Designing a Programming Terminology Aid

Jeffrey Stylos
Computer Science Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213 USA
jsystilos@cs.cmu.edu

Abstract

A problem arising in many types of programming is that of learning the correct terminology and finding the correct API methods needed to accomplish a task. Most programming support tools are inefficient at supporting this type of learning, although Internet resources such as Google have emerged as useful new tools for this task. We propose a new tool that will analyze large repositories of text and code and use this to find the most relevant keywords and code for any given search query.

1. Introduction

A problem common across many types of programming tasks is that of learning to use an existing code interface, be it a library, an application programming interface (API), framework or language. This task involves several steps, which we have distilled from our observations of programmers [2]. Two of the first steps in this task are the related problems of learning the general terminology used by the code interface, and finding the specific names of the methods, classes or interfaces [1]. This is particularly a problem with new programmers and programmers new to a language or API, as they are less familiar with the specific conventions and less able to make informed guesses.

This is a problem not only in professional languages and environments such as Java programming using Eclipse, but encountered across nearly all textual programming or scripting tasks, from creating a simple Apple Script or Word macro to creating a complicated spreadsheet formula. For example, to place a 3 second delay inside a script, one has to know if the language or library names this a variant of pause, delay, sleep, wait or something else and whether it associates the method with threads, timers or other libraries before one can easily find the correct method to use. The answers to these questions are different for each different domain, so even experienced programmers encounter this problem when they use a new language or library, and alleviating this problem would make more types of programming accessible and effective to more programmers.

Existing programming help tools are inefficient at solving this problem. The three methods provided by modern programming IDEs and help systems are typically: name guessing, sometimes aided by code auto-completion; documentation searching, using a help system built into the IDE or online; and documentation browsing, in the help system or in online documentation. Each of these has problems that limit its usefulness. Guessing requires the ability to come up with terms close to the actual method or other code names, specifically on a prefix level when using auto-completion. Documentation searching has a similar problem of require knowledge of the correct terms. Documentation browsing, by way of a search tool index or list of classes or API methods avoids the problem of having to know the terminology ahead of time, but is slow, not scalable for larger APIs, and error prone as programmers often do not recognize a relevant name or spurniously think a seemingly relevant name does something it does not.

A solution we observed programmers using that avoided many of these problems was to search on the Internet using Google to find the correct terminology and specific method names, using only novice terminology and textual descriptions of a problem. Searching using Google achieved markedly different results than searching documentation using an IDE’s built in help. Programmers were able to find the answers they were looking for even when they did not know the correct terminology or method names, often because other programmers had described problems and solutions using the same “incorrect” terminology. These answers came from such sources as programming forums, tutorial sites and bulletin boards. Interestingly, most helpful were often the questions posted to these sites, perhaps because these
were more descriptive, or because they were more plentiful.

However, useful as it was, the programmers we observed still encountered problems and inefficiencies when using Google as a programming aid. Programmers spent much time reading the pages linked from the search results looking for the relevant portions of the page (though using the Google Cache page, which highlighted the search terms, helped). Also, many pages turned out not to be programming related or not to have useful information.

2. Proposed Solution

Based on our observations we propose a new tool to aid programmers in the tasks of learning new terminology and method names. We propose building an index that maps search terms to programming terminology and specific method names used by a particular language or API. Such an index would have to be very large to be useful, but we believe it can be automatically generated by mining large databases such as programming forums or even the entire Internet, or by building on top of services such as Google’s Web APIs.

A programming terminology index would have several useful applications. First, used directly (as a program that took as input naïve descriptions of functionality and output proposed terminology and method names for a particular domain), programmers would be able to more quickly find much of the same information that we observed them finding using Google. Second, an index could be embedded into the search functionality of local help tools, such as the Eclipse IDE’s help search. Using the index would make it easier for programmers to discover the right documentation even when they were unfamiliar with the correct terminology, without placing an additional burden on the documentation writers. Thirdly, the index could be used to modify the results shown on an Internet search engine such as Google, for example by displaying a sidebar along side the page results that included the most relevant terminology and method names.

To be most helpful, an index would have to be specific to a language or set of APIs, but the technique could be applied to many different domains and aid programmers with such varied tasks as C# programming or creating a JavaScript-enabled webpage. In particular, this tool would make the learning curve of all of these varied tasks shallower by helping overcome a common initial stumbling block and making it easier and faster for programmers new to the task to find the existing code interfaces they need.

To create such an index, a resource-intensive but fully automated approach could be to use word correlation across all or some pages of the Internet to find the words and code most correlated with any given search terms. A first step requires the detection of which programming languages are used on a particular page. Separating relevant word results from code results would also require recognizing which specific portions of a page are code.

In addition to direct correlations, it might be useful to take into account indirect correlations, such as the terminology associated with the code associated with search terms. This could help find correct terminology even if it is not often used alongside the search terms.

Access to the search logs of popular search engines or search tools might also provide useful information. Finding correlations in the successive search terms of individual users could be used to supplement the index. For example, observing that users whose search query contains pause frequently have queries that contain sleep shortly afterward could be used to associate the second word with the first.

Finally, a form of explicit user input could be used to further build the index. Whether inside an IDE, search engine or programming forum, programmers might be willing to explicitly provide information for an index, associating the terminology they think most relevant with a particular method, in much the same manner that wikis allow users to add content to open websites.

3. Proposed Evaluation

When testing different algorithms to build a programming terminology index, an important concern is that of how to measure the quality and usefulness of the index. However, this is expensive to measure as it requires testing each version of the tool with real programmers. It may be possible to use a fixed set of observations to create benchmarks that can approximate the usefulness of an index automatically. For example, by distilling programmers’ tasks into questions with a single input (their initial search terms) and a single correct answer (the code they ended up using), we can measure if and how highly different indexes return the correct answer for a set of observed questions. This is an oversimplification of the task, as the programmers we used often took an iterative approach to finding terminology and code names, but it might be a cheap and useful approximation.

4. References