Usability Criteria for Graphical Code Completion

Cyrus Omar
Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213
cyrus@cmu.edu

YoungSeok Yoon
Institute for Software Research
Carnegie Mellon University
Pittsburgh, PA 15213
youngseok@cs.cmu.edu

I. Introduction

Many source code editors today feature interactive code completion. By invoking a keyboard shortcut, developers can summon a floating palette containing a list of contextually relevant variables, fields, methods, classes and code snippets. An adjacent floating window displays documentation for the currently highlighted item. This interface helps developers explore APIs and avoid basic spelling and logic errors [?], and empirical data indicates that professional developers invoke interactive code completion frequently [1].

A number of new to the code completion list have been proposed in the literature. These methods combine information about the program context with make use of usage history [?], examples extracted from code repositories [?, ?] and crowdsourced information [?]

Calcite, reordering, crowdsourced code completion papers.

Our proposal: graphical code completion. Inversion of control: look at class. Also possible to look at external database. Visual languages have shown this might be useful in certain circumstances (?).

Consistent with best practices, we begin by seeking the usability criteria that govern whether such a system would be useful to professional developers. We conducted a large survey of these developers and extracted several criteria that both constrain the overall system architecture and inform the design (or caution against designing) palettes for different kinds of classes.

II. Methods

Our subject pool consisted primarily of visitors to a large collaborative filtering website dedicated to programming called `/r/programming` on reddit.com[1]. At the time of our study, there were approximately 340,000 registered members. A small number of participants were also recruited using a mass email to local computer science graduate students. The recruitment materials in both cases stated that we were seeking developers "familiar with an object-oriented programming language like Java, C# or Visual Basic and an integrated development environment like Eclipse or Visual Studio".

Participants were asked to take an online survey that would take approximately 20 minutes to complete. Of the 696 participants who started the survey, 475 participants finished it. Of these, 457 participants came from /r/programming and 18 from the graduate student mailing list. All quantitative metrics used in this study are based only on these participants.

[1] http://www.reddit.com/r/programming

A. Familiarity with Languages and Editors

We began by asking participants to rate their level of familiarity with several programming languages. Figure 1 summarizes these ratings. On average, participants rated themselves as very familiar with Java, C, C++ and JavaScript, familiar with C#, Python and PHP and somewhat familiar with Visual Basic and Perl.

We also asked participants to select the integrated development environments and code editors that they were familiar with. The Eclipse IDE was familiar to 87.1% of participants. This was followed by Visual Studio with 66.0% of participants, Vi/Vim with 53.7%, Netbeans with 37.7%, Emacs with 24.8% and IntelliJ IDEA with 16.4%. Participants could also enter "other" choices. A number of editors and IDEs were entered, including Xcode, Textmate, Notepad++ and others.

B. Mockups

Next, we presented participants with a series of mockup palettes corresponding to a Color class, a regular expression Pattern class (Figure 2) and a SQL query ResultSet class. In addition to a screenshot of the palette itself, participants were shown mockups demonstrating how a user would invoke the palette, and a mockup showing the code that would be inserted once a selection had been made.

Before presenting each palette mockup we gathered information about the user's familiarity with that topic and the strategies that they would likely use to instantiate that class. The Color class was presumed familiar to all participants. Figure 3 summarizes responses for the regular expression and SQL classes. The low number of participants unfamiliar with these topics again provides evidence that our participants were not generally novice developers.

To determine the strategy typically used for instantiating the Color class, we asked the following question. The values in bold show the frequency of each response.

> A user interface designer asks you to change the background color of a window to "navy blue" when the user presses a button. You have determined that you need to instantiate the 'Color' class to do so.

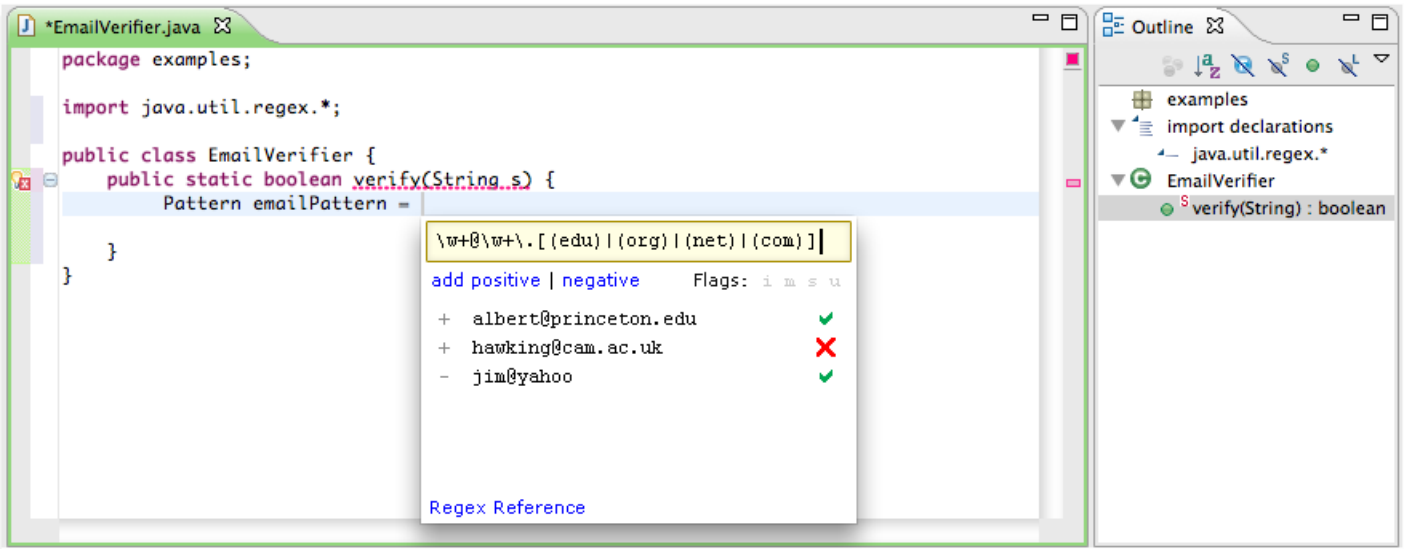

Fig. 1. The regular expression palette, shown in Eclipse.

	Regular Expressions	SQL
Never used	4.8%	0.0%
Use infrequently	46.7%	37.4%
Use frequently	**48.4%**	**62.6%**

Fig. 2. Participant's experience with regular expressions and SQL.

Which of the following strategies would you use first to create an instance of the 'Color' class representing navy blue?

(a) I would use the IDE's content assist menu (see note below) to look for a predefined constant representing navy blue. (**58.4%**)
(b) I would look up the documentation of the Color class to see if there is a predefined constant representing navy blue. (**19.0%**)
(c) I would use an external tool (e.g. Photoshop or some other image editor) to determine the RGB values for navy blue. (**14.0%**)
(d) I would guess at the RGB values for navy blue, then check whether my guess was correct by executing the application. (**5.3%**)
(e) Other (please specify) (**3.3%**)

Figure 4 summarizes the responses to a more general question asked about regular expressions and SQL queries.

After presenting each palette, we asked users to rate the usefulness of each palette. Responses are summarized in Figure 5. We then allowed participants to provide qualifications and open-ended feedback by asking the following question: "If you would like to qualify or elaborate on your answer to the previous question you may do so below. You may also suggest improvements for this palette." This information is analyzed in the remainder of the paper.

	Regular Expressions	SQL
Separate test script	29.7%	15.5%
Guess and check	13.4%	16.0%
External tool	**38.5%**	**58.9%**
Search for examples	12.1%	4.8%
Other	6.2%	4.8%

Fig. 3. Typical strategies for regular expressions and SQL queries.

CLASS	Nearly every time	Most of the time	Some of the time	Rarely	Never
Color	9.6%	22.1%	**32.4%**	28.2%	7.7%
RegExp	**36.6%**	29.5%	21.8%	7.3%	4.8%
SQL	18.2%	19.3%	**30.9%**	20.4%	11.4%

Fig. 4. For each of the classes considered, participants were shown mockups of a code completion palette designed for instantiating that class, as well as the source code generated after parameters were entered. This table shows the distribution of responses to the question: "Consider situations where you need to instantiate the [specified] class. What portion of the time, in these situations, do you think you would use this feature?"

III. Design Implications

We next analyzed the open-ended responses to extract some design principles and constraints. We got 193 responses for the color palette, 129 for the regular expression palette, and 142 for the SQL palette. Also, we got total 119 suggestions about what other types of classes could benefit from similar types of palettes. The design can be divided into two main categories: system design constraints, and general palette design considerations. System design constraints are the criteria that should be taken into account in the design of the overall system architecture, and the general palette design considerations are about how to make more usable palettes.

A. System Design Constraints

1) Handling Separation of Concerns: Professional developers were very wary of including constant data into the source code directly. Many responses expressed general sentiment or specific preference for project-wide color theme or external resource file. Also, many people stated that these tools might not be useful since they usually read an external resource file, or sometimes they explicitly suggested to add capability to handle the external resource files directly from the palette.

Examples of feedbacks in this category include:

- The use of magic numbers in the source code is a bad design (54)
- Color picking is the 'designer's job' (33)
- Would only use to define named constants (17)
- Would rather use color scheme / theme (3)
- Would use Object-Relational Mapping (ORM) / Language INtegrated Query (LINQ) instead of raw SQL query string (27 of 142)
- These palettes might be useful for prototyping, but not for real production (14)
- Instead of including the test cases into comments, they should be stored as unit tests (35 of 143)

Interestingly, most people seemed to consider regular expressions as part of the program logic. There were few people who complained about including the regular expression pattern string in the source code.

2) Reversibility: 19 participants expressed that it should be able to bring the palettes back when they want to modify the parameters after they used the palettes.

3) Palette Settings and State: Many participants wanted to be able to configure the palettes or have the palettes maintain state even after the palettes are hidden. The followings are the comments that are related to this category.

- Want recently used regular expressions, and control over the comments (12)
- Persistent database connection information is needed / want to manage connection pools (9)
- Want recent colors, capability of setting favorite colors (20)

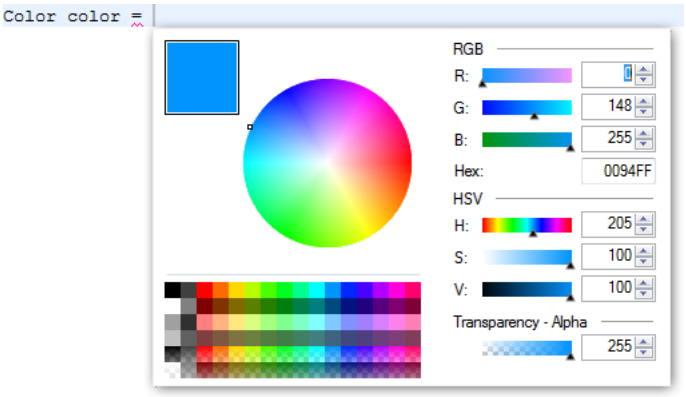

Fig. 5. test

4) Interaction with Code Context: Some people wanted to interact with the code context. For example, 13 people expressed that it would be great if the SQL query palette was capable of dynamically creating SQL query string by combining multiple string variables, or by assigning variables for the parameters.

5) Performance: Developers were concerned about the performance of these palettes. Some of the participants said that they would use these palettes only if it does not slow down the IDE. Also, this was the most popular comment on the reddit thread.

6) IDE Independence: Several participants expressed a desire for IDE or even language independence for these palettes.

B. General Palette Design Considerations

1) Simplicity vs. Capability: Even though the participants were shown the same mockup screenshots, their reactions were split. Our color palette was considered too complex by many participants (26), and many others (26) would be satisfied with seeing colors next to a list of color names. The regular expressoin palette and SQL palette were considered too simple (12, and 15 participants respectively). They wanted syntax highlighting, match highlighting, and browsing capability of SQL databases.

2) Keyboard Navigability: 12 of the participants expressed strong antipathy toward the mouse. This was mostly directed at the color palette.

3) Complexity:

IV. Suggestions from the Participants

We classified the suggestions into several categories. The number in the parentheses indicate the number of participants who suggested the classes which fall into the category.

A. Alternative/Tricky/Literal Syntax (16)

These are the classes of which syntax is tricky, or complex. Even for a basic string class, people wanted to input multiline string or unicode string in more convenient way with automatic escaping.

- Example classes
  - String
  - Collections (e.g., dictionary)
  - Vector, Matrix
  - Embedded languages
  - URLs, Paths
- Expected features
  - Proper escaping
  - Syntax highlighting
  - Auto-completion

B. Unclear Parameter Implications (11)

The classes in this category has some parameters in it, and it is not easy to predict the results of the parameter modifications. Instead of running the whole application to see the result, the palette can serve as a preview panel so that the developers may

check the results directly. Also, the palette can be a control panel which facilitates modifying the parameters.

- Example classes
  - Audio tweaks (modifying pitch, volume, etc.)
  - 3D transformation matrices
  - number / string / date formatting
  - message box / input box (e.g., `JOptionPane`)

C. Query Languages (17)

Similar to the unclear parameter implications category, the developers wanted to check if the query string is correct or not by checking the results directly on the palette.

- Example classes
  - Regular expression
  - SQL query
  - XPath / XQuery

D. Graphical Elements (27)

The other popular category was graphical elements. We've already shown the color example, and the participants suggested many other graphical properties such as font, brush, shape and others. Also, people wanted to check or directly manipulate the GUI frames and layouts.

- Example classes
  - Color, Shape, Brush, Font, etc.
  - JFrame, Layouts, any swing components

E. Complex Instantiation Procedures (5)

Some of the participants pointed out that these palettes might be useful to create an object which requires complex instantiation code. For example, in order to read a text file in Java, the developer might want to use `BufferedReader` class. The instantiation code might look as following:

```
BufferedReader reader = null;
try {
    reader = new BufferedReader(
        new FileReader(filePath));
} catch (IOException e) {
    e.printStackTrace();
}
// use reader here
reader.close();
```

By using a palette to choose a file or choose a variable which contains the file path, the developers could easily instantiate these objects. In the same way, it can alleviate the factory pattern usability problem. As long as the developers remember which class to use, they will not need to remember how to instantiate that class.

- Example classes
  - BufferedReader
  - Classes using factory patterns
  - Database connection string builder

F. Describe by Example (2)

Sometimes, it is possible to describe an object by examples. For instance, if there is a class which represents a shortcut key combination, we can easily instantiate an object of that class by pressing the actual shortcut key on the interactive palette.

- Example classes
  - Keyboard Keys
  - Regular expression

G. Integrating with Documentation, Tutorial (7)

Some participants suggested integrating the documentation or the tutorials into the interactive palettes. If the developers are not familiar of the specific API class, the palette could tell them how to use it.

V. Conclusion

Overall, professional developers appear willing to use these for a number of use cases. However, the usability criteria described above impose stiff requirements on the architecture. Of particular concern, we found that the developers preferred help with program logic, but not the constant data. This indicates that another area of application may be in specialized editors for resource files.

References

[1] G. C. Murphy, M. Kersten, and L. Findlater, "How are java software developers using the eclipse IDE?" *IEEE Software*, vol. 23, no. 4, pp. 76–83, 2006.