I. Developer Survey

Prior to implementation, we conducted a large online survey of professional software developers in order to extract criteria that govern whether our system would be useful to them. More specifically, we sought to address the following questions:

- What are specific use cases for active code completion in a professional setting?
- What are some functional criteria that predict whether a class would benefit from active code completion?
- What are some usability criteria that should govern palette interface designs?
- Which capabilities must a active code completion architecture support to enable these use cases and designs?

We posted a link to our survey on a popular collaborative filtering website [**?**] called `/r/programming` on reddit.com[1]. At the time of our study, the website listed approximately 340,000, though members were not required to registered to read the content, and contained content that could be considered relevant to our intended audience. An additional 22 participants were 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 people who started the survey by answering the first question, 473 participants completed it. Of these, 16 participants were computer science graduate students at Carnegie Mellon and 457 were from `/r/programming`.

Their responses revealed a number of interesting use cases and non-trivial design constraints for active code completion systems. Many of these findings can also be generalized to constrain the design of visual programming languages, particularly those that aim be useful to professional developers, rather than novice programmers (e.g., Barista [**?**]). We summarize the survey questions and the results in the following subsections.

A. Familiarity with Programming Languages and Editors

We first asked their level of familiarity with several programming languages, on a five-point Likert scale ranging from "None" to "Expert". 61.1% of the respondents expressed themselves as experts on at least one language, and 35.7% were very familiar with at least one language. 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 (IDEs) and code editors that they were familiar with. The Eclipse IDE was familiar to 87.1% of participants. This was followed by Visual Studio at 66.0%, Vi/Vim at 53.7%, Netbeans at 37.7%, Emacs at 24.8% and IntelliJ IDEA at 16.4%. Participants could also enter "other" choices.

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

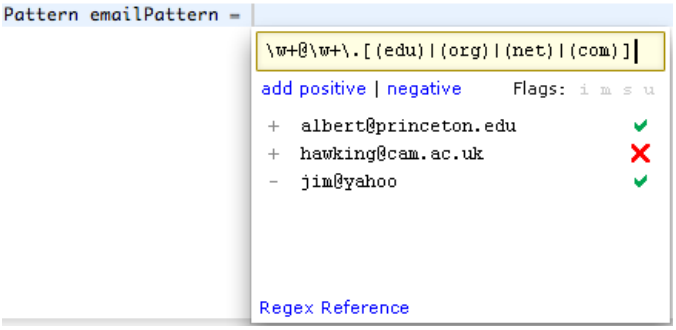

Fig. 1. An example mock-up palette for regular expressions shown to the survey participants. In the survey, they were shown the surrounding Eclipse IDE together.

	Regular Expressions	SQL
Separate test script	29.6%	15.4%
Guess and check	14.0%	16.1%
External tool	**37.9%**	**58.6%**
Search for examples	12.3%	5.1%
Other	6.2%	4.9%

Fig. 2. Typical strategies for regular expressions and SQL queries.

A number of editors and IDEs were entered, including Xcode, Textmate and Notepad++. None of these were dominant.

B. Mock-ups

Next, we presented participants with a series of mock-up palettes for a Color class, a regular expression class (Fig. **??**), and a SQL query class. Participants were also shown a mock-up demonstrating how a user could invoke the palette, and a mock-up showing the code that would be inserted once a selection had been made.

Before presenting each mock-up, we gathered information about the user's familiarity with the class and the strategies that they would likely use to instantiate the class under normal circumstances. For the Color class, the majority of participants indicated that they would look in the code completions menu (58.4%) or in the class documentation (19.0%) for a predefined constant. Another 14.0% indicated that they would use an external tool to determine the RGB values. Few participants were unfamiliar with regular expressions and no participants were unfamiliar with SQL queries which provides further evidence that our participants were not novice developers. Fig. **??** summarizes their usual strategies for instantiating regular expressions and SQL queries.

Participants were positive about our mock-ups. We asked the users to rate the usefulness of each mock-up palette, and their responses are summarized in Fig. **??**. For each palette, more than half of the respondents expressed that they would use it at least some of the time. Especially, people liked the regular expression palette a lot.

II. Design Implications

We analyzed the open-ended responses to extract design principles and constraints. We got 193 responses for the

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. 3. 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?"

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 extracted principles and constraints are summarized in the following subsections. The parenthesized number following each design constraint indicates how many people commented on that issue.

A. System Design Constraints

System design constraints are the criteria that should be taken into account in the design of the overall system architecture.

1) Handling Separation of Concerns (183?): Participants 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. Sometimes they explicitly suggested to add capability to handle the external resource files directly from the palette. 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 as opposed to the other two classes.

2) Reversibility (19?): Some 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 (41?): Many participants wanted to be able to configure the palettes or have the palettes maintain state even after the palettes are hidden.

4) Interaction with Code Context (13?): Some people wanted to interact with the code context. For example, 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

General palette design considerations are about how to make more usable palettes.

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

C. Suggestions from the Participants

At the end of the survey, we solicited suggestions from the participants about what types of classes could benefit from the active code completion palettes. 119 participants suggested various types of classes and we classified the suggestions into several categories.

1) 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. We address this issue in more detail in Section #.

2) 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 as in Juxtapose [?].

3) Query Languages (17): Similar to the regular expression and SQL palette we presented, the developers also wanted to check if the other types of query strings (e.g., XPath / XQuery) are correct or not by checking the results directly on the palette.

4) 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.

D. 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. It is error prone to use this class because it requires try/catch block, and also one must close the file after reading it. 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.

E. 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.

F. 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.