Obstacles in Using Frameworks and APIs: An Exploratory Study of Programmers' Newsgroup Discussions

Daqing Hou and Lin Li
Department of Electrical and Computer Engineering
Clarkson University
Potsdam, NY USA 13699
Email: {dhou,lil}@clarkson.edu

***Abstract*—Large software frameworks and APIs can be hard to learn and use, impeding software productivity. But what are the specific challenges that programmers actually face when using frameworks and APIs in practice? What makes APIs hard to use, and what can be done to alleviate the problems associated with API usability and learnability? To explore these questions, we conducted an exploratory study in which we manually analyzed a set of newsgroup discussions about specific challenges that programmers had about a software framework. Based on this set of data, we identified several categories of obstacles in using APIs. We discussed what could be done to help overcome these obstacles.**

***Keywords*-Frameworks; APIs; Usability; Case Studies; AWT/Swing**

I. Introduction

Software libraries, frameworks, and *Application Programming Interfaces* (APIs) are becoming indispensable in modern software development, and it is well-known that they can be hard to learn and use [1] [2] [3] [4]. However, what is less known about is exactly what constitutes that difficulty, and what can be done to alleviate it. For example, what do programmers actually question about an API in practice, and why? What existing artifacts and tools, e.g., online tutorials and forums, API documentation, and browsing and searching tools, can help with answering these questions? Where do existing tools break down? What additional capabilities and tools should and can be provided?

To investigate these questions and identify opportunities for improving the learning and use of APIs, we conducted an exploratory study of programmer newsgroup discussions about APIs. We selected a set of API-related discussions from a newsgroup and analyzed them in detail. From the results of analyzing the newsgroup discussions, we identified a set of specific API obstacles (Table II). We also discussed how potential new tools could be built to help programmers overcome these obstacles more effectively (Section VI).

In this study, we followed Eisenhardt's approach to case study research [5] (Section III). Since relatively little is known about the specifics of API obstacles, the goal of this study has been to identify important characteristics of API obstacles, not to test established hypotheses.

Following Fischer's conceptual framework for reuse and redesign [1], in our analysis of each newsgroup discussion, we distinguished between the notions of *task* and *solution*. A programmer generated a question about APIs in the context of creating a solution for a particular task. For each discussion, we strived to recognize what was the task that the original poster (OP) wanted to accomplish, how much progress OP had made in the transition from the task to a solution, and what the ultimate solution(s) would be, if they existed. At the top level, we organized the analyzed discussions into categories according to OP's current stage of progress. Within the category for each stage, we further divided the discussions according to the solution categories. This two-dimensional organization of our data has allowed us to identify the API obstacles more easily (Section IV).

The main contribution of this paper is the set of API obstacles identified from this analysis as well as a discussion of their implications for tools and practices (Section VI). In addition to informing future tooling efforts, these obstacles could also be further tested using the HCI approaches [2] [6] [7] [8].

The remainder of this paper is organized as follows. Section II compares this study with related work. Section III describes our research method. Section IV presents the identified obstacles. Additional observations derived from our data are presented in Section V. Section VI discusses the implications that the identified obstacles may have on future tool building efforts. Finally, Section VII discusses threats to validity, and Section VIII concludes the paper.

The data we used in this study can be found at this url:
http://www.clarkson.edu/~dhou/projects/icpc2011.tar.gz

II. Related Work

In [1], Fischer outlined a conceptual framework for reuse and redesign as well as describing several software tools motivated by that framework. His framework views design tasks as translation of the problems expressed in the situation model, often informal, incomplete, to the more precise, formal system model. The translation requires the construction of the right sequence of operations to yield the desired solution. Furthermore, the framework considers

a programmer's information needs during this translation process from a cognitive perspective. This has led to several important observations. For instance, novices might have difficulty in forming queries for the (large) API space since they do not know what they don't know. We also observed the same vocabulary problem in our data. In general, our study has been informed by this conceptual framework.

Robillard [9] and Robillard and DeLine [4] report a set of obstacles faced by Microsoft developers learning a wide variety of new APIs. These include issues such as inadequate or missing documentation for subtle design aspects, improper design structures, and difficulty in learning to compose larger solutions based on studying available code examples and tutorials. The obstacles we identified in this study largely overlap with theirs, but appear to be at a lower level of details. This could be explained by the different research methods employed. Their study used a combination of surveys and in-person interviews, and collected the opinions and experiences of several hundreds of professional developers, while ours involved the detailed analysis of programmers' newsgroup discussions about API use. A strength of their study would be stronger external validity and a broader scope of issues considered. However, the results of our study would be more useful or suitable as use cases for future tool building efforts since they are richer in details.

Ko, Myers, and Aung, based on observing how students learned Visual Basic.Net and their moments of breakdown, proposed six learning barriers for end-user programming systems [3]. Our study provides another but richer set of evidence corroborating the usefulness and relevance of the six barriers. Our results not only show the existence of such barriers, but also include a classification of their causes. The obstacles we identified could be used to motivate the building of new tools to reduce the barriers. We have also manually classified a subset of our data (64 cases) into the six learning barriers. The details of the classification process and our lessons learned can be found in [10].

In [11], we explored the critical roles that framework design knowledge can play in the process of using a framework to build applications, especially its interaction and relationship with using code examples as a learning aid. Robillard's recent study reports supporting evidence that professional programmers do consider it important to understand design aspects and rationale for various purposes, such as choosing between alternative ways of using APIs, structuring code accordingly, and using APIs more efficiently [9].

A study similar to the current one was reported previously [12]. Both studies observed the same framework, and collected and analyzed programmer discussions from the same newsgroup, but with distinctly different goals. Although also an exploratory study, the goal of [12] was to search for evidence that the way in which design features are structured may have an impact on their ease of learning and use. To that end, we picked two Swing widgets, JButton and JTree, to study. JButton serves as a representative of a simple component which reveals issues related to deep inheritance, and JTree represents a typical complex component with rich composition. Each question was categorized by the design features of the corresponding component. This classification highlighted the most problematic design features. We reflected on the reasons why they might have been difficult.

Since APIs are also a kind of interface between humans and an underlying computing system, HCI approaches have been applied to study API use. For example, several user studies have been conducted to test well-formulated hypotheses about API usability design choices [6] [7] [8]. Formative user studies have also been employed to gather feedback to improve the design of an API [2]. The results of our study could be further refined and formalized into design guidelines and tested by formal user studies.

Dekel and Herbsleb proposed to improve the effectiveness of online documentation in the IDE [13]. They introduced the *emoose* tool to highlight the more important pieces of text in JavaDoc and make them more salient to the programmers. Stylos et. al. built a tool called *Jadeite* to integrate API use frequency into Java documentation [14]. *Jadeite* also tries to make it easier to find 'misplaced' APIs [8] by means of the 'placeholder' feature.

A large number of tools have been investigated to help the programmers in finding API use examples. These include automatically generating code snippets based on a pair of input and output types given by the programmer [15], searching for API examples or applications, from private code repositories [16] [17] [18], or from the Internet [19] [20] [21] [22].

Programmers also encounter difficulty in understanding and adapting API-based examples [11] [9] [4]. Explainer is a tool designed to present information about an example using multiple views and perspectives [23]. The goal is to help the programmer develop good mental model of the example and transfer that knowledge to new tasks. More and better tools are needed to address these issues (Section VI).

III. Research Method

In this study, we followed the case study research process that Eisenhardt outlined in [5]. Our overall research goal was to identify a set of specific API usability obstacles and their causes. To reach this goal, we would need to analyze a sufficient amount of empirical data in order to obtain reliable conclusions. Since programmer newsgroups tend to contain numerous discussions about API use that are easy to obtain, they would naturally be a fertile field for the data that we needed for our study. Specifically, we chose the Swing Forum [1] as our data source. We focused on the Java Swing framework because it is mature, widely used, and reasonably well-documented.

[1] http://forums.oracle.com/forums/forum.jspa?forumID=950. All URLs verified on Feb. 10, 2011.

We manually collected and analyzed 172 newsgroup discussions from the Swing Forum. Each discussion typically consists of multiple messages. [2] The first few messages are often about the definitions and clarifications of the original poster (OP)'s task and goal. These tend to be followed by multiple comments, suggestions, and solutions posted by other forum participants. At the end, some OPs would explicitly confirm whether their problems have been solved, but many chose not to. Regardless, it was necessary for us to assess the states of each discussion independently, rather than being biased by words of the discussion participants, which were often proven inaccurate.

To ensure the quality of such a study, it is critical to fully understand the rich details contained in each collected newsgroup discussion. We carefully read through the messages and their textual description to fully understand OP's task, solution, and the difficulty they might have. In all cases, when code was posted, we tried to run the code. [3] This has been proven very helpful and in many times, even critical, for understanding OP's real problems as one often cannot describe accurately in words what he does not know.

It is also critical for us to understand the subject framework well enough in order to make sound judgments about the cases. Our main sources of documentation for Swing are Swing's on-line API JavaDoc and the Swing tutorial [24]. To ensure that our understanding of each framework feature be adequate and accurate, we consulted the online API documentation, occasionally Swing's source code, and a few books on Swing including the official Swing tutorial.

As a result, we have become intimately familiar with each case as a stand-alone entity. This process allowed us to accurately grasp the unique characteristics of each case. Specifically, for each discussion, we added one line of keywords to encode the key information contained in the discussion as well as a brief summary of its technical content. The following is an example of our encoding, where `CODE1` indicates that OP has posted code as part of his initial question, and `CODE2` indicates that participants have replied with code as well.

> DHOU: using a solution incorrectly (BorderLayout used, needs to remove a component before adding a new one; assumed adding a new one would automatically remove the old one) CODE1 CODE2

Consistent with what Eisenhardt pointed out in [5], this familiarity with cases also allowed us to iteratively encode each discussion along the various dimensions as we identified each of them. Coupled with within-case analysis was cross-case aggregation and search for patterns. Our rich familiarity with each case accelerated cross-case comparison. Finally, Unix utilities `grep` and `wc` were used to count the numbers reported in this paper.

[2] A total of 800 messages between Nov. 12, 2002 and Jan. 6, 2011.

[3] These codes are available in our data.

Table I
BREAKDOWN OF 172 SWING FORUM DISCUSSIONS ANALYZED AND SUBSECTIONS WHERE EACH CELL WILL BE DISCUSSED IN SECTION IV.

	calls	customization	custom tasks	understanding & others
solution seeking (A)	45 (A1)	36 (A2)	8 (A3)	
using wrong solution (B)	13	5		36
using solution wrongly (C)	28	1		16 (D)+20 (E)

IV. RESULTS

In this section, we present the results of our analysis of the 172 newsgroup discussions. Table I shows a breakdown of the overall results along two dimensions. We organize the data according to the *cognitive stages* where the OP was working at with their task and solution (the vertical dimension). Within each of these categories, the data are further broken down according to the categories of *task and solution* (the horizontal dimension).

A. Asking for a solution (89/172)

If no evidence can be detected in the messages of a discussion that the OP has actually attempted anything concretely before asking for a solution, we assign the discussion to this category, assuming that the OP's main goal is searching for a solution for a certain task. Note that this does not mean OP has not written any code. Quite to the contrary, many OPs in our data have provided code as contextual information to illustrate their needs for an additional feature. This category can be useful for detecting the various obstacles that prevent a programmer from successfully locating a solution. The hard-to-find solution occurs when the mapping from the API documentation to the desired solution is less obvious.

To help detect trends and patterns from the data more easily, we have divided the data into three subcategories: tasks whose solutions involve only compositions of API calls, tasks requiring the mutiple-steps customization of existing framework behavior, and custom design tasks that are mainly defined by problem-specific requirements. These subcategories are created to reflect the complexity and scope of the tasks and solutions involved in each forum discussion.

1) Tasks that require the composition of API calls (45/89): The easiest questions in this subcategory are probably ones whose answers consist of calling a single API method for achieving a certain task. Our data contain only a small number of such questions (four, to be precise). The following is an example discussion of this kind:

> Message Title: no border
> I have the background of my JTextField set to the same colour as the background of the JPanel. Is there a method call to set the border of the JTextField [...] so it blends into the JPanel?
> *Correct answer:*
> setBorder(null); or setBorder(new EmptyBorder(...));

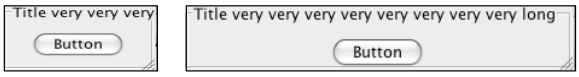
Other APIs asked are `Window.dispose()` (for disposing a window), `Frame.setIconImage()` (for setting an image icon on the title bar of a frame), and `JComponent.setFont()`.

Since the meaning of APIs should be, and many indeed are, self-evident from their names, it can be tempting to simply blame the OPs for laziness. It is certainly true that there are several cases where the questions could have been avoided if the OPs had searched and read the API documentation themselves. However, a closer look at all the cases in this subcategory reveals that not all such questions are totally unjustified.

For several reasons, programmers may still ask such questions even after reading the API documentation. They may not be sure about what value should be used for a parameter, they do not know enough about a certain aspect of the framework's design to fully understand the intent and function of an API (for instance, to understand `setUndecorated(true)`, one must know the notion of *frame decorations*), or the tasks and APIs may be less popular, unfamiliar to the OP, such as resizing an image. In these cases, the discussions often help clarify the situations.

When a programmer works on a feature that involve multiple classes, to find an API that implements a desired solution for customizing the feature, the programmer would have to search through not only the documentation for the primary classes, which he may perceive as being functionally more salient, but also those for the secondary helper classes. Thus, the programmer must know the design structure of the feature and all of its participating classes to locate an API.

Moreover, sometimes the location of a relevant API may be unexpected and surprising [8]. For example, the following discussion is about how to fully display the title of a titled border when the title text is longer than the JLabel that the border surrounds. The API



`TitledBorder.getMinimumSize(Component)` can be used to calculate the minimum size that the JLabel must have in order to fully display the title. Arguably, this API would have been more noticeable if it were positioned in the `JLabel` class rather than `TitledBorder`.

> Title: Providing Border for JLabel
> My JLabel is declared as
> ```
> Border inputborder
> = BorderFactory.createBevelBorder(...);
>
> inparam = new JLabel(" In Params:");
>
> inparam.setBorder(BorderFactory.
> createTitledBorder(inputborder,
> "INPUT PARAMETER MANUAL SETTINGS",...)));
> ```
> From the above code, [the border text ..] "INPUT PARAMETER MANUAL SETTINGS", is not fully visible since my JLabel is very a [short] text ("In Params:"). What should I do?
> *Correct answer:*
> ```
> border=BorderFactory.createTitledBorder
>            ("INPUT PARAMETER MANUAL SETTINGS");
>
> label.setBorder(border);
> // this is the key!
> borderSize = border.getMinimumSize(label);
> Insets insets = label.getInsets();
> label.setPreferredSize(new Dimension(
>    borderSize.width+insets.left+insets.right,
>    borderSize.height+insets.top+insets.bottom));
> ```

Other APIs that may have similar issues include `JRootPane.setDefaultButton()`, `JOptionPane(..., Object defaultValue)`, and `FlowLayout.setVGap()/setHGap()`.

Even if programmers have seen all the APIs relevant to their tasks and have read the API documentation, they still may not know immediately how to compose multiple API calls to form the final solution that they need. The number of meaningful compositions can be large. Without additional illustration or clarification, the distinction among the effects of all possible solutions may not be immediately obvious from reading the API documentation only. The following discussion may be used to illustrate this problem.

> Title: How to override setText for custom JButton
> [...] I have the code for a custom JButton. I would like to setText for this button so that the text is set over the image on the button. Would someone know how I can do this?
> code and several other posts elided
> *Correct answer:*
> Not exactly sure what you mean by *over*, but playing around with these two properties should give you what you want:
> ```
> setHorizontalTextPosition(JButton.CENTER);
> setVerticalTextPosition(JButton.CENTER);
> ```

In this case, it is likely that the OP has noticed the existence of two relevant APIs in the JavaDoc for the `AbstractButton` class: `setHorizontalTextPosition()` and `setVerticalTextPosition()`. What the OP might not have known is that there can be a total of nine relative positions that these two API methods can be combined to achieve, as shown in Figure 1, and that the first one would be what he needed. But the JavaDoc for `setHorizontalTextPosition()` shown below would not easily reveal these possibilities at all. This would have been the main reason triggering the original question.[4]

> **setHorizontalTextPosition**
> **public void setHorizontalTextPosition(int textPosition)**
> Sets the horizontal position of the text relative to the icon.
> **Parameters:**
> textPosition - one of the following values:
> SwingConstants.RIGHT
> SwingConstants.LEFT
> SwingConstants.CENTER

[4]The OP has written some code already. Thus he or she should be able to find these two API methods. Furthermore, asking the questions requires more effort than writing two lines of code to call these two methods.

SwingConstants.LEADING
SwingConstants.TRAILING (the default)
Throws:
IllegalArgumentException - if textPosition is not one of the legal values listed above.

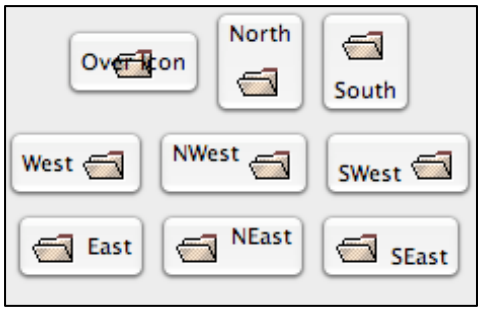


Figure 1. The nine relative positions that the text and icon in a button can have, which are achieved by combining three possible parameter values for each of the two API methods `setHorizontalTextPosition()` and `setVerticalTextPosition()` in the `AbstractButton` class.

Finally, there are also APIs whose functions are only inadequately, or not at all, documented. `UIManager.put("Component property name", value)` is a widely used API for setting widget properties such as fonts, colors, and borders. For some reason, this API method is not documented. As shown in Table III, this API has been mentioned *at least* [5] 364 times in the forum.

2) Customization tasks (36/89): These tasks are ones that require the programmer to write new code to customize an existing framework feature. Subclassing and method overriding are used in the new code.

Often the programmer needs to know enough about the internal design of the feature in order to successfully carry out a customization task. Thus, in addition to the obstacles identified in the last subsection, customization itself in general adds another factor for task complexity. This would be consistent with the relatively large number of cases in this subcategory. Furthermore, the numbers in the customization column in Table I would indicate that when facing with a customization task, a programmer is more likely to directly ask for a solution than trying out something concrete by himself first.

As illustrations, several customization tasks extracted from our data are given as follows:

- painting image icons or text differently to give a widget a non-standard look;
- customizing the size or look of a border;
- customizing the properties of the renderer or editor of a composite widget such as a ComoBox, a Tree, or a Table;
- customizing the title bar of a frame or a dialog;
- customizing the features of standard dialogs such as those made from a `JOptionPane`.

3) Custom design tasks (8/89): Custom design tasks are idiosyncratic tasks defined by domain specific requirements. Similar to customization tasks, custom design tasks also require the programmer to write new code. Unlike API composition tasks and customization tasks, which focus mainly on how to use a certain feature of the framework, discussions on custom design tasks can be more challenging and wide ranging in that they cover both requirements and solutions. The solutions for custom tasks tend to be larger in scope, often involving multiple tasks of the previous two subcategories. To illustrate, some custom design tasks from our data are given as follows:

- how to design the interaction behavior for a chess board;
- how to close all the other frames by clicking a button on one frame;
- how to move or resize a widget by means of mouse move or dragging;
- how to lay out the components of a specific user interface.

Interestingly, our data seem to indicate that there are not many custom design tasks discussed in the newsgroup.

B. Using a wrong solution (18/172)

OP has tried out a solution but the program does not work as expected. Instead, an alternate solution is needed. Sometimes, OP might have known that the solution used was wrong. But more often they did not.

Some APIs may share so much surface similarity that programmers may not be able to easily discern the roles of these APIs (we term this *API roles confusion*). As a result, they do not know when to use which. For example, in the discussion that follows, the OP indicates that he or she has (mis)used `setHorizontalAlignment()` to do what `setHorizontalTextPosition()` is supposed to do (highlighted in italic).

> Title: Problem about display both icon and text on a JButton!
> I want to put an icon on the upper part of the button, and some text on the bottom part of it. But when I use `setIcon()` and `setText()` methods of JButton, I found the text is in the right of the icon, this is not what I want! *I've tried some methods dealing with the text alignment, but it doesn't work*. How can I get what I want?
> Thanks! ...
>
> *Answer 1:*
> JButton extends AbstractButton. The method you require is found there. Check out the API `setHorizontalTextPosition(...)`.
>
> *Answer 2:*
> Hi,
> just use these two methods from class AbstractButton (JButton inherits them from there)
> - `setHorizontalTextPosition`
> - `setVerticalTextPosition`

[5]The current search interface for the forum appears to return only a subset of the matching discussions.

```
JButton button = new JButton();

button.setText(text);
button.setIcon(
         new ImageIcon(imageURL,altText));

button.setHorizontalTextPosition(
             SwingConstants.CENTER);
button.setVerticalTextPosition(
             SwingConstants.BOTTOM);
```

That's it!

In addition, we have seen in other discussions that programmers were also confused between `setAlignmentX()` and `setHorizontalAlignment()`. See Table III for data that show the number of times this confusion might have occurred in the forum. [6]

Other examples for this category include mistakenly using a mouse listener, rather than an action listener, on a button as well as using a `GridBagLayout` where a `BorderLayout` was in fact a better choice. In all such cases, the programmers had difficulty in choosing the right solution from multiple similar ones.

Confusion may also result from surface similarity between concepts from the task domain (*task concepts confusion*). An example is adding a decoration icon besides the text of a toggle button. A toggle button such as a Radio button and a Check Box generally has multiple states, including *selected* and *unselected*, *mouse over*, and *disabled*. There is one icon to indicate each of these states. For example, the default icon for the selected state for the radio button is the circle with a dot in the middle, and for the checkBox the check. These icons can be replaced by calling `setIcon()`, `setSelectedIcon()`, `setRolloverIcon()`, and so on. It is possible to add an additional icon besides the button text as decoration. Many programmers confused this decoration icon with the other standard icons and chose to use the APIs for the other icons to add the decoration icon.

Sometimes, a seemingly easier but improper solution might be chosen because the right one was not obvious or too hard to use. Furthermore, a wrong solution could also be used because the programmer was unaware of some platform-specific idiosyncrasies. For example, a custom made `ListCellRenderer` for a combo box worked well on Mac and Linux, but not for Windows Vista. Thus, the programmer must learn a special way to make it work for Windows Vista.

Finally, the default solution from the framework may not be what is needed. When the programmers are unaware of this, they end up with a wrong solution. For example, a common mistake many made was adding multiple widgets to the content pane of a frame but without setting a new layout manager. As a result, they were surprised to see that only the last widget added was displayed. This is because the default content pane of a frame by default uses the `BorderLayout`, which allows only one widget at center.

[6]Due to the scale of the set of the matching discussions, we have not inspected all of them to be sure that they all contain this kind of confusion. However, we did find this was true for the small subset we inspected. Moreover, this set of API methods are very specific. The probability that they appear in a discussion for any other reason, is very low.

C. Using a solution incorrectly (29/172)

OP has tried out a generally right solution but the program does not work as expected due to mistakes in carrying out some of the solution steps. Obviously, one cause for this is implementation bugs. However, this cause constitutes only a small percent in our data. There are several other reasons.

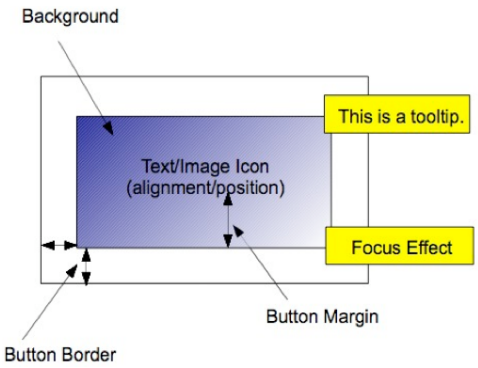


Figure 2. The visual elements of a button.

First, the programmer may be unaware of the subtle dependency between two APIs, or the effect of a precondition on the API. One example for the former is the pair of API methods `JComponent.setBorder()` and `AbstractButton.setMargin()`. The intention of the original design appears to be to provide two ways to control the marginal space between the main content area of a widget and its border. One way is by setting a custom border, and the other is by setting the margin distances from the left, top, right, and bottom sides. (The concepts of margin and border, along with several other visual properties, are illustrated in Figure 2 using a button.) But these two methods should not be used at the same time for the same widget. In particular, the margin approach can only be used when the default border is still in effect rather than a custom border. However, it appears that the JavaDoc for `setMargin` does not make this subtlety clearly enough:

> **setMargin**
> **public void setMargin(Insets m)**
> Sets space for margin between the button's border and the label. Setting to null will cause the button to use the default margin. The button's default Border object will use this value to create the proper margin. However, if a non-default border is set on the button, it is that Border object's responsibility to create the appropriate margin space (else this property will effectively be ignored).
> **Parameters:**
> m - the space between the border and the label.

The following discussion illustrates this confusion:

Title: JTextField with custom border and margin insets

I created a class which extends JTextField.
In the constructor, I want this class to have a line border with whatever color and a left inset of 10. However, setting a custom border ignores the setMargin(). If I remove the border line, the margin shows up fine. Is there a way to accomplish both?
This is what I thought should work:

```
LineBorder border =
   new LineBorder(new Color(100,100,100),1));
this.setBorder(border);
this.setMargin(new Insets(0,10,0,0));
```

What am I doing wrong? =)

Correct answer:

Use a CompoundBorder consisting of a LineBorder and an EmptyBorder.

Other interesting examples in this category include calling `getRootPane()` inside of the constructor of a `JFrame` subclass, and setting focus before a widget is displayed. In both cases, important preconditions for these APIs were violated. As a result, programmers failed to see the effects they expected.

The second reason in this category is executing a multiple-steps plan wrongly or incompletely (incomplete API sequences). One example of this kind is shown below, where the OP mistakenly used `setVerticalAlignment()`, which is unrelated to his task.

Title: JCheckBox vertical alignment of label

I am trying to get a JCheckBox to display its label below the checkbox. I have tried:

```
JCheckBox.setVerticalAlignment(
                      JCheckBox.BOTTOM);
JCheckBox.setVerticalTextPosition(
                      JCheckBox.BOTTOM);
```

...but no luck...what am I doing wrong?

Correct answer:

```
checkBox.setHorizontalTextPosition(
                   SwingConstants.CENTER);
checkBox.setVerticalTextPosition(
                   JCheckBox.BOTTOM);
```

Another common mistake of this kind happens when replacing the center widget on a panel that uses a `BorderLayout`. Some programmers added the replacement widget to the same location, assuming that its addition will automatically remove the previous one. Unfortunately, the original design requires the explicit removal of the previous widget instead. Yet another example is related to the issue of setting icons for the toggle buttons we discussed before. To be complete, a programmer must supply at least two icons, one for selected, and the other for un-selected, for a toggle button. Some supplied only one, typically by calling *setIcon()*, and were surprised to see that the check was not there any more for a checkBox.

Third, the semantics of certain APIs are not clearly documented and, thus, can be confusing to programmers. A well-known example of this is the issue of what is *the proper way* of calling `JComponent.revalidate()` and `Component.repaint()` after the content of a `JPanel` is changed. There were frequent, lengthy discussions of these APIs in the newsgroup [7] [8] [9]. Many programmers experienced problems when they missed the calls to these API methods.

Sometimes, a mistake was made by supplying the wrong parameter values to certain APIs. In some cases, this was because the programmer was unaware of the special constraints that the framework imposes, which, surprisingly, made the otherwise acceptable parameter values illegal. For example, for some reason, the framework requires a font object of `FontUIResource` type, rather than `Font`, to be set as the value of its default font property.

Finally, some problems were caused by incorrect runtime environment configurations. For example, icons were put in a wrong location, or a font existed in the development environment but not in the final deployment platform.

D. Understanding (16/172)

There were 16 cases whose main goals were not seeking for a solution but gaining a better understanding of something. In 11 cases, the discussions were about understanding why/why not a behavior happened in OP's programs. Some of these were due to misconceptions about the framework used, or improper expectations about program behavior. Others were due to platform specific behavior, or simply implementation bugs.

The other five cases were about understanding how API methods work. In one case, a lengthy discussion was conducted on the impact that `Window.dispose()` could have on memory leak for an embedded system. In three other cases, participants elaborated on the unclear semantics of a pair of API methods and shared experiences with each other. One other case discussed a particular concept.

E. Other situations (20/172)

There were also nine tasks for which no solutions were provided by the forum participants, although we judged that solutions would be feasible. Four other tasks were judged for which no solutions could be found. This usually happened when the task requirements broke the fundamental assumptions that the framework built on.

Finally, tasks for four discussions were judged to be unclear or undefined, and three tasks were unrelated to APIs.

F. Summary of identified API learning obstacles

Table II summarizes the set of obstacles discussed in the previous subsections.

[7] http://forums.oracle.com/forums/thread.jspa?messageID=5882657
[8] http://forums.oracle.com/forums/thread.jspa?messageID=5763908
[9] http://forums.oracle.com/forums/thread.jspa?messageID=5849774

Table II
API OBSTACLES IDENTIFIED IN THIS STUDY.

Sections	# Obstacles
IV-A	Clarification needed for API use or meaning.
	Less used, unfamiliar APIs.
Asking	Misplaced, hard-to-find APIs.
for a	Hard-to-distinguish solutions.
solution	Undocumented APIs.
	Difficulty with multiple-steps customization tasks.
IV-B	API roles confusion.
Using a	Task concepts confusion.
wrong	Deceived by easier solutions.
solution	Improper default solutions.
IV-C	Subtle API precondition or dependency.
	Improper execution of multiple-steps solutions.
Using a	Unclear API semantics by design.
solution	Wrong parameter values.
incorrectly	Wrong environment configurations.

V. ADDITIONAL OBSERVATIONS

We found that the final solutions for some of our case data were not officially documented (38 cases). 26 of these cases are ones whose solutions require the customization of framework behavior. There are also nine cases whose solutions are platform-specific. In addition, four more cases were also related to platform-specific behavior, but their goal was to *understand* the platform-specific behavior rather than finding a solution.

There are two main reasons why the platform-specific cases were not documented. One is that some cases require the customization of aspects of the underlying framework, which are not yet published as APIs. The other is due to framework evolution. When a framework evolves, previous ways of using it may become outdated and need to be replaced by new ways.

We have also collected evidence that novice programmers may need help to characterize their problems verbally. We found that in 22 cases, OPs were not able to clearly describe what they wanted and were asked to clarify their task descriptions. In many cases, OPs had difficulty in communicating with other participants verbally about API use problems.

When verbal communication became a problem, source code has proven to be the most effective alternative way of communication. Once an OP provided code, other participants could assess and comment on the code, ask questions about it, point out potential bugs, and offer solutions. In fact, communication using code has been so popular that the online forum community has even established a convention for posting code, SSCCE (a Short, Self Contained, Compilable Example) [10]. The goal for using the SSCCE convention is to improve the efficiency and effectiveness of forum communication using code. Overall, in our data, there were 92 cases where OPs used code as background when they asked their questions initially. There were 77 cases where other participants replied with code.

We have observed that there are several other ways a programmer may benefit from using the newsgroup. We have seen participants pointed out links to external information sources (9 cases), offered multiple suggestions for a problem (8 cases), compared and contrasted candidate solutions (5 cases), and inspected the OP's code for bugs and provided detailed programming helps (18 cases).

VI. IMPLICATIONS FOR NEW TOOLS AND PRACTICES

The Internet has been used as a medium for sharing code and seeking code-specific help [25] [26], and many recent work have been directed towards finding relevant code and documentation on the web [19] [20] [26] [21] [22]. But the Internet still has important limitations. One limitation is the vocabulary problem [27] [1]. Search engines index literal words rather than code semantics, making it hard to specify queries for code in domain terms that are more meaningful to users. Furthermore, when searching for more sophisticated solutions that appear on multiple pages, programmers also have to spend significant efforts in organizing the searching process and keeping track of the intermediate results [26]. Finally, to properly use an API, a programmer must know not only what can be used and what actually goes wrong, but also understand how and why it must be used in a certain way [1]. While searching can be a good starting point for obtaining relevant code, programmers still have to face significant challenges in judging the relevance of the returned results, or in understanding them in order to modify them successfully [1]. Many programmers thus post questions to online forums, but answers may have high latency or their questions may not be answered at all, which is also shown in our data.

Hence, additional research are needed to investigate ways to reducing, or eliminating, the latency and efforts in searching for and adapting a solution. The overall goal would be to increase the usability and accessibility of API-related information.

In particular, Integrated Development Environments (IDEs) could be significantly empowered to help programmers understand the interaction between their code and the APIs used or to be used. By using IDEs, pertinent information could be automatically brought to the programmers rather than forcing them to leave their working context to search for relevant information. Furthermore, because the provided information is attached to their code, it would save the programmers from manually associating them with code. Instead, programmers can start immediately to read and understand the documentation along with the code.

The data from this study appear to provide supporting evidence that justifies the investigation of this kind of tools. Specifically, majority of the programmers in our study were working with code when they had a question (Section V).

[10]http://sscce.org/

This indicates that such tools could have been applied to their code to help them. Furthermore, there were 22 cases where task clarifications were sought from the OPs, and 9 cases OPs could not describe their problems clearly at all (Section V). Such tools would be especially helpful to such novices because with the tools, the novices would be able to avoid the task of formulating queries but still obtain relevant information. Finally, to ensure that the problems we observed from our data were not isolated special cases, we searched the whole forum for occurrences of these APIs. The data shown in Table III indicate that these APIs are indeed frequently discussed in the forum. Thus, supporting them in the IDE would benefit many programmers.

There seem to be three problems with documentation. Some solutions were undocumented. For many other cases, we found that the documentation did contain the needed information, but it appeared that programmers had difficulty in accessing the solutions. This is highly likely because the needed information was either embedded deeply in the lengthy text and code in the documentation, or dispersed in multiple locations. Finally, additional documentation and tutorials are needed to elaborate on the design and use of some of the more subtle features.

In addition to the state of the art tools such as [13] [14], better tools would be needed to further improve the usability of documentation. At the core of such tools would be a set of condition-action pairs of the form "*If certain features appear in code, display pertinent information*." If the features for which the programmers need help are secondary in the sense they become relevant only when a main feature appears, the main feature could be used as a hook to decide whether information about the secondary features need to be pushed to the programmers. (This was true for many cases in our data.) For example, a clarification documentation for `setBorder` and `setMargin` (Section IV-C) would be needed if and only if a button-like widget appears in the code. This mechanism could be elaborated to provide documentation for the undocumented cases, cases that require clarification documentation, and the step-by-step instructions for customization tasks.

In general, such tools could offer explanations, recommendations, and criticisms for API-based code. Based on features detected in code, the tools could offer documentation explaining how the detected features work, or contrast the features with other competing solutions in context. Other related solutions that might be used together with the detected features could be recommended for consideration. For cases where a wrong solution was used (Section IV-B), the tools could operate by identifying the common ways where one solution is misused for another. Finally, cases in the category of "using a solution incorrectly" (Section IV-C) could be helped with critics that detect missing steps or failing preconditions from API client code.

There can be multiple options for authoring and publishing documentation for such tools. The documentation could be authored by the original framework designers or by others but reviewed by the designers for quality control. A channel could be established to automatically disseminate these documentation and rules to the programmers' IDEs whenever they become available.

Table III
A SUBSET OF "PROBLEMATIC" APIS IDENTIFIED FROM THE DATA OF THIS STUDY AND THE NUMBERS OF TIMES THEY APPEARED IN THE SWING FORUM.

APIs as Query Keywords	# Matches
`setDefaultButton`	151
`setUndecorated`	362
`UIManager.put`	364
`setHorizontalTextPosition` OR `setVerticalTextPosition`	229
`setHorizontalAlignment` OR `setAlignmentX`	369
`setHorizontalTextPosition` && `setHorizontalAlignment`	60
`setHorizontalAlignment` && `setAlignmentX`	62
`setBorder` && `setMargin`	111
`revalidate` && `repaint`	381
`TitledBorder` && `getMinimumSize`	5

Much more work needs to be (and can be) done to fully take advantage of the results reported by a study like ours. Currently, we are designing IDE tools to address some of the obstacles discussed in this paper [28].

VII. THREATS TO VALIDITY

This was an exploratory study. Our goal was not to collect quantitative data to show the frequency or predominance of specific trends or testing statistical significance. Instead, our goal was to collect qualitative data to identify and characterize the major categories of obstacles in learning and using APIs.

Due to our extensive past experience with teaching and researching with Swing [12] [11], we were relatively confident that our knowledge of this particular framework would be sufficient for helping ensure the construct and internal validity of this study. In our approach of analyzing newsgroup discussions, sometimes we made inferences about forum participants' cognitive status. [11] A limitation of this approach would be that we cannot observe or interview the participants in person to verify that our inferences about their cognitive status are accurate. However, we believe that our experience with programming and the subject framework has helped in sufficiently mitigating this risk and ensuring the overall validity of our results.

The external validity of our results can be improved by replicating them on other frameworks and APIs. With more data and more experience with other APIs, new categories

[11] In retrospect, we believe that this practice in itself is acceptable as long as enough care is taken to ensure the validity of the inferences.

of obstacles may become apparent. However, as discussed in Section II, we compared our set of obstacles with those of related work such as [3] [9]. We found that overall the results corroborated each other. This indicates that our results do have some degree of generality and that overall, our results should be representative.

VIII. Conclusion

In this paper, we reported an exploratory study where we analyzed in detail 172 programmer discussions collected from the Swing Forum. From the analysis data, we identified a set of API learning obstacles. We described in detail these obstacles and their probable causes. We also discussed the implications of these obstacles on building better tools to help increase the accessibility of APIs and documentation.

Future work should be directed to refining and expanding the list of obstacles as well as investigating tools to address these obstacles. More efforts are needed to accurately identify and characterize individual categories of API usage problems so that tools can be built and evaluated for each of them. Given the potentially large scope of solutions and tools needed to be investigated, we anticipate that many rounds of iterative development are required. However, we feel that the results reported in this paper have made a positive first step towards that goal. Currently, we are building such tools [28].

Acknowledgment

The authors wish to thank the four anonymous reviewers for their constructive feedback on this paper. This research was partially supported by an IBM UIMA Innovation Award.

References

[1] G. Fischer, "Cognitive View of Reuse and Redesign," *IEEE Software*, vol. 4, no. 4, pp. 60–72, 1987.

[2] S. G. McLellan, A. W. Roesler, J. T. Tempest, and C. I. Spinuzzi, "Building More Usable APIs," *IEEE Softw.*, vol. 15, no. 3, pp. 78–86, 1998.

[3] 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.

[4] M. Robillard and R. DeLine, "A Field Study of API Learning Obstacles," *Empirical Software Engineering*, 2011, to appear.

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

[6] J. Stylos and S. Clarke, "Usability Implications of Requiring Parameters in Objects' Constructors," in *ICSE '07*, 2007, pp. 529–539.

[7] B. Ellis, J. Stylos, and B. Myers, "The Factory Pattern in API Design: A Usability Evaluation," in *ICSE '07*, 2007, pp. 302–312.

[8] J. Stylos and B. A. Myers, "The Implications of Method Placement on API Learnability," in *SIGSOFT '08/FSE-16*, 2008, pp. 105–112.

[9] M. Robillard, "What Makes APIs Hard to Learn? The Answers of Developers," *IEEE Software*, November/December 2009.

[10] L. Li, "Towards Understanding Programmers' Information Needs in Using Frameworks and APIs," Master's thesis, Clarkson University, Potsdam, New York, USA, June 2010.

[11] D. Hou, "Investigating the Effects of Framework Design Knowledge in Example-based Framework Learning," in *ICSM '08*, 2008, pp. 37–46.

[12] D. Hou, K. Wong, and H. J. Hoover, "What Can Programmer Questions Tell Us About Frameworks?" in *IWPC '05*, 2005, pp. 87–96.

[13] U. Dekel and J. D. Herbsleb, "Improving API Documentation Usability with Knowledge Pushing," in *ICSE '09*, 2009, pp. 320–330.

[14] J. Stylos, B. A. Myers, and Z. Yang, "Jadeite: Improving API Documentation using Usage Information," in *CHI '09*, 2009, pp. 4429–4434.

[15] D. Mandelin, L. Xu, R. Bodík, and D. Kimelman, "Jungloid Mining: Helping to Navigate the API Jungle," *SIGPLAN Not.*, vol. 40, no. 6, pp. 48–61, 2005.

[16] Y. Ye and G. Fischer, "Supporting Reuse by Delivering Task-Relevant and Personalized Information," in *ICSE*, 2002, pp. 513–523.

[17] N. Sahavechaphan and K. Claypool, "XSnippet: Mining For Sample Code," *SIGPLAN Not.*, vol. 41, no. 10, pp. 413–430, 2006.

[18] R. Holmes, R. J. Walker, and G. C. Murphy, "Approximate Structural Context Matching: An Approach to Recommend Relevant Examples," *IEEE Transactions on Software Engineering*, vol. 32, no. 12, pp. 952–970, 2006.

[19] S. Thummalapenta and T. Xie, "Parseweb: A Programmer Assistant for Reusing Open Source Code on the Web," in *ASE '07*, 2007, pp. 204–213.

[20] J. Stylos and B. A. Myers, "Mica: A Web-Search Tool for Finding API Components and Examples," in *VLHCC '06*, 2006, pp. 195–202.

[21] J. Brandt, M. Dontcheva, M. Weskamp, and S. R. Klemmer, "Example-Centric Programming: Integrating Web Search into the Development Environment," in *CHI '10*, 2010, pp. 513–522.

[22] M. Grechanik, C. Fu, Q. Xie, C. McMillan, D. Poshyvanyk, and C. Cumby, "A Search Engine for Finding Highly Relevant Applications," in *ICSE '10*, 2010, pp. 475–484.

[23] D. F. Redmiles, "Reducing the Variability of Programmers' Performance through Explained Examples," in *CHI '93*, 1993, pp. 67–73.

[24] K. Walrath, M. Campione, A. Huml, and S. Zakhour, *The JFC Swing Tutorial (Second Edition)*. Addison Wesley, February 2004.

[25] J. Brandt, P. J. Guo, J. Lewenstein, M. Dontcheva, and S. R. Klemmer, "Two Studies of Opportunistic Programming: Interleaving Web Foraging, Learning, and Writing Code," in *CHI '09*, 2009, pp. 1589–1598.

[26] R. Hoffmann, J. Fogarty, and D. S. Weld, "Assieme: Finding and Leveraging Implicit References in a Web Search Interface for Programmers," in *UIST '07*, 2007, pp. 13–22.

[27] G. W. Furnas, T. K. Landauer, L. M. Gomez, and S. T. Dumais, "The Vocabulary Problem in Human-System Communication," *Commun. ACM*, vol. 30, no. 11, pp. 964–971, 1987.

[28] C. R. Rupakheti and D. Hou, "Satisfying Programmers' Information Needs in API-based Programming," in *ICPC '11*, 2011, 4 pp. to appear.