupakheti, Daqing Hou
Department of Electrical and Computer Engineering
Clarkson University, Potsdam, New York 13699
{rupakhr, dhou}@clarkson.edu
Fig. 1. CriticAL helps a programmer build the Swing application shown in Figure 2. produces as well as the desired GUI. Using CriticAL for help, the programmer may press the CriticAL button that is shown in Figure 1(a). As a result, CriticAL symbolically executes the code in Figure 1(a) to create the states for the program, which are shown in Table I. Our critic checks the program states against API use rules to infer the current status of the program as well as the programmer’s intent and goals for the program, and to offer advice. Generated advice are presented as markers on the ruler of the text editor (the left-hand side of Figure 1(a)), indicating that CriticAL has critiques (E: Explanation, R: Recommendation, and C: Criticism) for the code at the corresponding lines. Because the number of critiques at each line is small, we have not found it necessary to rank them.

A. Explanation

The facts holding in program states can be used to help the programmer understand why the program exhibits a certain behavior. For instance, CriticAL finds that a component is added to the center location of the `BorderLayout` that manages the content pane (line 29, Table I). On hovering over the expla-
TABLE I
SYMBOLIC STATES FOR THE PROGRAM OF FIGURE 1(A).

<table>
<thead>
<tr>
<th>Line #</th>
<th>Facts that hold after line # for symbolic objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>14</td>
<td>example.parent = null</td>
</tr>
<tr>
<td></td>
<td>example.layout = FlowLayout()</td>
</tr>
<tr>
<td></td>
<td>example.lblTotalRepairs = JLabel(...)</td>
</tr>
<tr>
<td></td>
<td>example.button1 = JButton(...)</td>
</tr>
<tr>
<td>15</td>
<td>frame.title = “This is the window”</td>
</tr>
<tr>
<td></td>
<td>frame.visible = false</td>
</tr>
<tr>
<td></td>
<td>frame.contentPane.children = []</td>
</tr>
<tr>
<td>16</td>
<td>frame.contentPane.layout = BorderLayout()</td>
</tr>
<tr>
<td></td>
<td>frame.contentPane.properties = []</td>
</tr>
<tr>
<td>22</td>
<td>example.layout = GridLayout(2,2)</td>
</tr>
<tr>
<td>28</td>
<td>labelPanel.children = [lblTotalRepairs, lblRepeatRepairs]</td>
</tr>
<tr>
<td></td>
<td>fieldPanel.children = [totalRepairs, repeatRepairs]</td>
</tr>
<tr>
<td></td>
<td>buttonPanel.parent = [button1]</td>
</tr>
<tr>
<td>29</td>
<td>frame.contentPane.children = [fieldPanel]</td>
</tr>
<tr>
<td></td>
<td>fieldPanel.parent = frame.contentPane</td>
</tr>
<tr>
<td></td>
<td>frame.contentPane.layout.properties = [CENTER:fieldPanel]</td>
</tr>
<tr>
<td>30</td>
<td>frame.visible = true</td>
</tr>
<tr>
<td></td>
<td>frame.defaultCloseOperation = HIDE_ON_CLOSE</td>
</tr>
<tr>
<td></td>
<td>buttonPanel.parent = null</td>
</tr>
<tr>
<td></td>
<td>buttonPanel.parent = null</td>
</tr>
<tr>
<td></td>
<td>example.parent = null</td>
</tr>
</tbody>
</table>

In general, every critique has a short tool-tip description and a detailed explanation document, which can be stored either locally or remotely on the Internet. Assume that to fix the problem, the programmer added the two orphan panels to the frame. Now, at line 30, where the GUI is made visible, the critic detects that labelPanel contains two JLabel's as its children, and fieldPanel contains two JTextField's (from facts at line 28 of Table I). By examining this symbolic GUI data structure, our critic infers that the programmer is creating a 2-by-2 table. The critic is also able to conclude that this way of making a table is problematic as it will be impossible to properly align a label and its corresponding text field. Instead, such a table can be made in a single container using SpringLayout, GridLayout, or GridBagLayout. These information have been added to the critic as a rule of criticism.

III. CONCLUSION AND STATUS

We describe an API critic that is designed to significantly improve the experience of learning to use a new API. The key elements for our critic consist of symbolic program states, API use rules, and advising documentation. We have built a critic framework that is capable of symbolically executing API client code. Each API usage rule is implemented as a plugin for this framework. We have identified and implemented a set of API rules for the AWT/Swing layout logic [7]. Future work includes in-depth validation, building critics for other APIs to show the generality of the critic approach, and empirical evaluation of the critics with human subjects. Our working prototype CriticAL for the Java Swing API is open-source.

REFERENCES